A Model to Select the Right Infrastructure Abstraction for Service Function Chaining

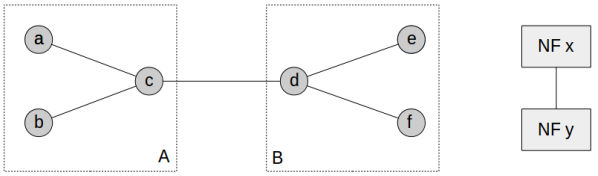
Thomas Soenen*, Sahel Sahhaf*, Wouter Tavernier*, Pontus Sköldström†, Didier Colle* and Mario Pickavet*
*UGent - iMinds: {thomas.soenen, sahel.sahhaf, wouter.tavernier, didier.colle, mario.pickavet}@intec.ugent.be
†Acreo: {pontus.skoldstrom}@acreo.se

***Abstract*—Through network function virtualization (NFV), telecom providers aim to flexibly re-use generic-purpose hardware to provide services on-demand and in an agile way. Service function chaining is becoming the preferred model to describe the characteristics of the packet-processing network functions which, together, form these services. NFV allows for network function embedding freedom, creating new dynamics between providers and the users requesting services. Users want this freedom to optimise the performance of their requested services, while providers aim to optimise their resource cost with it. This trade-off is heavily influenced by how the available infrastructure is exposed to the users. In this paper, we present an infrastructure abstraction model for network, compute and storage resources that exposes the infrastructure in an abstracted manner. We use this abstraction to propose a solution for the placement freedom trade-off problem by studying its relation with metrics that capture both the user's and the provider's aspects. We conclude with a heuristic that determines the right abstraction for particular scenarios.**

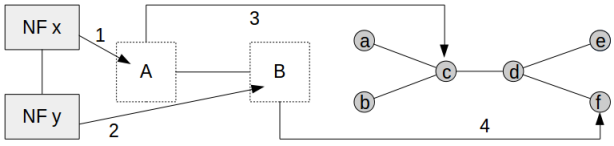
I. Introduction

Network function virtualization (NFV) allows telecom operators to implement network services as software applications and to deploy them on-demand on any generic-purpose device in their network. Since the location where a service is running is no longer predefined, it can be chosen so that it optimises a certain metric. Service Function Chaining (SFC) [1] is a network service concept that further increases this mapping flexibility by splitting the network service up into different modules, called Network Functions (NFs), where each NF can run on a different device. A service graph describes how the NFs are chained together to construct the network service. Multiple research groups/projects are focusing on SFC to address its limitations, like: i) the NFV group in ETSI [2], ii) a dedicated working group (Service Function Chaining Working Group) in IETF [1], [3] and iii) UNIFY [4] which is an EU-funded FP7 project.

Using this mapping flexibility, users desire to optimise the performance of their requested services by deciding themselves how the NFs are mapped to the infrastructure. On the other hand, providers want to map the NFs to optimise the cost of their resources. In this paper, we propose to combine both perspectives by providing the users with an abstract view of the infrastructure. The users map their NFs on the abstract view, giving them some degree of freedom. After this embedding is done, the provider maps the result of the user embedding onto the exact infrastructure. This process, together with the infrastructure abstraction, is shown in Fig. 1. The mapping



(a) A network consisting out of 6 devices (e.g. a server, a switch and a router), partitioned into two abstract nodes A and B (left), and a network service consisting out of two NFs x and y (right).



(b) The user places the network service by mapping NF x on abstract node A (1) and NF y on B (2). The provider refines this placement by mapping NF x on c (3) and NF y on f (4).

Fig. 1: Two phased mapping with infrastructure abstraction.

freedom that remains for both users and providers depends heavily on the way that the infrastructure is abstracted. A high abstraction leaves few options for the users and many for the providers, and vice versa.

Next to managing mapping freedom, infrastructure abstraction has some other advantages for the providers. It prevents exposing the exact structure of the topology, allowing business secrets to be kept hidden. Secondly, not all changes in the infrastructure cause an update of the abstracted view. Therefore, the information share rates that provide users with the latest state of the network are lower. Also, users do not want to be tasked with the exact placement of the requested NFs. Which device inside a datacenter runs the NF is of no value to the user. Therefore, it makes sense to represent racks of devices, server rooms or entire datacenters in an abstracted manner. Lastly, computing the optimal mapping of NFs in the infrastructure is NP-hard [5]. When the size of the network and the amount of NFs increases, placement becomes a time consuming task. Therefore, outsourcing parts of it to the user reduces the duration of the placement phase.

Exposing an abstract view to the user holds some trade-offs. Increasing the abstraction makes the infrastructure more opaque, averaging out its best assets (fastest connections, most powerful servers, ...). This can refrain users looking for these high-end assets from making requests. Also, the abstraction could reduce the mapping freedom of the users or the providers

too much. With these stumbling blocks in mind, solving the placement freedom trade-off problem comes down to defining the right infrastructure abstraction.

Our contribution. In this paper, we solve the placement freedom trade-off problem by using the concept of infrastructure abstraction. We present an infrastructure abstraction model for network, compute and storage resources to generate a range of abstraction levels. We showcase this model by using a partitioning algorithm to create different abstracted views of the infrastructure and by analysing these views with respect to the following metrics: bandwidth, delay, stability, similarity and acceptance rate. Finally, we propose a heuristic that enables network providers to select the right infrastructure abstraction for particular scenarios.

The paper is organised as follows. Section II contains an overview of the related work, followed by Section III which explains the proposed model for infrastructure abstraction. To demonstrate our model, we construct an abstracted view in Section IV and Section V includes the analysis of these views. Section VI describes the heuristic for selecting the right infrastructure abstraction and Section VII lists the future work for this domain. Finally, Section VIII concludes the paper.

II. Related work

To the best of our knowledge, the problem of finding optimal infrastructure abstractions remains unexplored. Existing work mainly focused on finding an optimal mapping when limited information on the infrastructure is available. In [6], authors proposed a hierarchical embedding in which an abstract view of the physical network is provided. They evaluated the impact of exposing more information on the performance of their proposed embedding algorithm. In [7] a request is split among infrastructure providers with respect to an abstract view of the underlying (AS-level) network. However, no discussion on possible trade-offs is present. In the scope of this paper, we will need graph partitioning algorithms to construct the infrastructure abstraction. In [8], the authors propose a heuristic for the community partitioning algorithm that scales good for bigger networks. In [9], a network is partitioned by using k-clustering. It demands that within each group of nodes, all nodes are separated by k or less hops. In [10] and [11], k-means clustering is described. In k-means clustering, the nodes are partitioned in k groups, by minimising the total distance between nodes and the center of their groups.

III. Unified infrastructure abstraction

We represent the exact infrastructure of a network as an undirected graph. Each physical device (e.g. servers, routers, switches and databases) in the network that provides computing power, storage capabilities or network forwarding functionality is represented as a node. These nodes have attributes to describe their characteristics, which include CPU power, number of ports, available memory, etc. The physical connections between these devices are represented as links, also with attributes describing bandwidth, delay, etc.

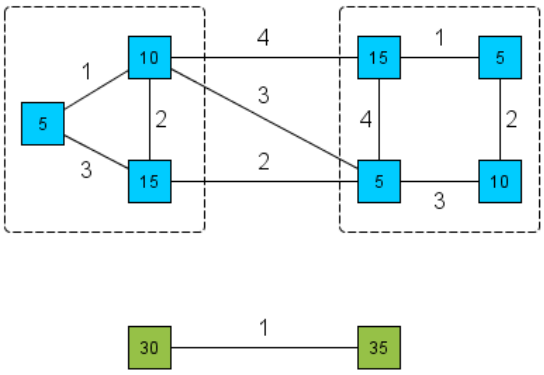


Fig. 2: Example of abstract node and link creation. The number inside each node indicates its CPU power while the numbers on the links show their bandwidth capacity. The abstract view of the topology is shown at the bottom. For this abstract view, the resources of the nodes were aggregated and for the links we calculated the worst-case bandwidth.

We abstract this infrastructure by combining multiple nodes into an abstract node. An abstract link is used to connect two abstract nodes if a physical link connects physical nodes between them. This is shown in Fig. 2. A partition of a set of nodes (or a graph) is a set of groups, where every node from the original set is part of exactly one of these groups and a group is again a set of nodes. Therefore, each possible abstraction of the infrastructure represents a partition of the nodes.

Determining the attributes for nodes and links in the exact infrastructure is straightforward. Less straightforward are the abstract nodes and links, since they need to describe the resources inside them. For the abstract nodes, the best way to describe the resources seems to be aggregating them. This would expose the total CPU power, available memory, available operation systems, etc., of the abstract node. This abstraction of information might have negative effects on the acceptance rate. A user might request to deploy a NF on this abstract node that needs 60% of its available CPU power. If the CPU power of the abstract node is the aggregation of the CPU power of 5 nodes that each contribute 20%, this request will be rejected. This could be avoided by adding an indication of average CPU per node as attribute to the abstract node, but this would imply exposing more information to the user.

A similar situation arises when describing the attributes of the abstract links. Exposing the average bandwidth of all the links within this abstract link hides the strongest ones. If one of the links allows for 100 MB/s and the other 9 links for 10 MB/s, a user looking for a 50 MB/s connection will be thrown off by the exposed average bandwidth of 19 MB/s. Therefore, the way the attributes of the abstract nodes and links get calculated should be considered when constructing the abstraction.

Next to the partition of nodes that leads to the abstracted view and the way the attributes get combined, a third part of

the infrastructure abstraction model is the embedding algorithm that is used by the provider. This embedding algorithm maps the NFs from the abstracted view to the exact infrastructure, after the user has mapped them on the abstracted view. Since the embedding algorithm influences the acceptance rate of the requests, it should be part of the infrastructure abstraction model, since a higher acceptance rate might be the deciding factor when selecting the right abstraction. In [12], the authors give a thorough survey of the available embedding algorithms.

Different partitions of the nodes and different ways of calculating the attributes of the abstract nodes and links allow for different infrastructure abstractions. Since nothing forces us to have only one abstraction, it is possible to customise abstractions per user. For example, over-the-top providers like Netflix that use the NFV network of a provider are strongly interested in which datacenters their functionality is mapped to and how fast the connections between these datacenters are, while other customers do not care for this information. These user preferences should be used by the providers to customise the infrastructure abstractions that are shared with the users. This concept is shown in Fig. 3.

In the next section, we will demonstrate this model by creating some abstracted views.

IV. Abstraction construction

A. Problem description

In this paper, we investigate the placement freedom trade-off problem and propose a solution by using infrastructure abstraction. Both users and providers desire optimal freedom in mapping NFs to the infrastructure. By abstracting the infrastructure, the users get some freedom to place the NFs on this abstraction, while the providers get freedom in mapping that result on the exact infrastructure. The detail of information (i.e. the level of abstraction) that is exposed to the user relating the resources of the infrastructure, impacts service request acceptance significantly. While the network providers prefer to reveal as little information as possible (to keep the network hidden and to claim most of the placement freedom), higher levels of abstraction might lead to failing request embeddings. This is a consequence of the discussion regarding attributes of abstracted nodes and links in Section III: when resources are aggregated in an abstract node, users might expect the nodes behind this abstract node to have more resources than they in fact have. On the other hand, low levels of abstraction give away more details of the network than the provider desires, and reduces the provider's mapping freedom. Therefore, solving the placement freedom trade-off problem comes down to determining which is the right infrastructure abstraction, by considering these aspects. We quantify the aspects that influence the problem with the following metrics:

- Acceptance rate, the number of accepted requests divided by the number of made requests.
- Similarity, an indication on how similar the abstracted and exact view are. Section V contains a formula for similarity.

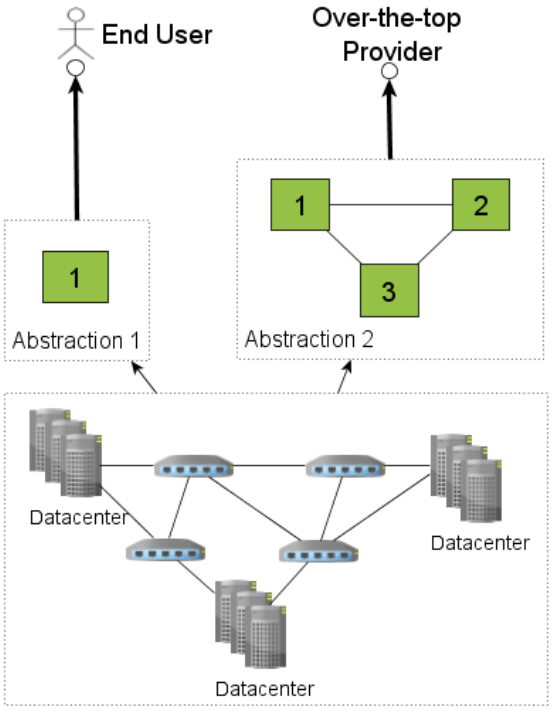


Fig. 3: Example of different infrastructure abstractions. A more detailed view can be given to an over-the-top provider while an end user who does not care about the details of the infrastructure is given a more abstract view. The nodes with numbers 1, 2 and 3 on the top right side represent the abstraction of the datacenters while for the end user on the left side, an abstract view composed of a single node is provided.

- Stability, an indication on how the abstracted view alters by changes (failures, updates, etc.) made to the exact topology. Section V contains a formula for stability.
- Bandwidth of the abstracted links.
- Delay of the abstracted links.

More specifically, the acceptance rate is an indication of how much of the made requests get accepted and how much of them get rejected. This is an important metric for both users and providers, since both want to see as much requests embed as possible. Similarity offers a measurement of how much the abstraction and the exact infrastructure look alike. This is a relevant metric, since the providers wish to hide the business secrets and the weak spots that their networks contain. Stability indicates how much changes must be made to an abstracted view when the exact topology of the network changes. The metric helps the providers in determining an abstraction which requires low updates to the users. The bandwidth and the delay of the abstracted links indicate how appealing the abstraction looks to the users. The less appealing, the less requests will be made. This forces the provider to look for an abstraction with higher bandwidths and lower delays. These metrics are studied in Section V. Eventually, their study and analysis helps us in defining a heuristic that balances the placement freedom

trade-off problem by proposing the right abstraction out of a set of abstractions.

Formally, given a physical infrastructure modelled as an undirected graph $G_p = (N_p, L_p)$ composed of physical nodes (N_p) connected via physical links (L_p), assuming each node has certain capacity in terms of computation (C), memory (R) and storage (S) and links have delay (D) and capacity in terms of bandwidth (BW), we calculate the right abstraction.

B. Abstraction construction approach

To be able to select the right abstraction, we first need a set of abstractions to select from. In this paper, we abstract an infrastructure with the following steps:

1) We partition the infrastructure with a partitioning algorithm.
2) Clusters of nodes from the partitioning become abstract nodes in the new topology.
3) Two abstract nodes get an abstract link if any two nodes, one from each cluster, are physically connected.
4) The resources of the infrastructure are combined into the abstract nodes/links.
5) We select an embedding algorithm to map the NFs on the exact infrastructure.

In Section III, we explained that an abstraction is linked to a partition of the nodes. Therefore, by partitioning the nodes by using some partitioning algorithm, we take the first step towards an infrastructure abstraction. All the nodes in the same group get combined in the same abstract node, and every pair of abstract nodes in which two nodes, one node of each abstracted node, are connected by a link in the exact infrastructure is connected by an abstract link.

As explained in Section III, an infrastructure abstraction is also characterised by how the attributes of the individual nodes and links are combined into attributes for the abstract nodes and links. For the abstract nodes, we decided to aggregate the available C, R and S of all the nodes that are part of the abstract node. We decided to add no extra information on how many nodes are part of the abstract nodes and what the average resources per node are to prevent sharing too much detailed information of the infrastructure with the users.

For the resources of the links, we chose to assign to each abstract link a BW that represents the worst-case bandwidth between the two abstract nodes that the link connects. This value is found by iterating over every pair of original nodes, one from each abstract node that is connected to the abstract link and determining what the bandwidth is along the shortest path (in hopcount) between them. The lowest value of all these pairs is the bandwidth of the abstract link. Fig. 2 shows an example of worst-case bandwidth. The delay of the abstract link is the worst-case delay between the two abstract nodes that it connects and is calculated in a similar way. The idea behind these worst-case values is that they provide the users with the minimal performances they can expect from that link. If we were to show average bandwidths and delays, or best-case values, a user deciding to use this abstract link might get assigned a physical link with a worse performance than the values shown in the abstracted view. This could cause users to terminate their services premature. On the other hand, showing the worst-case values decreases the number of requests, since it makes the network look less appealing.

As embedding algorithm, we opted to use the one proposed in [13]. This algorithm embeds the NFs by minimising the resource usage, while maintaining the QoS demands for the links between the NFs. This implies that, to embed a NF, the algorithm first looks to add this NF to a node that is already running some NFs, before considering embedding it to a node that is in the idle state. We chose this particular embedding algorithm as [13] states that it maximises the number of accepted requests and thus the acceptance rate.

In the following section we describe a partitioning algorithm, so we can create some abstracted views.

C. Community partitioning

According to Section IV-B, we need a partitioning algorithm to start constructing infrastructure abstractions. We expect abstract nodes that contain well-connected infrastructure nodes to lead to abstract links with decent (worst-case) bandwidth and delay rates, since intra-cluster links will not significantly decrease their values. We also expect it to improve the performance of groups of NFs or entire services that get assigned to the same abstracted node, since their NFs will be well-connected. Based on this, we chose a community partitioning algorithm to demonstrate our model, since it partitions a network in well-connected groups. The Louvain method [8] seems like a good implementation of a community algorithm to start from, since it is a heuristic that scales well.

The Louvain method divides the nodes of a network into communities (i.e. clusters), which are groups of nodes that are well-connected. In [8], the authors define the metric *modularity* (M), which is a scalar that compares the intra-cluster connectivity to the inter-cluster connectivity of a partitioned topology. A higher M indicates that the density of the links between nodes inside a cluster compared to the density of the links connecting nodes not in the same cluster is higher than for a partition with a lower M. The Louvain method iterates over possible partitions by moving nodes, one by one, from their own cluster into the cluster of one of their neighbours, if this increases M. The iteration is stopped when it is no longer possible to make a gain in M by moving a node. At this point, the Louvain method reaches a first community partitioning. In a second phase, this process is repeated, but instead of moving separate nodes to the community of neighbouring nodes, entire communities are moved towards neighbouring communities. Iterating this until no more gain in M (the original nodes are still used to calculate M) can be found results in a second, higher level partitioning. This process is repeated until it is no longer possible to start a new phase, as every migration of a community leads to a decreasing M.

The Louvain method returns a set of partitions that resemble local maxima in M. The number of communities for each partition is not predictable. Since we desire to evaluate different abstraction levels, we extend the Louvain method so we

get a more predictable set of abstraction levels. By starting from the lowest-level partition (i.e. the one with the most communities), we merge those two communities of which the merger results in the highest gain in M, on the condition that these communities are in the same cluster in the higher-level partitions. After this step, we have a new partition with one less community than the previous partition. Repeating this step until only one community remains, results in a set of abstraction levels, one for each number of communities smaller than the amount of communities in the lowest level partition returned by the Louvain method. This gives us a bigger set of abstractions to choose the right abstraction from.

V. Experimental analysis

As stated in Section IV-A, quantifying the aspects that influence finding the right infrastructure is done by using the metrics : (i) acceptance rate, (ii) similarity, (iii) stability and (iv) bandwidth and (v) delay of the abstract links. In this section, we will demonstrate how to evaluate their dependence on the abstraction level. The evaluations were done for different infrastructure topologies, to investigate the influence of topology characteristics on the metrics.

Our experimental environment, that we use for the acceptance rate evaluation, is based on Python code. The Networkx library is used for graph-based implementations. For the community partitioning algorithm we extended the library associated with [8]. To evaluate the impact of a varying network degree (3, 4 and 5) and size (100, 200 and 300 nodes), we chose to use random regular graphs [14] as infrastructure topology. The capacity of the nodes (C, R, S) and the BW of the links are numbers uniformly distributed between 100 and 300. The D of the links is uniformly distributed between 10 and 50. As infrastructure abstraction, we consider the partitions resulting from our extension of the Louvain method (See Section IV-C), where C, R and S are aggregated over all nodes in an abstract node. The abstract links get assigned a worst-case BW and D, calculated like explained in Section IV-B. The service requests are represented as directed graphs (NFs as nodes and connections as links). The number of NFs in a service request is generated randomly between 5 and 10, and each pair of nodes is connected with a 0.5 probability. The resource demands of the NFs and the links are numbers uniformly distributed between 1-50. The delay on the connections should be lower than 1000. The requests arrive in a Poisson process with average rate of 4 requests per 100 time units and have exponentially distributed lifetimes with an average of $\mu = 1000$ time units. Each scenario runs for 10000 time units, is iterated 20 times and the results are averaged.

A. Bandwidth and delay

The first experiment evaluates the impact of different abstraction levels on the theoretical calculated bandwidth and delay of the abstract links. Since these metrics are used by users to determine their mapping, it can lead to rejected requests if the network can't give this performance. If too low, they can also refrain users from making requests to the provider in the first place. The bandwidth of an abstraction link l_a is defined as

$$BW(l_a) = min(BW(l_p), \forall l_p \in P, \forall P \in Paths), \quad (1)$$

where $Paths$ is the set of shortest paths (see further) for each pair of nodes, where we consider every pair of nodes that contains one node from each abstract node connected by the abstract link. P is the set of links that form such a shortest path. The delay of a link is defined as

$$D(l_a) = max\left(\sum_{\forall l_p \in P} D(l_p), \forall P \in Paths\right). \quad (2)$$

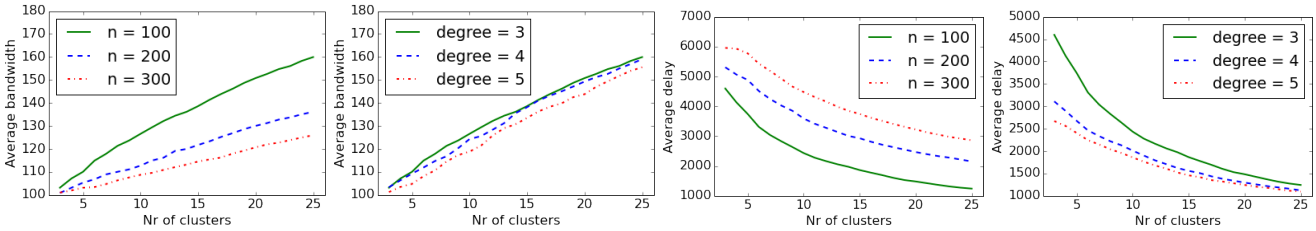
For calculating the elements in $Paths$, different algorithms can be used. We considered: i) shortest path (in hopcount) and ii) least-cost path algorithms. In the latter, we chose to use the reverse of the link bandwidth as cost for each link. This favors the least-cost paths to have high bandwidths. However, the results indicate that the difference in the resulting bandwidth and delay between these two algorithms is negligible. Therefore, Fig. 4 illustrates the results of only the shortest path algorithm. Fig. 4 shows the calculated bandwidth and delay, averaged over all the abstract links in the abstract view, averaged over 20 iterations for every infrastructure topology. As we see, increasing the abstraction level leads to a lower bandwidth (Fig. 4a, 4b) and a higher delay (Fig. 4c, 4d). When keeping the number of clusters constant, we see that larger networks have worse bandwidths and delays while a lower delay in networks with higher degree is achieved. The impact of degree on bandwidth is insignificant.

B. Similarity

Since providers prefer to reveal as little information on the infrastructure as possible (to keep business secrets and weak spots hidden), we would like an indication on how similar an infrastructure abstraction is to the original topology. Therefore, we introduce *similarity* as

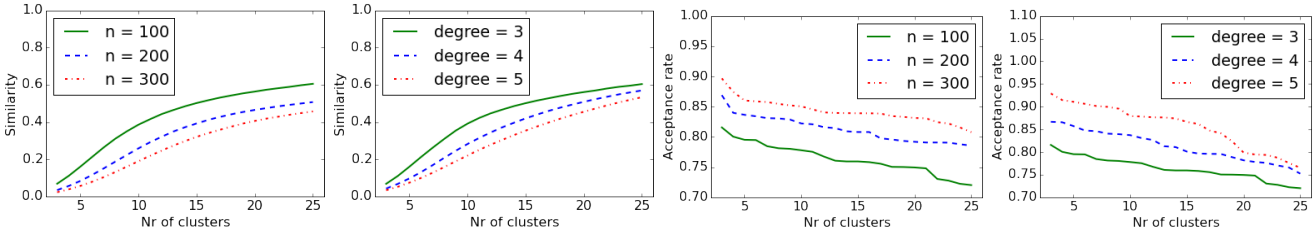
$$E(A, G) = \frac{1}{2}\left(\frac{\|N_a\|}{\|N_p\|} + \frac{\|L_a\|}{\|L_i\|}\right), \quad (3)$$

where A is the abstraction, N_a is the set of abstract nodes, N_p is the set of original nodes, L_a is the set of abstract links and L_i is the set of infrastructure links that are connecting abstract nodes. In other words, L_i contains all the links that have been combined into abstract links. The equation contains an indication on the change in nodes, the change in links and is normalised, so that two identical topologies have $E = 1$. The more nodes (links) that are combined in (between) clusters, the more difficult it is to reconstruct the original topology and the lower E will be. This is shown in Fig. 5a, where the *similarity* decreases when the average cluster size increases (decreasing number of clusters) and when the number of clusters is kept constant, while the number of nodes increases. Fig. 5b shows that the *similarity* decreases when the degree increases, as the ratio *connected clusters* to *inter-cluster links* decreases.



(a) degree 3, n=number of nodes (b) 100 nodes, different degree (c) degree 3, n=number of nodes (d) 100 nodes, different degree

Fig. 4: Average bandwidth and delay evaluation



(a) degree 3, n=number of nodes (b) 100 nodes, different degree (c) degree 3, n=number of nodes (d) 100 nodes, different degree

Fig. 5: Similarity and acceptance rate evaluation

C. Acceptance rate

To evaluate the impact of different abstraction levels on the acceptance rate we use the experimental environment described earlier in this section. When the requests arrive, they need to be mapped on the abstraction to simulate the user's behaviour. If this first phase gets accepted, NFs mapped to an abstracted node must be mapped on the nodes inside this abstracted node and the connections that are mapped on the abstracted links must be mapped on the links. For both mapping phases, we need to use an embedding algorithm. We opted to use the embedding algorithm proposed in [13] for both phases. As explained in Section IV-B, this algorithm embeds the NFs by minimising the resource usage, while maintaining the QoS demands for the links between the NFs.

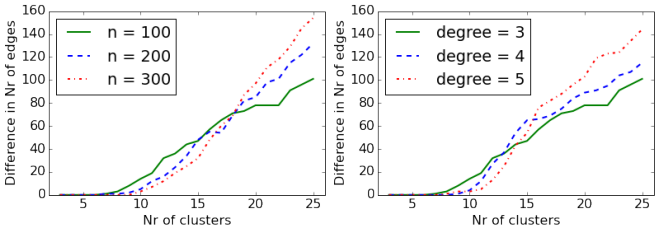
If the first phase is able to place the request on the abstracted view and the second phase can place this on the infrastructure, the request is accepted. If there are not enough available resources to place a request in any of the two phases, it is rejected. Next to the average bandwidth and delay of the abstracted links, the way the attributes of the abstract nodes and links are calculated has in influence on the first phase of the embedding. The lower these attributes are, the more request will be rejected in this first phase.

Figs. 5c and 5d show the acceptance rate of this embedding for different abstraction levels. Both a network with a higher degree and one with more nodes perform better when it comes to acceptance rate. We also see that a higher abstraction level leads to a higher acceptance rate. The reason is that, for a higher abstraction, the first phase can choose between abstracted nodes with higher resources, and the second phase of the embedding has more options, since the abstract nodes contain more original nodes. We expect this trend to reverse when the abstraction approaches the exact infrastructure, since the acceptance rate of embedding directly on the infrastructure, without an abstraction in between, should be higher than the case with an abstraction in between.

The behaviour of the acceptance rate seems to conflict with the trend that bandwidth and delay get worse when the abstraction increases. However, more factors are playing here. Although the bandwidth decreases with increasing abstraction level, it stays above the range of bandwidth requirements of the links in the requests. Therefore, the trend of the bandwidth has no real influence on the acceptance rate in this experiment.

D. Stability

Since the abstraction hides some details, not every change in the infrastructure leads to a change in the topology of the abstracted view. If a link or a node fails, the attributes of the abstracted link or node they belong to will change, as its total or averaged resources change. This is a change that has a low priority when it comes to urgency in sharing it with the users, since the remaining resources can handle the requests. However, if the failure causes a change in the abstracted topology, this needs to be shared with the users. Since the abstract topology changed, it is possible that the failed node or link were directly exposed in the old abstract topology (an abstract link/node representing one link/node). To prevent the users from mapping on links or nodes that are no longer available and thus, to prevent rejected requests, the topology update must be shared with the users. In what follows, we call this robustness against topology changes the stability S. Increasing S means that the abstraction becomes more robust against topology changes. To evaluate the impact of different abstractions on the stability, we calculated the number of changes made in the abstracted topology caused by 10% of links failing in the infrastructure. These changes can be the addition or the removal of abstracted links, since we use the number of abstracted nodes as reference frame. Fig. 6 illustrates the number of changes for each abstraction. The trend of increasing instability for increasing number of clusters becomes steeper for larger networks or for a higher degree. We define the numerical value of stability as the inverse of the



(a) degree 3, n=number of nodes (b) 100 nodes-different degree

Fig. 6: Stability evaluation

number of changes made in the abstracted topology caused by 10% of links failing in the infrastructure.

VI. Abstraction selection algorithm

The experiments in Section V show how the level of abstraction impacts the different metrics. In this section, we propose a simple heuristic to enable a provider to select a suitable abstraction so that different aspects of the problem are factored in.

We define a cost function which combines the factors i) similarity, ii) average delay, iii) average bandwidth, iv) acceptance rate and v) stability as

$$C(a) = \alpha f_E(a) + \beta f_D(a) + \frac{\delta}{f_S(a)} + \frac{\gamma}{f_{BW}(a)} + \frac{\kappa}{f_A(a)}. \quad (4)$$

In this cost function, α, β, δ, γ and κ are parameters that allow the provider to tune the impact of each factor according to his/her preference. $f_E(a)$ is the impact of the similarity on the cost function for abstraction a. Assuming that there are multiple ways of abstracting the physical topology (i.e. other partitioning algorithms, other ways of combining attributes and other embedding algorithms), the cost of each abstraction is calculated and the one with the minimum cost is selected and sent to the user. Profiling of the requests allows a provider to select a better abstraction. If more delay-sensitive applications are requested, the parameters in the cost function can be tuned to favor abstractions with a lower average delay.

VII. Future work

Section V shows that the abstraction has an influence on a range of metrics. The impact of other abstractions on these metrics can be the subject of further research. To obtain other abstractions, other partitioning algorithms (e.g. k-clustering and k-means clustering), different ways to combine resources into abstracted nodes and links (e.g. best-case values and average values) and different embedding algorithms ([12]) can be considered. The influence of other embeddings on the acceptance rate can be studied, together with the cause of the rejections (lack of link/node resources). These topics can add knowledge to improve the abstraction selection process.

VIII. Conclusions

In this paper we provide a guideline for abstracting infrastructure in the context of service function chaining. When abstracting infrastructure, a spectrum of options and a resulting trade-off becomes apparent. One end of the spectrum gives providers total freedom in mapping the services, enabling them to optimize their resource cost. On the other end, users are given full access to the infrastructure, enabling them to make as many placement decisions as possible. We proposed a mechanism to generate a range of infrastructure abstractions, as well as a number of metrics to quantify the referred trade-off. These include the acceptance rate of made requests, the similarity of the abstraction with the actual infrastructure, the stability and the bandwidth and delay of the abstract links. The behavior of these metrics was analysed in a number of controlled setups to validate trends and relationships. Increasing the abstraction level induces a positive impact on the stability as well as on the acceptance rate. At the same time, increasing the abstraction level reduces the similarity. In general, the acceptance rate increases when the network increases in size or degree, while the stability increases more quickly when the size or the degree is higher. To enable providers to generate infrastructure abstractions according to desired properties, we proposed a heuristic which minimises a weighted cost function of the analysed metrics. This gives providers a concrete tool to make well-informed trade-offs and decisions when exposing their network to users for service function chaining.

Acknowledgement

Conducted within the FP7 UNIFY project framework, the Horizon 2020 and 5G-PPP SONATA project framework and partially funded by the UGent BOF/GOA project 'Autonomic Networked Multimedia Systems'.

References

[1] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," IETF RFC 7665, nov 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7665.txt

[2] ETSI, "Service chaining for nw function selection in carrier networks," feb 2014. [Online]. Available: https://www.etsi.org

[3] T. Nadeau and P. Quinn, "Problem statement for service function chaining," IETF RFC 7498, nov 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7498.txt

[4] A. Csaszar et. al., "Unifying cloud and carrier network: Eu fp7 project unify," in *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, Dec 2013, pp. 452–457.

[5] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 7–13.

[6] I. Vaishnavi et. al., "Recursive, hierarchical embedding of virtual infrastructure in multi-domain substrates," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–9.

[7] I. Houidi et. al., "Virtual network provisioning across multiple substrate networks," *Computer Networks*, vol. 55, no. 4, pp. 1011–1023, 2011.

[8] V. D. Blondel et. al., "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[9] Y. Fernandess et. al., "K-clustering in wireless ad hoc networks," in *Proceedings POMC*, ser. POMC '02. ACM, 2002, pp. 31–37.

[10] J. Hartigan and M. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[11] T. Kanungo et. al., "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, Jul 2002.

[12] A. Fisher et. al., "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, pp. 1888–1906, Fourth 2013.

[13] S. Sahhaf et. al., "Network service chaining with optimized network function embedding supporting service decompositions," *Computer Networks*, 2015.

[14] N. C. Wormald, "Models of random regular graphs," *London Mathematical Society Lecture Note Series*, pp. 239–298, 1999.