QoS-Aware Virtualization-Enabled Routing in Software-Defined Networks

Alba Xifra Porxas[*], **Shih-Chun Lin**[†], and **Min Luo**[‡]
[*†]Broadband Wireless Networking Lab, School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332, USA
[‡]Shannon Lab, Huawei Technologies Co., Ltd., Santa Clara, USA
Email: [*]alba.xifra@ece.gatech.edu, [†]slin88@ece.gatech.edu, [‡]min.ch.luo@huawei.com

***Abstract*—Software-Defined Networking (SDN) has been recognized as the next-generation networking paradigm. It is a fast-evolving technology that decouples the network control plane from the data forwarding plane. A logically centralized controller is responsible for all the control decisions and communication among the forwarding elements. However, current traffic engineering techniques and state-of-the-art routing algorithms do not effectively use the merits of SDNs, such as global centralized visibility, control and data plane decoupling, network management simplification and portability. In this paper, a multi-tenancy management framework is proposed to fulfill the quality-of-services (QoSs) requirements through tenant isolation, prioritization and flow allocation. First, a network virtualization algorithm is provided to isolate and prioritize tenants from different clients. Second, a novel routing scheme, called QoS-aware Virtualization-enabled Routing (QVR), is presented. It combines the proposed virtualization technique and a QoS-aware framework to enable flow allocation with respect to different tenant applications. Simulation results confirm that the proposed QVR algorithm surpasses the conventional algorithms with less traffic congestion and packet delay. This facilitates reliable and efficient data transportation in generalized SDNs. Therefore, it yields to service performance improvement for numerous applications and enhancement of client isolation.**

I. Introduction

During the past decade, due to the increment and complexity of client demands and requirements, networks required the enhancement of routing strategies to be taken into consideration. Nowadays, network demands from clients are still increasing and becoming more challenging. Network operators in commercial clouds and data centers have been trying to improve network performance while fulfilling application requirements. However, this objective keeps increasing in difficulty. Software-Defined Network (SDN) decouples control and data planes, simplifying network management and control by providing global visibility and direct control of the underlying forwarding elements through a central controller. This new network technology permits to separate routing from forwarding elements, so that routing decisions can be taken in the centralized controller. To be more specific, instead of letting forwarding elements manage the traffic, software from outside the devices directs networking traffic. Open-Flow forwarding elements [1] just look-up routes in a table and forward packets without being concerned about path selection, thus, contributing to faster functionality. This centralized architecture provides a global real-time network state view, which allows the implementation of improved routing techniques since the controller is aware of the overall network status.

The routing framework needs to be reevaluated from a fully distributed path-selection computation towards a central administration of the path calculation in the controller. The improvement of network performance, e.g. delay, capacity and reliability, is still unexplored in SDN routing schemes. Networks and their capabilities need to be designed to serve adequately the requests demanded. These service demands can be associated with specific quantitative characteristics of flows, e.g. delay bounds, throughput restrictions, or priorities between different classes or entities. Taking into consideration these requirements presents new routing challenges in the SDN scenario. Cloud computing and data centers widely use a multi-tenant and resource sharing architecture. This architecture is used due to its infrastructure and maintenance cost-effectiveness, simplification, lower system requirements and flexibility. In single-tenant architectures, clients have their own dedicated resources. For nearly all clients 80% of these resources will be idle most of the time, based on the Pareto principle. In multi-tenancy applications, resources are shared between clients, which means usage of fewer resources and capability to increase the utilization of resources. However, multi-tenancy may generate conflicts when resources become overloaded. Since applications and resources are shared, tenant workloads may interfere with each other, as high demand tenants can monopolize the shared resources and performance goals, as stated in [2], [3]. SDN global view of the network allows to monitor tenant resource consumption, hence detecting aggressive tenants occupying all resources. Nevertheless, it is still required to isolate tenants from each other to allow customization performance and enforce security.

In addition, best-effort and real-time traffic need to be supported at the same time. Networks need to cope with multimedia traffic, like video and voice, which has different characteristics than data applications. Real-time traffic does not have the elasticity to adapt the packet transmission rate depending on the network congestion as elastic traffic, which tolerates delay lithely. The presence of different traffic concurrently in the network brings the following situation: real-time traffic will eventually interfere with non-real-time traffic, due to its quality of service demands. In congestion scenarios, elastic traffic could achieve starvation situations, as real-time flows would not hold back and share resources fairly [6], [7]. Hence, as considered in [8], [9], routing design needs to fulfill service demands regarding response delivery and QoS provision. Flow allocation applying traffic engineering techniques in SDN is discussed in [4], [5]. However, multi-tenancy architecture and QoS requirements guarantees are not considered.

978-1-4673-6432-4/15/$31.00 ©2015 IEEE

Leveraging SDN new architecture, it is possible to develop a routing framework for multi-tenancy environments with QoS-aware flow allocation and prioritization. Therefore, service quality and client demands will be enhanced. This is of remarkable significance in cloud computing and in enterprise data centers [10]. Demand flow allocation in this environment should involve resource isolation of multiple tenants, established client priorities and guarantee client QoS requirements. For each new flow entering in the network, the most appropriate path needs to be found according to these requirements. As a first step, a network virtualization is proposed to slice the physical layer into separate subnetworks topologies. As it is known, graph partitioning is a NP-complete problem [11], hence our problem will be at least NP-hard as a result of its analogy. Therefore, an algorithm is proposed to simplify the exploration of the network slicing. It will procure isolation between tenants and introduce priorities in the model. In conjunction, a QoS-aware Virtualization-enabled Routing (QVR) algorithm is developed to conjointly isolate clients in a multi-tenancy architecture and prioritize them, as well as guaranteeing QoS-requirements whilst computing the flow allocation. This represents an advancement for network operators to manage easily the network, providing a portable and customized solution, enhancing service performance and improving client satisfaction.

The rest of this paper is organized as follows. Section II provides the system model. Section III introduces the network virtualization algorithm to accomplish multi-tenant priorities and isolation. Section IV presents the optimization problem formulation and our QVR solution. Section V provides the performance evaluation of our routing design. Section VI discusses conclusion and future work.

II. System Description

The objective of our work is to develop a routing algorithm that maximizes the isolation between tenants, in such a way to conjointly fulfill the QoS requirements. It consists two stages. The first stage consists of a network virtualization, dividing the network in layers in order to separate flows from different tenants. In the second stage a routing algorithm is developed to control the allocation of new flows arriving in the network according to client priorities and requisites. The first stage will be done offline. It will not change as new flows enter in the network, i.e, layers computed will be static. Moreover, the second stage is done online, adjusting paths routes of new flows arriving in the SDN controller. If online stage is fail to provide the required performance, the Management Tool (MT) will feedback to offline stage to enable a better network slicing. The system architecture is shown in Fig. 1. MT running above the SDN controller is responsible for virtualizing the network into layers and storing the physical network's subnets. In SDN, when a new flow arrives into the network, it is forwarded to the controller which requests the MT to route this new flow. MT determines the most suitable path for this new flow and decide the best resource allocation according to the current status of the network. Subsequently, the SDN controller translates the corresponding routes into understandable routing entries for the involved forwarding elements [1].

First, we introduce some notations and definitions for our system model. We define the physical topology of the network

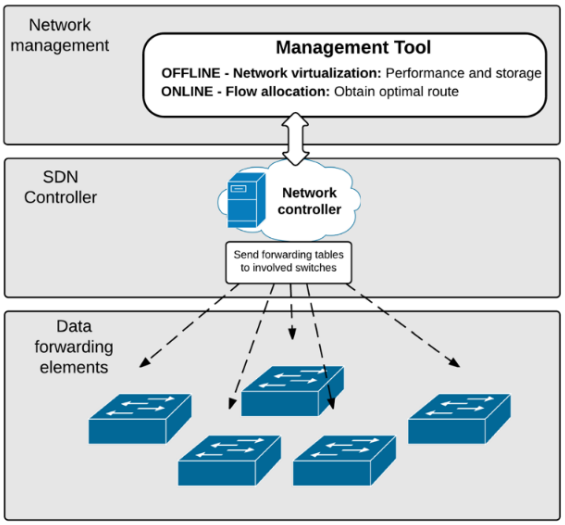


Fig. 1. System architecture

as an undirected graph $G = (V, E)$, where V refers to the set of vertices of the network and E to the set of edges connecting them. An edge from node u to node v is defined as $u \rightarrow v$. An undirected graph G is a graph in which edges have no orientation. Thus, the edge $u \rightarrow v$ is the same as the edge $v \rightarrow u$. A directed acyclic graph G is defined as a graph formed by a group of vertices and directed edges such that, it is not possible to start at some node u and follow a sequence of edges that loops back to u. A graph $S = (V', E')$ is a subgraph of G, $S \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. A complete graph G is defined as an undirected graph in which each pair of vertices has a unique edge connecting them. A connected graph G is a non-empty graph in which any two of its vertices are connected by at least one path in G. The complement of a graph G is the graph $\bar{G} = (V, \bar{E})$, which has the same vertex set, but whose edge set consists of the edges not present in G. A minimum spanning tree of a connected undirected graph G is a tree that includes every vertex using the minimal set of edges according to its weights.

III. Network Virtualization

A SDN is modeled as a graph $G = (V, E)$. Nodes forward traffic from hosts and applications, whereas edges are the links that allow communication between vertices possible. When a flow needs to be sent from any source to any destination, it is very unlikely that there is a direct physical link between them. Thus, we assume there will be more than one viable route between source-destination pair. Our approach is to virtualize the physical network in distinct separate subnets in order to isolate multiple tenants in the current physical network. To achieve tenant isolation, a network virtualization algorithm is examined in the following.

Assume there are N tenants within a network graph G. Several subgraphs $G_1, ..., G_N$ are define for each corresponding tenant. These subnets need to fulfill the following requisites: all vertices in G need to be included in subgraphs G_n, where $n = \{1, ..., N\}$, because traffic can be generated from and destined to all vertices in the network, thus $G_n = (V_n, E_n)$, where $V_n \subseteq V$ and $E_n \subseteq E$. Note that V_n can be

a virtual node that aggregates multiple physical nodes. Hence, subgraphs G_n need to be connected graphs because any node must be able to reach any other node in the network. Ideally subgraphs G_i and G_j, where $i \neq j$, should be edge-disjoint, i.e., they should not have any edge in common: $E_i \cap E_j = 0$. One subgraph should be the complement graph of the other N-1 subgraphs: $G_n = \bar{C}$, where $C = \cup_{k=1,k\neq n}^{N} G_k$. While it might happen that *shared edges* appear among tenants for dependent resource utilization, in Section IV, a flow allocation algorithm is proposed to ensure each tenant's requirements are fulfilled. Furthermore, as stated previously, the centralized controller of SDN allows tenant-aware traffic monitoring, so that the capacities and shared links between tenants can be adaptively rearranged. Our second goal is to provide different priorities among tenants. We assume Tenant n demands are less imperative as n grows, thus Tenant 1 demands the highest priority and Tenant N the lowest. Next we define $w_{1,...,N}^{max}$ as the maximum path weight a tenant requests, i.e., either path delay or packet losses, which are used to examine whether the subgraph fulfills the tenant's requirements or not. Towards this, the overall framework can be formulated as:

$$\min_{\{G_n\}} E_i \cap E_j$$

$$\min_{\{G_n\}} \sum_{1 \leq n \leq N} \mathbb{I}_{\{W(G_n) > w^max_n\}}$$

where $\mathbb{I}_{\{\cdot\}}$ is the indicator function and $W(\cdot)$ is the function to characterize the weight provided by the subnet.

While the proposed framework of network virtualization is a multi-objective optimization problem, the computation complexity is considerable, and normally it does not have a single optimal solution. Instead, we propose a fast algorithm that aims to attain $E_i \cap E_j \to 0$, i.e., minimal number of edges shared between tenants. This will provide the isolation needed in the multi-tenancy architecture, thus minimizing interference among tenants. Fist of all, a minimum spanning tree is searched within the graph G using Kruskal's algorithm [13]. The edges are added to the highest priority subgraph G_1. Next, it is verified that all source-destination pairs have at least one path with path weight less than w_1^{max}. In particular, the path weight from source u to destination v is determined through a shortest path algorithm, e.g. Dijkstra's algorithm [14], as follows. Pairs (u, v) that do not fulfill the w_1^{max} constraint are first found. For each discovered pair (u, v), Dijkstra's algorithm is run over G_1, and new edges are added to G_1 to guarantee the corresponding route has a weight less than w_1^{max}. The iteration continues until w_1^{max} is fulfilled in all source-destination pairs inside subgraph G_1. Furthermore, the algorithm works on finding the edges belonging to the lowest priority subgraph G_N. In particular, Kruskal's algorithm is applied over the graph $-G$ for a maximum spanning tree. The edges are added to the lowest priority subgraph G_N. Then, the paths between source-destination pairs are found and compared with w_N^{max} to add more edges if it applies. Finally, subgraphs G_t where $t = 2, ..., N-1$ will be constructed, running Kruskal's algorithm over the graph $G - G_N - \sum_{k=1}^{t-1} G_k$. Then, there is a searching for isolated nodes, and the edges from these isolated nodes to subgraph G_t are further added into G_t. If more than one possible edge for an isolated node is found, the algorithm will choose the edge with less weight. Note that there might be some edges that are already included in the previous subgraphs, and thus will become *shared edges* among subnets. Last, all source-destination pairs (u, v) need to be verified that each pair has at least one path, whose weight is less than w_t^{max}. A pseudo-code for the proposed fast algorithm of network virtualization is given as follows:

Network virtualization algorithm

Input: Network topology (G), Edge weights $w_{1,...,N}^{max}$
Output: Network subnets $G_1, ..., G_N$

Subnetwork Tenant 1
- Run Kruskal's alg. over graph G % To obtain minimum spanning tree of graph G;
- Add the found edges to G_1;
- Find source-destination pairs (u, v) with path weight larger than w_1^{max};

for each discovered pair (u, v)
  while path weight between $(u, v) > w_1^{max}$
    Run Dijkstra alg. over G_1 for pair (u, v);
    Add new found edges to G_1;
  end
end

Subnetwork Tenant N
- Run Kruskal's alg. over graph $-G$ % To obtain maximum spanning tree of graph G;
- Add the found edges to G_N;
- Find source-destination pairs (u, v) with path weight larger than w_N^{max};

for each discovered pair (u, v)
  while path weight between $(u, v) > w_N^{max}$
    Run Dijkstra alg. over G_N for pair (u, v);
    Add new found edges to G_N;
  end
end

Subnetworks Tenants 2,...,N-1
for each tenant $t \in \{2, ..., N-1\}$
  Run Kruskal's alg. over the graph: $G_t = G - G_N - \sum_{k=1}^{t-1} G_k$
  Find the isolated nodes;
  Add the edges from the found nodes to G_t % To transform G_t into a connected graph;
  Find source-destination pairs (u, v) with path weight larger than w_N^{max};
  for each discovered pair (u, v)
    while path weight between $(u, v) > w_t^{max}$
      Run Dijkstra alg. over G_t for pair (u, v);
      Add edges to G_t;
    end
  end
end

IV. QoS-Aware Virtualization-Enabled Routing (QVR) Algorithm Design

QoS-aware Virtualization-enabled Routing (QVR) algorithm is proposed that combines the network virtualization algorithm in Section III with the proposed QoS-aware flow allocation in this section. Our final objective is to route flows through paths that fulfill the corresponding QoS requirements. The details of the QVR algorithm are explained as follows.

Let $k = \{s, d\}$ determine the $\{source, destination\}$ pair and $c_{i,j}$ represent the maximum capacity of link (i, j). Thus, $c^n_{i,j}$ determines the bandwidth allocation in link (i, j) for Tenant n. We use a^n_k to represent flow arrival from Tenant n for source-destination pair k, and $Y^n_{i,j\to d}$ to represent flows routed on link (i, j) for destination d from Tenant n. Let A^n_k represent the current amount of injected traffic in the network. Let $\hat{\mathrm{A}}^n_k$ represent possible future injected traffic arrivals for $\{source, destination\}$ pair k, predicted using traffic matrix estimators as in [15]. Hence, we considers new flows between all $\{s, d\}$ pairs, and $\hat{\mathrm{Y}}_{ij\to d}$ represents possible future traffic routed on link (i, j) for destination d. Let λ_k represent the scale indicator of the amount of future injected traffic the network can handle and deliver appropriately. If the optimal value λ_k is less than 1, it indicates that future flows won't fit in the network unless some current flows end. We want to maximize the future resource availability, i.e., minimize the current maximum link utilization, which is a usually used performance metric [5], [16], [17]. Therefore, our objective is to maximize the minimum residue bandwidth among links as shown in (1). Assume the QoS requirements for flow a^n_k are given as: the maximum acceptable end-to-end delay α^n and packet loss β^n, the required throughput γ^n, and the maximum allowable jitter ψ^n. Let $p^n(v)$ represent the packet loss probability in node v, $L\{a^n_k\}$ represents the packet length of traffic a^n_k, $T_{EE}\{a^n_k\}$ indicate end-to-end delay from s to d for a^n_k, and $T_w(v)$ denote the queuing time in node v. Given the variables are $Y^n_{i,j\to d}$ to determine the best flow route, the flow allocation problem of QVR algorithm is finally formulated as:

$$\max_{\{Y^n_{ij\to d}\}} \min_k \lambda_k \tag{1}$$

subject to

$$Y^n_{ij\to d} \geq 0 \tag{2}$$

$$\sum_{n\in\{1,\ldots,N\}} \sum_{d\in N} Y^n_{ij\to d} \leq c_{ij} \tag{3}$$

$$\sum_{p:(p,i)\in G^n} Y^n_{pi\to d} - \sum_{p:(i,p)\in G^n} Y^n_{ip\to d} \geq \begin{cases} \sum\limits_{s\in N, s\neq i} A^n_{si} & \text{if } i = d \\ A^n_{id} & \text{otherwise} \end{cases} \tag{4}$$

$$\sum_{p:(p,i)\in G^n} (Y^n_{pi\to d} + \hat{\mathrm{Y}}^n_{pi\to d}) - \sum_{p:(i,p)\in G^n} (Y^n_{ip\to d} + \hat{\mathrm{Y}}^n_{ip\to d}) \geq \begin{cases} \sum\limits_{s\in N, s\neq i} (A^n_{si} + \lambda^n_{si}\hat{\mathrm{A}}^n_{si}) & \text{if } i = d \\ -A^n_{id} - \lambda^n_{id}\hat{\mathrm{A}}^n_{id} & \text{otherwise} \end{cases} \tag{5}$$

$$T_{EE}\{a^n_k\} = \sum_{(i,j)\in Path} \frac{L\{a^{n,QoS}_k\}}{Y^n_{ij}} + \sum_{n\in Path} T^n_w(n) < \alpha^n \tag{6}$$

$$\prod_{n\in Path} p^n(n) < \beta^n \tag{7}$$

$$Y^n_{ij\to d} \geq \gamma^n \quad \forall(\text{i,j}) \in \text{Path} \tag{8}$$

$$var(T_{EE}\{k\}|_{Path}) < \psi^n \tag{9}$$

The set of inequalities in (2) ensures that flows on any path are always positive. The set of inequalities in (3) ensures that resources allocated for different tenants do not exceed link capacity, either the link is shared or not. The set of inequalities in (4) ensures that the current traffic arrivals flow conservation, i.e., ensures that the matrix traffic is routed through the network, and in (5) residue bandwidth is ensured for possible future injected traffic flow. Last constraints (6), (7), (8) and (9) guarantee the QoS requirements for each flow from the corresponding tenant. Not each one of these constraints will be applied to allocate all new arrivals. Sensitive QoS constraints will be recognized depending on traffic characteristics and flow patterns, and applied accordingly [21]. For example, in interactive applications the overall one-way delay needs to be short in order to give the user an impression of real time responses. Thus, delay and jitter constraints will be more stringent than the loss constraint. Further, for web browsing or email the jitter constraint it is not applicable because it has little impact on its performance, whereas throughput and loss constraints are of considerable significance. Hence, α^n, β^n, γ^n and ψ^n indicate the actual value required for each parameter, which will be different depending on each application. A pseudo-code for the proposed QVR algorithm is as follows:

QVR Algorithm

OFFLINE - Network Virtualization
Input: Network topology (G), Edge weights, $w^{max}_{1,\ldots,N}$
Output: Network subnets $G_1, ..., G_N$

ONLINE - Flow Allocation
Input: Network subnets $G_1, ..., G_N$, Current flow allocation matrix Y, Link aapacities C, Traffic arrivals matrix A, Requirements new flow a^n_k
Output: λ, Updated flow allocation matrix Y

Paths = find feasible paths from s to d in G_n;
for each path ∈ Paths
  if ∃ path ∈ Paths s.t. $T_{EE} > \alpha^n$
    Do not consider this path
  end
  if ∃ path ∈ Paths s.t. $PL > \beta^n$
    Do not consider this path
  end
  if ∃ link (i, j) ∈ Paths s.t. $c_{i,j}$ available $> \gamma^n$
    Do not consider this path
  end
  if ∃ path ∈ Paths s.t. $Jitter > \psi^n$
    Do not consider this path
  end
end
for each path ∈ Paths remaining
  Calculate possible future injected traffic
end
Determine mininum λ_k
Choose the path maximizing λ_k
Update flow allocation matrix Y

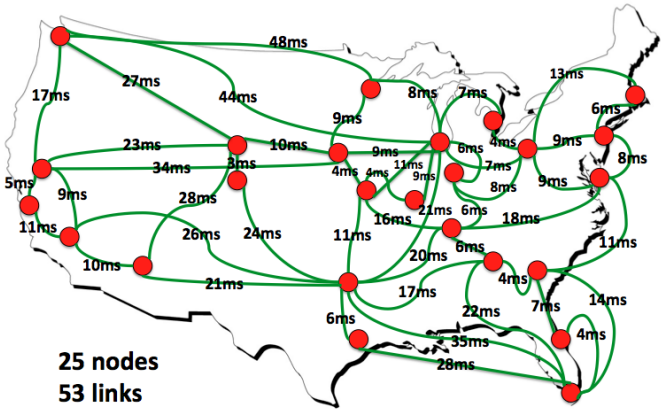
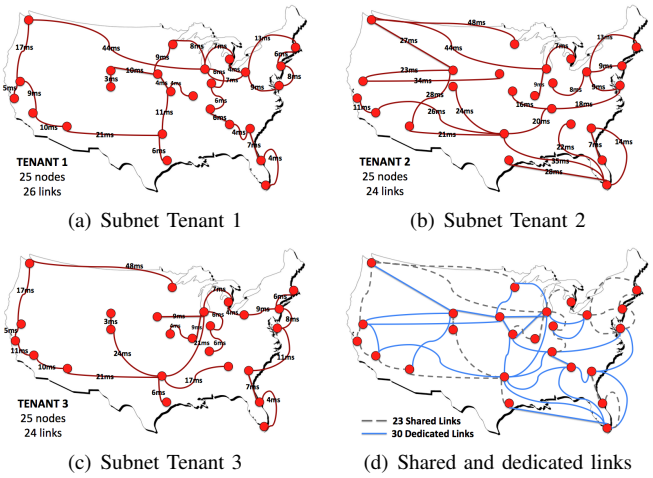


Fig. 2. Network topology



(a) Subnet Tenant 1 (b) Subnet Tenant 2

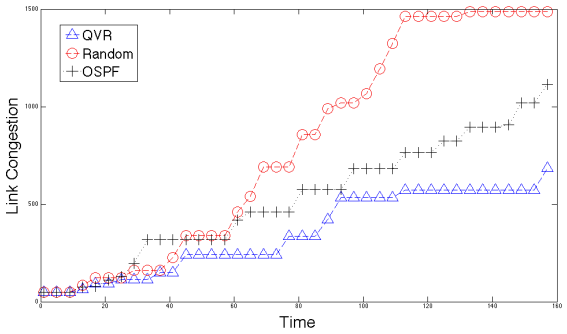
(c) Subnet Tenant 3 (d) Shared and dedicated links

Fig. 3. Tenant's subnets and shared links

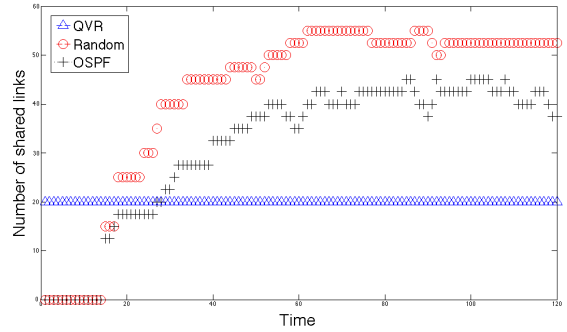
V. Performance Evaluation

In order to evaluate the proposed QVR algorithm, a network [19] with 25 nodes and 53 links is considered as shown in Fig. 2. It is assumed there are three tenants that operate in this network. In particular, Tenant 1 generates traffic from real-time applications, which can be modeled as Pareto distribution due to its burstiness. Furthermore, Tenant 2 and 3 generate traffic from non-real-time applications, which can be modeled with Poisson arrivals and exponential service times. The maximum path weight allowed for each tenant will be 100 ms, 400 ms and 250 ms respectively, with respect to the QoS requirements of its applications [21].

Fig. 3 shows the tenant isolation by QVR algorithm. In Fig. 3a, 3b and 3c, the subnets for Tenant 1, 2 and 3 are shown, respectively. Moreover, the superposition of these tenants is provided in Fig. 3d, where the shared links are highlighted. Thus, after running the virtualization algorithm we have the physical network divided in three subgraphs. These layers have been built to have the minimum superposition, nevertheless, there will be some links where the traffic from different tenants will combine together. Based on isolated network subgraphs for different tenants, we compare the QVR algorithm with OSPF [18] and random flow allocation routing algorithms.



(a) Evolution of the minimum link congestion

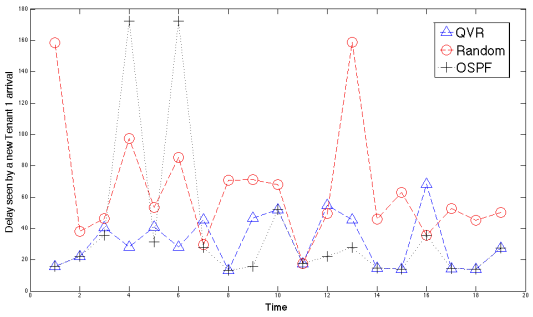


(b) Evolution of number of shared links

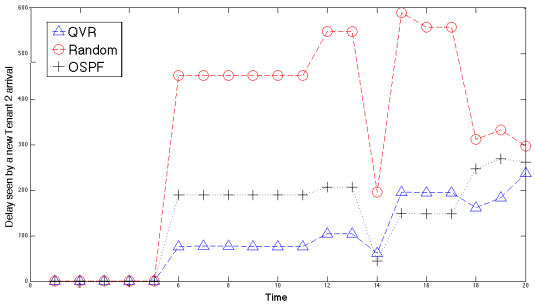
Fig. 4. Minimum link congestion and number of shared links

We ran a first experiment to measure the evolution of link congestion in all three scenarios: OSPF, randomly flow allocation and finally QVR. Fig. 4a shows the performance results. It can be noted that QVR does significantly better than a randomly flow allocation and OSPF. QVR has much less congested links than the other two algorithms, because QVR chooses routes to maximize the minimum residue bandwidth for future accepted traffic flows. Furthermore, Fig. 4b shows the number of shared links between tenants for different routing algorithms. Whereas under OSPF routing and random flow allocation, the number of links shared among tenants reaches the maximum in an early stage, QVR maintains the number of shared links as a constant, thus achieving better isolation. It is shown that QVR accomplishes tenant isolation while also improving the link congestion of the network, hence, being able to accept more future flows from tenants.

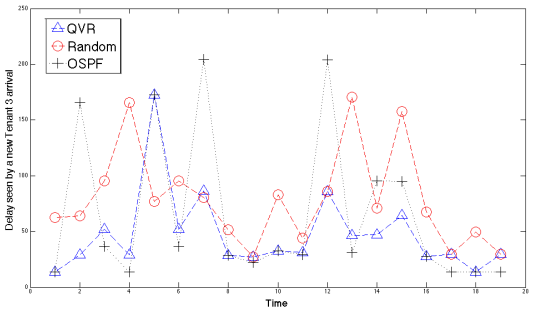
Furthermore, the delay perceived by tenant arrivals when they enter the network is examined and the results are shown in Fig. 5. Tenant 1 considers real-time applications that provide much less delay as compared to Tenant 2 and Tenant 3. Tenant 1 brings more bursty traffic than Tenant 2 and 3, that deal with non-real-time traffic. Tenants maximum delays are 100 ms, 400 ms and 250 ms respectively. It is shown in Fig. 5 that random flow allocation and OSPF do not fulfill this requirement for Tenant 1, and are closer to the limit than QVR for Tenant 2 and 3. Hence, the end-to-end delay from QVR is much less than the one from OSPF or random allocation. Thus, QVR meets the need of a good traffic engineering tool, facilitating reliable and efficient transmissions in SDNs.



(a) Tenant 1 arrivals



(b) Tenant 2 arrivals



(c) Tenant 3 arrivals

Fig. 5. Delay perceived for tenant arrivals

VI. Conclusion

In this paper we have shown that applying traffic engineering in SDN can provide solutions to enhance network management and service quality. This is of notable significance in cloud computing and enterprise data center environments. We have proposed QVR as a routing framework for multi-tenancy management to fulfill QoS requirements through tenant resource isolation, prioritization and flow allocation. It enables efficient management and network control to network operators, providing a portable, simplified and customizable solution. It has been shown the enhancement of the network performance accomplished in a centralized SDN scenario regarding traffic congestion and packet delay. But with the centralized scenario scalability issues may arise. In contrast, a distributed scenario is more scalable, but there may be state inconsistency and increase of shared information. In general, a centralized approach is better for data centers or home networks, whereas a distributed approach is better for large scale networks, e.g. cloud environments. Future work will be focused on a distributed controller scenario, as well as improved techniques in flow control and admission management.

References

[1] Openflow switch specification v1.0-v1.4 <https://www.opennetworking.org/sdn-resources/onf-specifications>.

[2] D. Shue, M. J. Freedman, and A. Shaikh, "Performance Isolation and Fairness for Multi-Tenant Cloud Storage", *10th USENIX Symposium on Operating Systems Design and Implementation*, 2012.

[3] W.T. Tsai, Q. Shao, and J. Elston, "Prioritizing Service Requests on Cloud with Multi-tenancy", *IEEE International Conference on E-Business Engineering*, 2010.

[4] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks", *Computer Networks 71, pp. 1-30*, June 2014.

[5] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic Engineering in Software Defined Networks", *IEEE INFOCOM'13*, April 2013.

[6] S. Shenker, "Fundamental Design Issues for the Future Internet," *IEEE Journal*, Sep. 1995.

[7] S. Shenker, D. Clarck, and L. Zhang, "A scheduling service model and a scheduling architecture for an integrated services packet network", 1993.

[8] S. Chen and K. Nahrstedt, "An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions" *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*, Nov./Dec. 1998.

[9] Z. Wang and J. Crowcroft, "Quality-of-Service Routing for Supporting Multimedia Applications", *IEEE Journal on Selected Areas in Communication*, Vol. 14, No. 7, Sep 1996.

[10] S. Jain, et at., "B4: experience with a globally-deployed software defined wan", *Proceedings of the ACM SIGCOMM Conference, SIGCOMM'13*, pp. 3-14, Aug. 2013.

[11] M. Garey and D. Johnson, *Computers and Intractability - A Guide to the Theory of NP-completeness*, Freeman, 1979.

[12] R. Sedgewick and K. D. Wayne, *Algorithms* (4th ed.), Addison-Wesley Professional, ISBN 9780321573513.

[13] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem", *Proceedings of the American Mathematical Society*, 1956.

[14] E. W. Dijkstra, "A note on two problems in connection with graphs", *Numerische Mathematik*, pp. 269-271, 1959.

[15] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: traffic matrix estimator for openflow networks", *Proceedings of the 11th International Conference on Passive and Active Measurement, PAM'10*, pp. 201-210, Apr. 2010.

[16] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world", *Selected Areas in Communications, IEEE Journal on*, vol. 20, pp. 756-767, May 2002.

[17] C. Jian and L. Chin-Tau, "Optimal link weights for IP-Based networks supporting Hose-Model VPNs", *IEEE/ACM Transactions on Networking*, vol. 17, pp. 778-788, June 2009.

[18] J. Moy, "Ospf version 2", *IETF RFC 2328*, 1998.

[19] https://www.sprint.net/performance/

[20] R. Raghavendra and E. M. Belding, "Characterizing High-bandwidth Real-time Video Traffic in Residential Broadband Networks", *Proc. WiOpt*, pp. 597-602, 2010.

[21] Y. Chen, T. Farley, and N. Ye, "QoS Requirements of Network Applications on the Internet", *IOS Press*, 2004.