2019 IEEE Wireless Communications and Networking Conference (WCNC)

Service-Aware Virtual Network Function placement in Software Defined Networks

Elin Sundby Boysen
University of Oslo, SINTEF Digital
Email: elinsundby.boysen@sintef.no

Joakim Flathagen
Norwegian Defence Research Establishment (FFI)
Email: joakim.flathagen@ffi.no

Josef Noll
University of Oslo
Email: josef.noll@its.uio.no

***Abstract*—Current mobile units have multiple network interfaces and can connect to multiple access points simultaneously. This represents new opportunities. Choosing the right network, and the optimal route through that network, requires knowledge about the state of the network. With Software Defined Networking where control plane and data plane are separated, SDN controllers can have a good overview over a wide variety of network parameters. Thus, traffic from a service with specific *service parameter requirements* can be routed so that these requirements are likely to be met. In this paper we have investigated how the optimal placement of a network function in SDN varies based on service parameters such as latency, price, packet loss and throughput. We present a method to choose the best placement based on a service profile according to the current network state. The results are validated by implementing the methods in an emulated SDN network.**

I. Introduction

Mobile units that can connect to multiple access points simultaneously (as illustrated in Figure 1) represent new opportunities. One example scenario where this can be useful is for public transportation such as ferries and tourist buses where the transport company provide Internet access to its passengers. Network availability, capacity, speed or quality will vary throughout the journey, but can be improved using multiple access points simultaneously. Services such as video streaming or delay-sensitive applications (interactive games) have different requirements with regard to parameters such as security, reliability, bandwidth, packet loss, latency or even cost. In the rest of this paper we will refer to these parameters as *service parameters*.

Choosing the right network, and the optimal route through that network, requires some knowledge about the network's state. With the introduction of Software Defined Networking (SDN) [1] where control plane and data plane are separated, SDN controllers can have a good overview over the situation in their own network. Thus, the traffic of a service with specific service parameter requirements can be routed so that these requirements are likely to be met.

Available networks and their parameters will vary over time. For our mobile unit in the example scenario, one or more of the following situations can occur:

1) From the available networks available, one is considered the most likely to meet the service's parameter requirements. The SDN controller sets up an end-to-end path between the ambulance and the hospital.

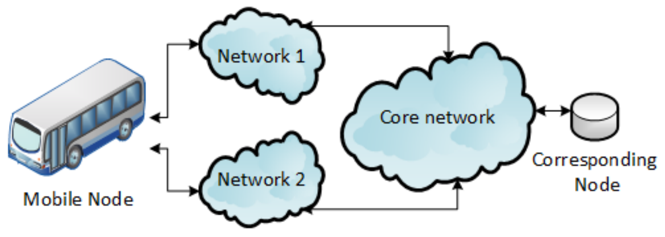


Figure 1. Target scenario

2) Due to mobility, a running session between ambulance and hospital is experiencing degrading quality, and another available network can be accessed. However, even though the SDN controller has information on the general state of this network, the access link quality should be assessed through a make-before-break handover. Such a solution is described in [2].
3) Capacity requirements for a given service that is needed between the ambulance and the hospital cannot be met by any single available network, but by a combination of them. Thus, the mobile node connects through more than one accessible network, and a traffic splitter function is required to route traffic via the two or more paths connecting to the ambulance.

The first of our three situations is a routing problem where [3] and [4] are among many offered solutions. The focus of this paper lies on situations 2) and 3), where both require a functionality somewhere in the network that can duplicate packets or split packet streams. The physical placement of this functionality would until recently have been restricted by the service providers' server installations, and thus represented a trade-off between service latency and installation costs. Now, SDN in combination with Virtual Network Functions (VNF) has the potential for service providers to meet the end users' service requirements [5]. As the SDN control plane consists of one or more SDN controllers that has a global view of the entire network it rules, the SDN controller can be contacted by SDN applications that present their requirements of network resources or network behaviour. VNFs - the process of moving network services away from dedicated hardware into a virtualized environment - are complementary to SDN. SDN in combination with VNFs, makes it possible to establish on-demand network functions such as the functions we describe in our example situations 2) and 3), in optimal distance to the

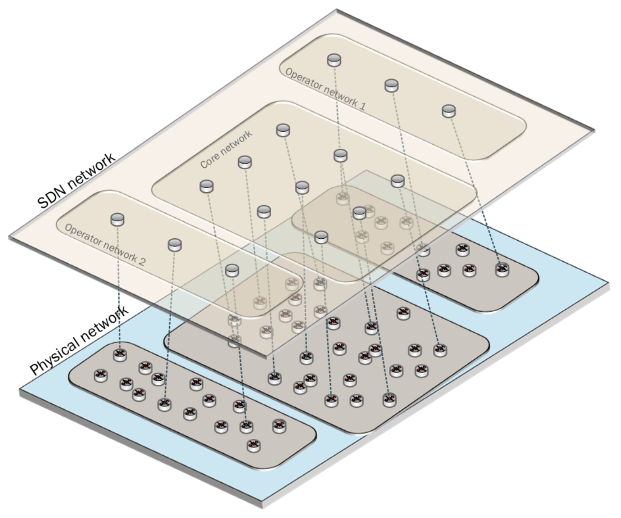


Figure 2. SDN network with one core network and two operator networks

user, and to update routing schemes and placement of network functions according to varying user needs or network state. A service provider can utilize a set of nodes and establish an overlay to connect these nodes. SDN provides the functionality to create such an overlay on top of the physical network, as depicted in Figure 2.

In this paper we have investigated how the optimal placement of a network function in SDN vary based on service parameters and present a method to choose the best placement based on the user's choice. The results are validated by implementing the method in an emulated SDN network.

The rest of this paper is organized as follows. In section 2 we present related work and provide a background for our contribution. Section 3 describes our target scenario and requirements, before we present our virtual network function and our algorithms for optimized virtualized network function placement based on metric-specific weight functions. Our prototype implementation is described in section 4, and evaluation and analysis of the results are presented in section 5. In section 6, we conclude and give some prospects for future work.

II. Related Work

The first approach to address mobility for Internet applications was Mobile IP. Since the introduction of Mobile IP (MIPv6) in 2002, various enhancements have been proposed [6], [7], [8] to mitigate high signalling overhead, long latency or packet loss. However, they all require some form a mobility anchor, either as centralized function or as functions in the local networks. The placement of this anchor function is based on implementation on physical routers and the QoS experienced by the end user cannot be altered by the anchor function itself. Even with the improvements proposed by PMIPv6, QoS suffers during handovers. Kukliński et al. [9] have considered the following aspects of SDN as beneficial for mobile services: *Direct operations on dataflows, handover that can be based on multiple criteria, no need to change IP address of the mobile node and faster handover operations.* As SDN can make it possible to dynamically establish a network function that can act as an anchor point, its placement can be dynamic and thus changed according to current requirements and network state. This is discussed in [10], but their solution does not take the combination of service and user requirements fully into consideration as we do here.

Several works that discuss virtual network placement are grounded in the need for placing and chaining traditional network functions such as firewalls and load balancers and turn to linear programming [11], [12] or genetic algorithms [13] to solve the problem. However, as we target a network with limited nodes and paths, we aim for a simpler solution in our work.

III. Network Model and Problem Statement

In this section we give a short description of the challenges we wish to solve and describe our basic model.

A. Target scenario and design requirements

We consider a mobile node (MN) that has multiple communication interfaces, such as the tourist bus in our introduction. Typically, each of these interfaces use different technologies and connect to separate networks maintained by different operators - here we call them "operator networks" - as depicted in Figure 1. Nevertheless, these networks are connected to a larger network (i.e., the Internet) where the desired services are available. The MN can, in other words use any of its interfaces to connect to a desired service, and should be able to choose the interface that provides the optimal combination of service parameters as defined in our introduction. As these parameters can be subject to change at any time, selecting the appropriate interface is a non-trivial problem. In addition, the chosen interface may also lose connection to its operator network at any given time. It is in the interest of both the MN and the service provider (at CN) that the MN utilizes its interfaces in the best possible way to ensure optimal service delivery. This could involve automatic service handover between the interfaces, or some sort of manual interaction performed by the user. A solution relying merely on IP routing would not suffice, even if interfaces are chosen based on QoS primitives, as the IP handover would cause service interruption. Our solution must therefore solve the problem at a higher level, involving the service provider, including some awareness of both the core and the operator networks.

As depicted in Figure 1 our mobile node has two interfaces and can connect to two different networks and reach the corresponding node through a core network. Both the core network and the two operator networks consist of a vast number of routers and possible paths. We assume that a virtual network is created on top of the physical network, for instance using network slicing techniques and that this overlay is controlled by a common SDN-NFV as in in Figure 2.

B. Virtual Network Function

Our network demonstrator is a simplified version of a VNF, and merely consists of flow rule manipulations for layer 3 packets. It serves the purpose for this study, and it is easy

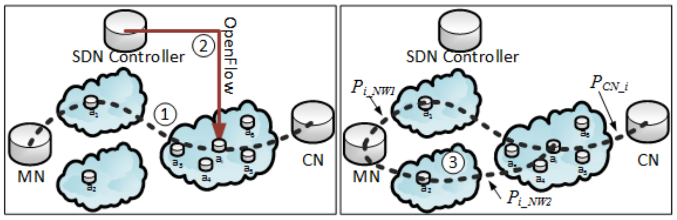


Figure 3. 1: A path is set up betweeen CN and MN. 2: SDN controllers sets up a duplication point. 3: A second path is set up from the duplication point to MN.

to envision enhancements such as application layer proxy mechanisms that could extend the functionality in the future. For that reason, we call this mechanism a network function despite its simplicity.

Our communication client on the MN communicates with the SDN controller via the service provider through a northbound REST-interface. The SDN controller then establishes a "duplication point" in one of the nodes and transmits flow-rules using OpenFlow [14] via the southbound interface, see Figure 3. Routes are established via the Intent-framework in ONOS SDN controller platform. Our architecture thus provides the MN with the capability to establish a streaming session on two interfaces simultaneously by communicating directly with the SDN-server which establishes a "duplication point". In a full setup, direct communication between MN and SDN controller may not be feasible as this feature may be limited by the SLAs between service providers and between service providers and end user. However, for the scope of this work, our solution solves the challenge in the following way. We allow direct communication between the MN and an intermediate node. This node represent the service provider in our case, which handles SLAs from the services and allows to send control messages to the SDN controller.

First, we define a virtual network function that bridges both operator networks and can direct traffic from the CN on both networks towards the MN. Second, we create an algorithm that can find the optimal position of this network function based on the requirements set by the user application on MN.

C. Network Function Placement

We are searching for the optimal placement for the network function that will either duplicate packets or increase capacity from the CN to the MN over the two networks that MN connects to. At first sight, the placement may intuitively be on the node that provides the least hops to each of the operator network nodes that connect to MN. However, each connection has a certain latency and a max capacity, other traffic in the network will influence the available paths, capacity or packet loss. Monetary cost (price) is also a factor that must be taken into consideration. The optimal placement of the network function depends on these metrics and on the requirements set either by the individual service MN is accessing or by the MN itself. These requirements also indicate the importance and/or constraints of the metrics when running a service. We assume that the SDN controller has updated information about the metrics on each link in the core network, and can obtain the same metrics from the SDN controllers in the two operator networks.

When considering the metrics described above, we can describe the network as a weighted graph $G = (V, E, l, pr, pl, c)$, where $V = \{v_1, v_2, \ldots\}$ correspond to the nodes MN, CN and all the nodes in the operator networks and core network, and E are edges that connect the nodes. The weight functions l, pr, pl and c describe the metrics on each of the edges, where l = latency (ms), pr = price (€), pl = packet loss (%) and c = capacity (Mbps).

When the network function is placed on a certain node v_i in the core network, the paths from CN to MN consist of the paths P_{CN_i} from CN to the node i, P_{i_NW1} from node v_i to MN via Network 1, and P_{i_NW2} from node v_i to MN via Network 2, see Figure 3. Although we use *two* networks in our description, this is not a limitation of the model. The number of interfaces of the MN gives the practical limitation of the number of operator networks involved.

Let P be a path in the graph G with a set of vertices and edges. The edges F in the path P are a subset of E ($F \subseteq E$). The cost of the path P with respect to each of the metrics described above can be described as follows:

Latency on the path P:

$$L(P) = \sum_{e_j \in F} l(e_j) \tag{1}$$

where $l(e_j)$ is the latency on the edge e_j

Price of the path P:

$$Pr(P) = \sum_{e_j \in F} pr(e_j) \tag{2}$$

where $pr(e_j)$ is the price of the edge e_j

Packet loss on the path P:

$$Pl(P) = 1 - \prod_{e_j \in F} (1 - pl(f_j)) \tag{3}$$

where $pl(e_j)$ is the packet loss on the edge e_j

Max throughput on the path P::

$$C(P) = \min_{e_j \in F} c(f_j) \tag{4}$$

where $c(e_j)$ is the capacity on the edge e_j

To calculate the path with the least weight we use Dijkstra's algorithm [15]. However, to use this algorithm, we must be able to *add* the weights. Hence, to calculate the packet loss in any path we find an alternative function for the packet loss where we use the sum of logarithms. As the packet loss on a link $0 \leq pl(e_j) \leq 1 \Rightarrow \log_b(1 - pl(e_j)) \leq 0$.

Packet loss on the path P:

$$Pl(P) = b^{-\sum_{e_j \in F} M(e_j)} \tag{5}$$

where $M(e_j) = (-1)\log_b(1 - pl(e_j))$ and $pl(e_j)$ is the packet loss on the edge e_j

The highest absolute values of $\log_b(1-pl(e_j))$ represent the highest packet loss, and as Dijkstra's algorithm is searching for a *minimum* path, we multiply each logarithm with -1 to get $M(e_j)$. Thus, we can both work with positive weights, and find the lowest values on the links with the least packet loss. We use $M(e_j)$ as our weight function for calculating packet loss in the following.

Algorithm 1: Finding optimal placement from weight function W using Dijkstra on positive weighted edges

```
Data: Weight function $W$ as an array,
List of nodes in core network $Core\_nodes$
Result: Node in Core network at which the Network
        Function should be placed
begin
    $N$ = Size of $Core\_nodes$;
    $W1$ = $W$ with all edges involving Network 2 set
     to -1;
    $W2$ = $W$ with all edges involving Network 1 set
     to -1;
    for $i$ = 0 to $N$ do
        Use Dijkstra's algorithm on edges with weight
         $>= 0$ to find the cost of the shortest path
         from $Core\_nodes[i]$ to MN using $W1$;
        Use Dijkstra's algorithm on edges with weight
         $>= 0$ to find the cost of the shortest path
         from $Core\_nodes[i]$ to MN using $W2$;
        Calculate mean cost for $Core\_nodes[i]$;
    Find node with lowest mean cost $n\_mean$;
    Return $n\_mean$;
```

The optimal placement of our Network function can be calculated with respect to either latency, price or packet loss using the weight function $l(e_j)$, $pr(e_j)$ or $M(e_j)$ as input to the pseudo code in Algorithm 1.

To calculate the path from the Network Function to the MN via each of the operator networks that MN connects to, we use Dijkstra's algorithm for shortest path twice; once with all edges involving Network 1 removed (W1 in Algorithm 1), and once with all edges to Network 2 removed (W1 in Algorithm 1). By using this method for each of the core nodes where our Network Function can be placed, we find a cost associated with the placement for each of the connecting networks.

The method in Algorithm 1 has its shortcomings as it only returns the best placement based on *one* parameter, i.e. W is either price - $pr(e_j)$, latency - $l(e_j)$ or packet loss - $M(e_j)$. It also does not take into consideration that there might be restrictions on the min/max values of certain edges in the path, such as minimum bandwidth. To compensate for this, we use Algorithm 2. In this algorithm we introduce the term *importance weight IW* - for each of the metrics (latency, price and packet loss) to identify which metrics are of importance to the type of service that uses the Network Function. Thus, we can either define the importance of each metric (IW) for each new stream or we can define *traffic classes* that have predefined weight profiles.

Algorithm 2: Calculating aggregated weight function W based on metric-specific weight functions

```
Data: Weight functions $c(e_j)$, $l(e_j)$, $pr(e_j)$ and
      $M(e_j)$ as arrays of size $N$,
   Importance weights $IW_l$, $IW_{pr}$, $IW_M$ as integers
   where $\sum_{i=l,pr,M} IW_i = 100$,
   Edge threshold_values $T_c$, $T_l$, $T_{pr}$, $T_M$ (if any)
   with indication on whether they are max or min
   thresholds
Result: Aggregated weight function $W$ to be used in
        Algorithm 1
begin
   // Check if threshold is violated and then
      find normalized value $f'$ from $f$:
   foreach $f(e_j) = (c(e_j), l(e_j), pr(e_j), M(e_j))$ do
      Find $f_{max}$;
      Find $f_{min}$, $f_{min} >= 0$;
      for $i$ = 0 to $N$ do
         if $f(e_i) < 0$ OR $f(e_i)$ exceeds given
          threshold then
            $f'(e_i) = -1$;
         else
            $f'(e_i) = \frac{f(e_i)-f_{min}}{f_{max}-f_{min}} \implies$
               $0 <= f'(e_i) <= 1$ ;
   // Calculate aggregated weight function:
   for $i$ = 0 to $N$ do
      if $l'(e_j) >= 0$ AND $pr'(e_j) >=$
       $0$ AND $M'(e_j) >= 0$ then
         $W(e_i) = l'(e_j) \times IW_l + pr'(e_j) \times IW_{pr} +$
          $M'(e_j) \times IW_M$ ;
      else
         $W(e_i) = -1$ ;
   Return $W$;
```

IV. Prototype implementation

The prototype of our architecture is developed based on the ONOS SDN controller [16]. ONOS is an open-source project supported by a large community of both network operators and vendors and is also considered stable for production networks.

Although our scheme is developed with ONOS in mind, we implemented the prototype as an application separate to ONOS. This was done for the following reasons: First, we wanted to ensure compatibility to a broader spectrum of SDN controllers. To ensure this, we wanted to avoid a tight integration to the Java based app framework in ONOS. Second, there is a tendency to move SDN northbound applications off-plattform to save controller resources and provide isolation. The gRPC technique can provide this via efficient language-agnostic communication. As we will explore this direction in the future, we developed a separate application in Python.

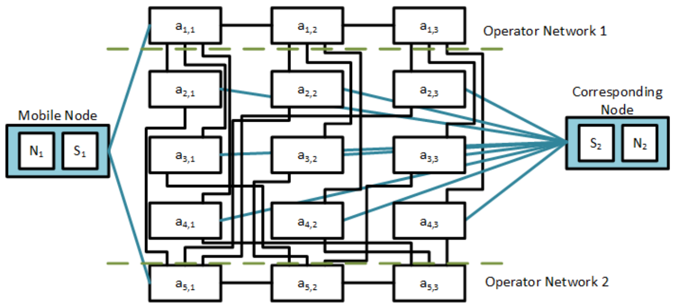


Figure 4. Connection details in test setup

Regardless of the choice of SDN controller, the placement algorithm must access the essential topology information repositories. In particular, the algorithm needs information about all links (i.e., the current topology), the remaining capacity on all links in the topology, and the latency over these links. The entire network topology can be retrieved from the link database in ONOS. This database is typically updated by the Link Layer Discovery Protocol (LLDP), which is exchanged frequently on all links. Link capacities can be retrieved from the port statistics, which are requested per switch using OpenFlow. We have not found a method to obtain link latencies from ONOS. However, this can possibly be measured by extending the LLDP protocol with piggybacked time stamps as proposed in [17]. Alternative methods are also proposed in [18], but further investigations in this direction is beyond the scope of this paper. In our prototype, link latencies are therefore entered manually into the algorithms.

Notice that in an SDN network, the MN can not communicate directly with the SDN controller, since the MN is only aware of the data-plane. Hence, we used a dual-interface network node that could communicate both with the control plane and the data plane. When the MN requests for example a "gold service" for its connection with CN, it sends a mini-SLA message to this intermediate node, which communicates with the SDN controller via its northbound REST interface. At the controller, the placement of the network function is decided and subsequently installed on the designated switch via OpenFlow messages.

The Python code implements algorithms 1 and 2.The networks we use in the evaluation are based on Open vSwitch switches within the Mininet emulator. We used ONOS 1.10, OpenFlow 1.3 and Open vSwitch 2.8.0 in the experiments.

V. Evaluation and analysis

To evaluate the efficiency of our system, we have constructed a simple network layout consisting of nine switches in a core network, and two operator networks consisting of three switches each, as depicted in (Figure 4). The core network in our simulations consists of a 3x3 matrix of nodes. Each node in Operator Network 1 connects to one *column* of nodes in the core network, while each node in Operator Network 2 connects to one *row* of nodes in the core network. The CN has a direct path to each node in the core network. Thus, each node in the core network have connection to one node in each of the two operator networks and to the CN. The MN connects to one node in each of the two operator networks.

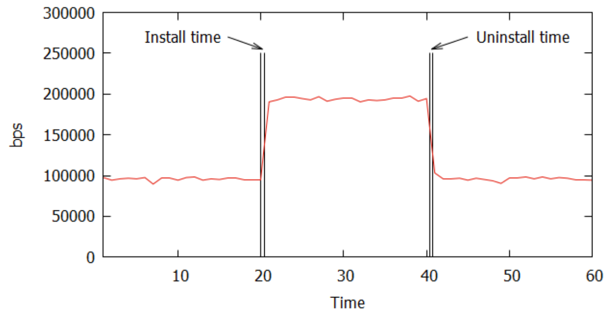


Figure 5. Traffic received at MN before, during, and after invoking the network function.

We chose this particular network layout to accommodate a great variety of metric combinations and possible network function positions in a rather compact layout manageable in Mininet emulations. The weight functions l, pr, pl and c in G were assigned a variety of values that allowed us to investigate various combination of metrics.

A. Network function proof of concept

To demonstrate the network function, a real-time traffic flow is sent between CN and MN via Operator Network 1. Some packet loss is observed at the MN, which degrades the QoE. Hence, after a while, at MN, a control message (mini-SLA) is sent to the SDN controller to initiate a "gold service", which installs the network function on one of the switches in the core network. In this experiment the network function is installed at $a_{1,1}$. Subsequently $a_{1,1}$ starts to duplicate packets going from MN to CN and utilises Operator Network 2 in addition to Operator Network 1. The throughput measured at the MN is shown in Figure 5. As illustrated in this figure, the transmission of the command to the SDN node did indeed result in a doubling of received traffic at the MN. After a while, the "gold service" is no longer necessary, and the MN requests standard service through the same interface.

After ten trials, the install time was measured to 47 ± 12 ms. The time is measured from a request message is initiated at the MN to the duplicated streams are received at MN. Uninstallation is merely a different mini-SLA and in practice the same process. Consequently, the time consumption for this process is identical. Notice that there is no latency on the individual links in this particular test. Admittedly, a more advanced network function would involve code execution, which can prolong the setup time substantially. However, it is worth noting that our simple function can be initiated without service interruption since the original data flow can continue simultaneous with the new flow via the duplication point.

B. Network function placement based on one metric

We now evaluate the network placement algorithm. We consider the one-metric case first, while the multi-metric case is discussed in the next section. For the evaluation, we use the network presented in Figure 4.

Table I
RELATIVE IMPORTANCE OF METRICS IN TWO TRAFFIC PROFILES

Profile A		Profile B	
Latency	15%	Latency	75%
Price	50%	Price	10%
Packet loss	35%	Packet loss	15%
Capacity (Threshold)	3	Capacity (Threshold)	2

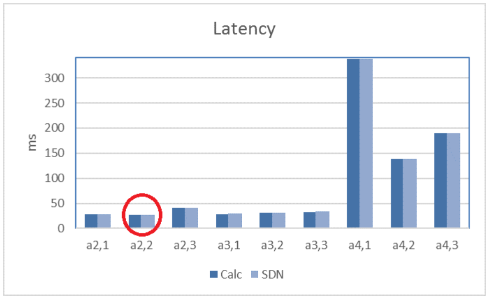
Corresponding weight functions have been used in both the emulated SDN network and as input to the Pyhton-implementation of Algorithm 1. Using Algorithm 1 we have calculated the mean costs of latency, capacity, packet loss and price respectively related to the paths from CN to MN via each of the nine core nodes. By running traffic from CN to MN in the emulated network (set up with the same characteristics as used in calculations), the measured latency, capacity and packet loss could be compared to the calculations. To generate the traffic, we used UDP with Iperf. Each of the nine nodes have been traversed twice; once with only Operator Network 1 connected, and once with only Operator Network 2 connected, as described in Algorithm 1, and the mean value was calculated. Let's first consider the results when the network function placement is optimised based on the latency metric. Figure 6a illustrates both the estimated and the measured *latency* between CN and MN for each of the possible placement alternatives. The optimal placement, identified by the algorithm, is shown in the figure outlined with a red circle. We observe that there is very good correlation between the estimated and the measured latencies illustrating the soundness of the placement algorithm.

In the same way, we can instruct the algorithm to optimize for *capacity* rather than latency. Figure 6b shows both the measured and the estimated capacities. The measured values are consistently lower than the estimated, since some of the paths have high packet loss, which reduces the measured capacity. The algorithm does not translate the accumulated packet loss along a path to reduced capacity. However, the result show that this does not necessarily impact the practical applicability of the placement algorithm. The result show that the algorithm can effectively pick the optimal placement (marked in the figure).

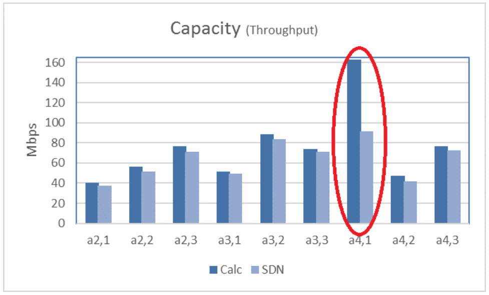
Following this further, we can investigate *packet loss* as the optimization metric in Figure 6c. In Figures 6d and 6e the results of considering the number of hops or price as metrics are presented. As illustrated by the graphs in the figures, the optimal placement of network function will vary depending on which metric is considered important.

C. Network function placement based on multiple metrics

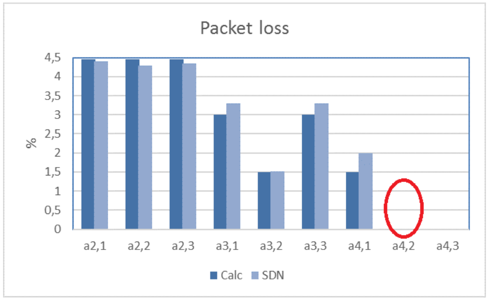
When optimizing with respect to a traffic class, the graphs presented in Figures 6a - 6e represent extremities, where only *one* metric is considered in each case. However, for practical purposes, more than one metric can be considered important, and optimization will be a trade-off between relevant metrics. Using the weight functions for each of the metrics price, latency, packet loss, as input to Algorithm 2 along with a traffic profile description, we can see that the optimal network



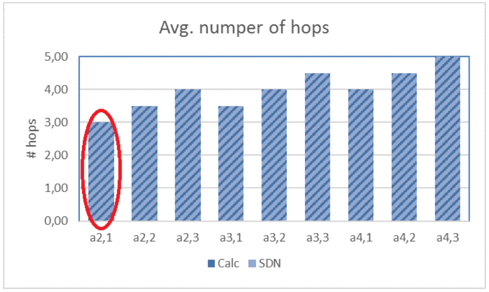
(a)



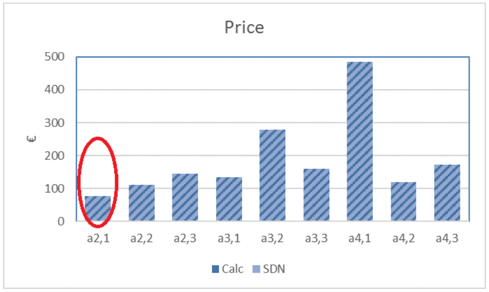
(b)



(c)



(d)



(e)

Figure 6. Optimal placement of Network Function with respect to various metrics.

Table II
OPTIMAL NETWORK FUNCTION PLACEMENT BASED ON COMBINED AND WEIGHTED METRICS

	Profile A	Profile B
Node	$a_{4,2}$	$a_{3,2}$

function placement varies again. For the experiment, we have created two different profiles. In these profiles, price, latency, and packet loss are each given an importance weight that add up to 100% (Table I). The capacity metric is used as a threshold value, indicating a minimum required available capacity on any link in a path. Links with capacity lower than the threshold is assigned a negative value and will not be considered by the algorithm. The resulting optimal network function placements are summarized in Table II.

VI. DISCUSSION AND CONCLUSION

In the solution we have proposed, we assume a simple setup where the MN can notify the SDN controller of its need for a new anchor point. In our proof-of-concept we have not performed in-depth studies of the setup time for the virtual network function, the signalling overhead or practical implementation of signalling between the MN, the CN (service provider) and the SDN controller. When SDN is used for data center networks, SDN control signalling would often be on dedicated links. In our case, where the user is mobile, this is generally not possible. Thus, the signalling must be in-band and must run on top of the data that it try to configure, and could be restricted by loss or congestion on the links. One solution to this is discussed in [19] and should be subject to further research. Ways to communicate Service level agreements (SLAs) between MN, CN and SDN controllers should also be studied in more detail as control signalling routes can add to the time of setup for a VNF in a new placement, and should be as short as possible. Also, further research is needed to study how the placement of a VNF which is optimal for a single MN (or a small group of MNs), influences overall network quality and traffic balance, and how these interests of the individual users and the network as a whole can be met.

In this paper we have presented a network function and a method for optimized placement of that virtual network function in an SDN that takes network state and service requirements into considerations. By focusing on one vital service parameter or by defining a specific service requirement profile, a virtual network function, such as an anchor point, can be placed according to the service requirements. We have shown that the chosen service parameter, or a combination of these in a service requirement profile, will alter the optimal placement of the virtual network function.

Such a profile can be a part of an initial SLA between a mobile user and a service provider or it could be part of a particular request that are transmitted on the fly when the user experiences service degradation. In this work we have considered the service parameters latency, price, packet loss, number of hops, and throughput. With a broader scope, such parameters could also include values that define the connections' security levels [20].

In this paper we propose a next step in using the opportunities laid out by software defined networks for bringing "connection everywhere, anytime, to the best service available" to life, with dynamic flow manipulation not just from a centralized view, but also for the individual user.

REFERENCES

[1] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[2] E. S. Boysen and T. Maseng, "Seamless handover in heterogeneous networks using SIP: A proactive handover scheme with the Handover Extension," *International Journal on Advances in Internet Technology*, vol. 2, no. 1, pp. 184–193, 2009.

[3] D. Bertsekas, "Dynamic behavior of shortest path routing algorithms for communication networks," *IEEE Transactions on Automatic Control*, vol. 27, no. 1, pp. 60–74, February 1982.

[4] C. W. Ahn and R. S. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 6, pp. 566–579, Dec 2002.

[5] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.

[6] R. Koodli, "Mobile IPv6 Fast Handovers," RFC 5568 (Proposed Standard), IETF, Jul. 2009.

[7] H. Soliman, C. Castelluccia, K. ElMalki, and L. Bellier, "Hierarchical Mobile IPv6 (HMIPv6) Mobility Management," RFC 5380 (Proposed Standard), IETF, Oct. 2008.

[8] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil, "Proxy Mobile IPv6," RFC 5213, IETF, Aug. 2008.

[9] S. Kuklinski, Y. Li, and K. T. Dinh, "Handover management in SDN-based mobile networks," in *2014 IEEE Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 194–200.

[10] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *2015 IEEE 4th CloudNet*, pp. 255–260.

[11] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 98–106.

[12] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization." in *CloudNet*, 2015.

[13] G. Xiong, Y.-x. Hu, L. Tian, J.-l. Lan, J.-f. Li, and Q. Zhou, "A virtual service placement approach based on improved quantum genetic algorithm," *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 7, pp. 661–671, Jul 2016.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM*, vol. 38, no. 2, pp. 69–74, 2008.

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

[16] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *HotSDN*. ACM, 2014, pp. 1–6.

[17] Y. Li, Z.-P. Cai, and H. Xu, "LLMP: Exploiting LLDP for latency measurement in software-defined data center networks," *Journal of Computer Science and Technology*, vol. 33, no. 2, pp. 277–285, Mar 2018.

[18] D. Sinha, K. Haribabu, and S. Balasubramaniam, "Real-time monitoring of network latency in software defined networks," in *Advanced Networks and Telecommuncations Systems (ANTS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1–3.

[19] O. I. Bentstuen and J. Flathagen, "On bootstrapping in-band control channels in software defined networks," *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, 2018.

[20] A. Fiaschetti, J. Noll, P. Azzoni, and R. Uribeetxeberria, *Measurable and Composable Security, Privacy, and Dependability for Cyberphysical Systems: The SHIELD Methodology*. CRC Press, 2017.