Modelling and Analysis of SDN and NFV Enabled Communications Networks

Joseph Billingsley, Wang Miao and Geyong Min
Department of Computer Science
University of Exeter, UK
Email: {jb931, wang.miao, g.min}@exeter.ac.uk

***Abstract*—TODO: What Why How Impact**

I. Introduction

The rest of this paper is organised as follows. Section II reviews the preliminaries that are useful for understanding the subsequent sections. In Section III we derive the analytical model. Section IV validates this model with extensive simulation experiments. Finally Section V concludes the paper, explores the implications of the models and examines future research directions.

II. Preliminaries

This section first introduces the concepts of Network Function Virtualisation and Software Defined Networking and presents the implications of these changes to the communication network architecture.

A. Network Function Virtualisation

In telecommunications networks, services are composed out of several network functions such as load balancers, firewalls and intrusion detection systems. Traditionally these network functions would be provided by specially engineered pieces of network hardware. In a Network Function Virtualisation (NFV) enabled network, these network functions are virtualised and can be run on industry standard servers reducing both capital and operating expenditure [].

Services can be defined using Directed Acyclic Graphs (DAG) which encapsulate dependencies between network functions such as in Fig. **??**. Some network functions can affect the data rate. Network functions such as firewalls or can reduce the rate by filtering out packets. Other network functions such as video decoders can increase the data rate by creating more packets for subsequent network functions to process. The task of constructing a 'concrete' service from an abstract DAG is called service composition and has received much interest in optimisation [**?**]. In Section III we describe how our model can be used to evaluate the performance of a given concrete service.

B. Software Defined Networking

Software Defined Networking (SDN) allow for dynamic routing of packets throughout the network [] and configuration of VNFs. A logically centralised SDN controller exists in the network which maintains a global view of the network. When a packet arrives at a SDN enabled device a flow lookup is performed to determine how the packet should be routed. If an appropriate flow does not exist in the router the device sends a request to the controller requesting an update to the flow table. Depending on the implementation, portions or the entirety of the flow table may be populated in advance.

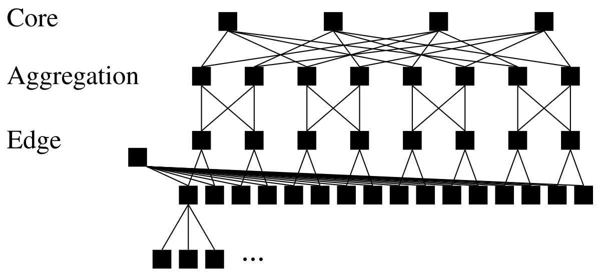

Fig. 1. The three layer topology modelled in this network

There may exist one or more SDN controllers. In a centralised SDN solution, a single control entity has a global view of the network. In contrast hierachical and distributed approaches use several control entities and either decompose or share the network information respectively.

C. The Network Architecture

In this work we extend the Fat Tree network [] frequently found in datacentres [] to be SDN and NFV capable. Fig. II-C illustrates this network. Generally the network is laid out as follows:

We define k as the number of ports for physical switches and k_{vm} the number of ports for the virtual switches. There are $(k/2)^2$ core switches. Each core switch connects to one switch in each of the k pods. There are k pods each containing two layers (aggregation and edge) of $k/2$ switches. Each switch in the edge layer is connected to each of the $k/2$ aggregation switches. Each of the remaining switches in the edge layer is connected to $k/2$ servers. Each server has a virtual switch with k_{vm} ports connecting to a VNF.

For each pod, each aggregation switch connects to the core switch with the same index as it and all subsequent switches $k/2$ steps away. As an example, if $k = 4$ the first aggregation switch in a pod connects to core switch 1 and 3, the second aggregation switch connects to 2 and 4 with this pattern repeating for each pod. This topology results in $n = k^3/4{*}k_{vm}$ VNFs.

For this research we constructed a network where each server is connected to a single centralised SDN controller. This topology is used in practice, most notably resembling VMwares NSX architecture []. Alternatively all elements in the network could be connected to the SDN [] or a distributed SDN. This exploration is left to future research.

III. Analytical Model

A. Assumptions

The analytical model is based on the following assumptions which are commonly accepted in the literature []. Assumption 3 is based on the high speed of data centre interconnection networks and the short distances between components.

1) At each VNF, messages are generated according to an independant Poisson process with a mean rate of λ messages a cycle. Furthermore, message destinations are uniformly distributed across the VNFs.
2) At each network component messages are serviced according to an independant Poisson process with a mean rate of μ messages a cycle.
3) The time taken for a message to travel between network components in negligible.
4) The SDN controller ensures messages take the shortest path between the two destinations and that messages are evenly distributed over the switches in the network.
5) Queues at each network component have infinite capacity.
6) Messages sent from a VNF to VNFs on other servers, may need to visit the SDN controller with probability p_{sdn_user}. The same does not apply to messages arriving at a server as the route to the destination will be known.

The mean message latency of the network can be calculated as the sum of the waiting time at each network component a message visits. As a result of the network architecture, messages will take shorter paths to visit VNFs under the same server, edge switch or pod. Consequently, the traffic arriving at servers at each layer in the network will also vary. The mean message latency for a single hop service chain can hence be calculated as:

$$\begin{aligned} Latency_{base} = &((w_{vm} + w_{sdn} + w_{srv}) \cdot p_{srv} \\ &+ (w_{vm} + w_{sdn} + 2w_{srv} + w_{edge}) \cdot p_{edge} \\ &+ (w_{vm} + w_{sdn} + 2w_{srv} + 2w_{edge} + w_{agg}) \cdot p_{agg} \\ &+ (w_{vm} + w_{sdn} + 2w_{srv} + 2w_{edge} \\ &\qquad + 2w_{agg} + 2w_{core}) \cdot p_{core}) \end{aligned} \tag{1}$$

Where w_{vm}, w_{sdn}, w_{srv},w_{edge},w_{agg}, w_{core} represent the average time spent in a VM, the SDN controller, a virtual, edge, aggregate and core switch respectively. Similarily p_{srv}, p_{edge}, p_{agg} and p_{core} represent the probability that the highest level a message which reach is a virtual, edge, aggregate or core switch respectively. The remainder of this section will deduce these values for arbitary settings of k and k_vm.

B. Probability of Highest Level

As messages will always take the shortest path, a message will visit only a virtual switch if the destination VM is in under the same virtual switch as the source. Given $n = k^3/4 \cdot k_{vm}$ total VMs:

$$p_{vm} = \frac{k_{vm} - 1}{n - 1} \tag{2}$$

Similarily the probability of a message visiting at highest an edge switch is the proportion of destinations that are under the edge switch, excluding those destinations that could be visited via a shorter route.

$$p_{edge} = \frac{(k/2) \cdot k_{vm} - k_{vm}}{n - 1} \tag{3}$$

This same techinque can be used to find the remaining probabilities:

$$p_{agg} = \frac{(k/2)^2 \cdot k_{vm} - (k/2) \cdot k_v m}{n - 1} \tag{4}$$

$$p_{core} = \frac{n - (k/2)^2 \cdot k_{vm}}{n - 1} \tag{5}$$

Finally we can calculate the probability p_{sdn} by excluding all messages that will not leave the server:

$$p_{sdn} = p_{sdn\_user} \cdot (1 - p_{vm}) \tag{6}$$

C. Calculation of Mean Waiting Time

To determine the mean waiting time at each network component, we model each component as a M/M/1 queue where the mean waiting time is calculated as []:

$$MM1(\mu, \lambda_{nc}) = \frac{1}{\mu - \lambda_{nc}} \tag{7}$$

Where nc is the network component under question. Despite messages being distributed evenly across switches in each layer by the SDN controller, the arrival rate, λ_{nc}, will be different for each layer as not all messages will visit all layers.

As destinations are evenly distributed over the VNFs, all VNFs will send an equal proportion of messages to all others:

$$\begin{aligned} \lambda_{vm} &= \frac{n-1}{n-1} \cdot \lambda \\ &= \lambda \end{aligned} \tag{8}$$

A portion of the messages being produced by every VNF will require the SDN controller to be consulted:

$$\lambda_{sdn} = n \cdot p_{sdn} \cdot \lambda \tag{9}$$

Servers can receive messages from three sources: generated from VNFs it is running, received from other VNFs and reflected messages that it had sent to the SDN. Following the same format:

$$\begin{aligned}\lambda_{srv} &= k_{vm} \cdot \lambda \\ &+ (n - k_{vm}) \cdot \frac{k_{vm}}{n-1} \cdot \lambda \\ &+ k_{vm} \cdot \lambda \cdot p_{sdn}\end{aligned} \tag{10}$$

Where line two can be understood as the number of VNFs that are not hosted by the server, sending an equal proportion of their messages to each of the VNFs hosted by the server.

The arrival rate for the edge switches can be deduced in the same way. Following the same order:

TODO: What about the impact of the messages being received to the upper level switches from the SDN?

$$\begin{aligned}\lambda_{edge} &= ((k/2) \cdot k_{vm}) \cdot (n - k_{vm}) \cdot \frac{1}{n-1} \cdot \lambda \\ &+ (n - ((k/2) \cdot k_{vm}) \cdot \frac{(k/2) \cdot k_{vm}}{n-1} \cdot \lambda\end{aligned} \tag{11}$$

The same principle can also be followed for the aggregate switches only now all traffic will be split between each aggregate switch in the pod. In the same order:

$$\begin{aligned}\lambda_{agg} &= (k/2)^2 \cdot k_{vm} \cdot \frac{n - (k_{vm} \cdot (k/2))}{n-1} \cdot \frac{\lambda}{k/2} \\ &+ n - ((k/2)^2 k_{vm}) \cdot \frac{(k/2)^2 \cdot k_{vm}}{n-1} \cdot \frac{\lambda}{k/2}\end{aligned} \tag{12}$$

Finally, as all VNFs are descendants of all core switches the arrival rate at each core can be calculated as the portion of messages arriving at a core switch, split evenly between each of the core switches.

$$\lambda_{core} = p_{core} \cdot n \cdot \frac{1}{(k/2)^2} \cdot \lambda \tag{13}$$

By substituting 8 to 13 for the arrival rates at each network component into 7 for the average waiting time we can calculate the latency incurred when visiting a network component, w_{nc}, for all network components except for the SDN controller.

When a message is sent to the controller it will incurr added latency both waiting at the controller and waiting at the server again when it returns. The expectation of the waiting time incurred by the SDN controller is:

$$w_{sdn} = (MM1(\mu, \lambda_{sdn}) + w_{srv}) * p_{sdn} \tag{14}$$

D. Mean Latency of Long Services and Many Services

As discussed in the preliminary section telecommunications services are typically composed from several network functions that pass messages from one to the other. As a result messages persist in the network for a longer period of time.

Consider a situation where each VNF send a message to an adjacent network function every cycle, under the same server or otherwise, so that all network functions will receive a message each cycle. Consider also that we have a service chain of three network function so that messages will be required to make two hops.

After the first cycle all messages will have sent and received one message. After the second cycle all messages will have sent two messages, forwarding the one received in the previous step and a new message from this cycle, and also received two messages, a message with no hops remaining and one with one hop remaining. At the third cycle one message will be destroyed having completed, leaving one message to be forwarded and one new message created for each VNF. The net result is that on average each VNF is producing 2 messages per cycle.

We can extend this intuition to arbitary length chains. Applying this back to the original analytical model we get:

$$\lambda = \lambda_{new} \cdot (len(chain_i) - 1) \tag{15}$$

where λ_{new} is the average number of new messages that are generated each cycle, len is the number of network functions that compose a given service chain and $chain_i$ is the service being modelled.

We can further extend this to several services, each of which may have different numbers of network functions. If a given message has probability $p(chain_i)$ of belonging to a particular service the average number of hops that a message persists for and hence the impact on the production rate can be calculated as:

$$\lambda = \lambda_{new} \sum_{i=1}^{ns} p(chain_i) \cdot (len(chain_i) - 1) \tag{16}$$

where ns is the number of different services and $\sum_{i=1}^{ns} p(chain_i) = 1$.

Finally we must also consider that messages that require several hops must also traverse the network several times. We can extend equation (1) by multiplying it by the average number of hops in the network considering each service:

$$Latency = Latency_{base} \cdot \sum_{i=1}^{ns} p(chain_i) \cdot (len(chain_i) - 1) \tag{17}$$

E. Mean Latency with Filtering VNFs

We make one final extension for the case where one or more network functions in a service may not forward all of the messages that they receive. As a result subsequent network functions in a service chain have lower production rates.

We can use the same conceptual model as before to solve this. Consider a situation where we have a service chain with 4 VNFs with 2 filtering VNFs that remove 50% of the traffic, as illustrated in **??**. After the first cycle all VNFs will have sent and received 1 message. After the second cycle all VNFs will have sent at least one message, and half of the VNFs will have forwarded another message bringing the average production rate to 1.5. After the first cycle all VNFs will have produced at least 1 message, another half will have forwarded the message received in the previous cycle and a quarter will have forwarded the message sent in the first cycle bringing the average production rate to 1.75.

We can extend this process to several services in the same manner as before by averaging the production rates of each service. The complete algorithm for calculating the production rate considering all aspects is as follows:

```
1: for $i = 1 : ns$ do
2:     $multiplier \leftarrow 1$
3:     for $j = 1 : len(chain_i)$ do
4:         $multiplier \leftarrow multiplier \cdot chain_i(j)$
           $\lambda_i \leftarrow \lambda_i + \lambda_{new} * multiplier$
5:     end for
6: end for
7: $\lambda \leftarrow \sum_{i=1}^{ns} \lambda_i \cdot p(chain_i)$
```

IV. Validation

V. Conclusions