An Ensembled Scheme for QoS-aware Traffic Flow Management in Software Defined Networks

Gagangeet Singh Aujla*, *Student Member, IEEE,* Rajat Chaudhary†, *Student Member, IEEE,*
Neeraj Kumar‡, *Senior Member, IEEE,* Ravinder Kumar§, and Joel J.P.C. Rodrigues‖, *Senior Member, IEEE*
*†‡§ Computer Science & Engineering Department, Thapar University, Patiala (Punjab), India
* Computer Science & Engineering Department, Chandigarh University, Gharuan (Punjab), India
‖National Institute of Telecommunications (Inatel), Santa Rita do Sapucaí, MG, Brazil;
Instituto de Telecomunicações, Portugal; University of Fortaleza (UNIFOR), Fortaleza, CE, Brazil
(e-mail: gagi_aujla82@yahoo.com, rajatlibran@gmail.com, neeraj.kumar@thapar.edu, ravinder@thapar.edu, and joeljr@ieee.org)

***Abstract*—In recent times, smart communities such as-smart grid, smart healthcare, and smart manufacturing units consists of large number of connected devices equipped with advanced processing and communication capabilities. The focus of these smart communities have shifted towards the use of intelligent processing and control for providing better quality of service (QoS) to the end user domain. To support this aspect, software-defined networking (SDN) is being widely deployed in different domains such as–data center networks, fog/edge computing, smart grid, and vehicular networks. The variable requirements of different applications in smart communities make it necessary to deploy flexible and scalable SDN. The dynamic flow management capability of SDN has lots of potential that needs to be effectively explored in order to provide QoS guarantee for traffic generated from different smart applications. In this direction, in this paper, an ensembled scheme for QoS-aware traffic flow management in SDN is designed. The proposed scheme works in three phases: 1) a linear ordering scheme for dependency removal of the incoming packets is designed, 2) an application-specific traffic classification scheme is designed, and 3) a queue management scheme is designed for efficient scheduling of traffic flow. The proposed scheme is evaluated over an experimental setup. The results obtained shows that the proposed scheme behaves effectively with respect to different QoS parameters.**

***Index Terms*—Classification, Decision Tree, Linear Ordering, Queue Management, QoS, Software-defined Networking.**

I. Introduction

Nowdays, smart communities such as-intelligent transportation system, smart grid, smart healthcare, and smart manufacturing comprise of large number of intelligent devices equipped with advanced processing and communication capabilities. Due to this reason, the smart communities have shifted their focus towards intelligent processing and control for providing better quality of service (QoS) to the end user domain. To support this aspect, the smart devices are connected with each other using advanced network and communication infrastructure [1]. For this purpose, wireless networks, cloud computing, and big data have emerged as key enablers for these smart communities. Moreover, the advent of internet of things (IoT) have emerged as a revolution to enhance the connectivity and scalability of smart communities. According to a recent report (CISCO global cloud index) [2], the trend show that IoT-based communities are expected to achieve 2.7 fold growth by 2020. Such an increase can lead to a 10-fold growth in the volume of data generated by intelligent devices deployed in smart communities.

Hence, in near future, high movement of huge volume of data generated by smart devices is expected for the smooth functioning of smart communities. However, the traditional TCP/IP-based network infrastructure may not be much effective to handle the dynamic and scalable requirements of such a huge volume of data generated by smart devices [3]. For real-time analysis and processing of data generated by these devices, the underlying network backbone need to support low latency, high data-rate and bandwidth guarantee. But, the use of traditional TCP/IP network infrastructure may elevate the challenges (poor bandwidth, link quality and high load) of real-time handling of data traffic generated from geo-seperated nodes [4]. Hence, a dynamic, customized, and scalable network and communication backbone is essential requirement for efficient functioning of various applications deployed in different smart communities [5].

For this purpose, software-defined networking (SDN) is widely adopted to handle different applications in smart communities due to its manifold capabilities such as–flexibility, programmability, and re-reconfigurability. In this direction, SDN has been deployed in various smart communities such as-data center networks [6], [7], vehicular networks [8], edge-cloud networks [9], [10], and SG systems [11]–[13]. By decoupling the data plane from the control plane, the complete decision making process is solely handled by a logically centralized controller. One of the most beneficial property of SDN is adaptable flow management according to the dynamic requirements and available resources. By exploiting this property of SDN, the incoming flow can be handled dynamically to meet the QoS requirements in different smart communities. As compared with the conventional best effort service models (integrated services, differentiated services, and multi-protocol label switching) that lack in traffic control, the SDN Controller provides with the guaranteed application-specific QoS requirements to process the huge volumes of data flows within the specified time interval.

However, in order to meet the QoS guarantees, the SDN controller has to face manifold challenges from variable communication requirements of different applications. Hence, a flexible and dynamic traffic flow management is required to handle the incoming packets from different applications

978-1-5386-3180-5/18/$31.00 ©2018 IEEE

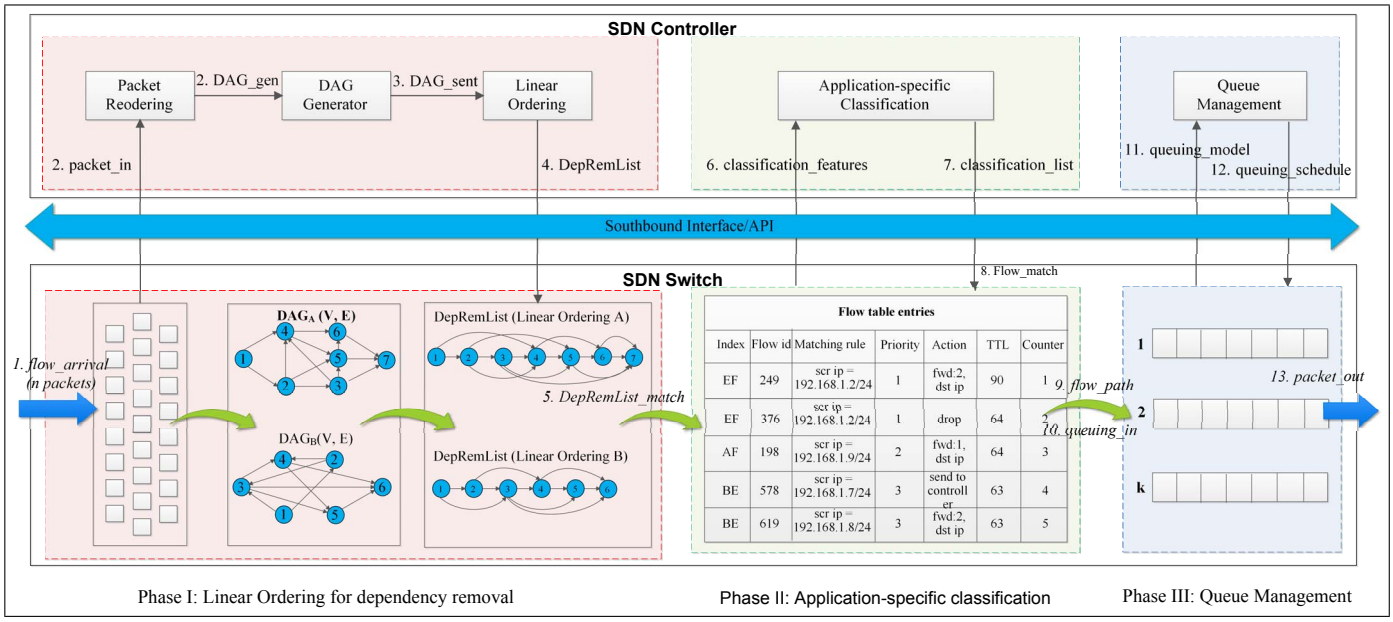


Fig. 1. System model of the proposed scheme

explicitly to provide the QoS guarantees. In this direction, many existing proposal have explored traffic management, routing, and flow scheduling in SDN. For example, Sood et al. [14] presented a distributed flow balancing scheme to minimize the network resources and operational costs. Similarly, Jin et al. [15] proposed a fair queuing scheme to reduce the Openflow switch scheduling issue through flow-level bandwidth provisioning. Xu et al. [16] proposed a real-time flow route update algorithm to address delay-satisfied route update problem. In [10], the authors proposed a workload consolidation scheme for load balancing at SDN switches in an edge-cloud environment. In an another work, Aujla et al. [9] proposed an application-specific workload slicing and flow scheduling algorithm in multi edge-cloud environment. In [8], the author proposed a data offloading scheme for flow balancing on the SDN network by using the priority manager and load balancer. After analysis of the aforementioned proposals, it is evident that an efficient scheme for QoS-aware traffic flow management in SDN for smart communities is the utmost requirement of the hour.

A. Contributions

Hence, in this paper, following contributions are presented.

1) A linear ordering scheme is designed for removal of dependencies among the incoming packets in the traffic flow at SDN switches.
2) An application-specific traffic classification scheme based on decision tree is designed. In this scheme, the incoming traffic flows are classified according to the different features.
3) Finally, a queue management scheme is designed for efficient scheduling of the classified traffic flow. In this scheme, a queue migration and priority shifting scheme are also designed for handling the congestion and starvation issues.

II. System Model and Problem Formulation

The proposed system model comprises of an SDN-based layered architecture with an SDN controller decoupled from underlying data plane. The data plane consists of OpenFlow switches (s having p ports) which works according to the instruction set provided by the controller. Fig. 2 shows the system model of the proposed scheme comprising of an OpenFlow switch and an SDN controller. The incoming traffic flow f consists of n packets arriving at OpenFlow switch that are 1) linearly ordered, 2) classified, and 3) queued for QoS-aware scheduling using the proposed scheme. The preliminaries associated with the proposed model are described as below.

In order to schedule the incoming packets, the most important QoS parameter is latency. Hence, it is utmost important to achieve low latency for a better QoS provisioning to the incoming traffic flow. Now, latency is dependent on delays at various levels such as– propagation (τ_{pg}), serialization (τ_{sr}), queuing waiting (τ_{wt}), processing (τ_{pr}), jitter (τ^{jit}), and migration (τ^{mig}). So, the latency associated with n^{th} incoming packet at and OpenFlow switch is defined as follows.

$$\tau_n = \tau_{prop} + \tau_{ser} + \tau_{wt} + \tau_{proc} + \tau_{jit} + \tau_{mig} \tag{1}$$

Now, the τ_{prop} presented in Eq. 1 depends on the distance (D_{s_1,s_2}) between two switches (s_1, s_2) and is defined as below.

$$\tau_{prop} = \sum_{p} \frac{D_{s_1,s_2} \times \psi_{f,s_1}(t) \times \psi_{f,s_2}(t) \times \tilde{a}_{s_1,s_2}}{\tau_{med}(t)} \tag{2}$$

where, $\psi_{f,s_1}(t)$ and $\psi_{f,s_2}(t)$ represents decision variables for flows related to s_1 and s_2, respectively, $\tilde{a}_{s_1,s_2}$ represents a binary variable for adjacency between s_1 and s_2, and $\tau_{med}(t)$ is the propagation delay of the medium.

Similarly, τ_{ser} depends on the packet size ($\Im_n$), bandwidth (β_n), and occupancy ratio (θ_n) and is defined as below.

$$\tau_{ser} = \sum_{s} \sum_{p} \frac{\Im_n(t)}{\beta_s \times \Theta_s(t)} \tag{3}$$

Now, τ_{wt} is defined as below.

$$\tau_{wt} = \sum_{s} \sum_{p} \frac{|q(t)|}{B_s \times \Theta_s(t)} \tag{4}$$

where, $|q(t)|$ denotes ready queue.

In Eq. 1, processing delay (τ_{pr}) is defined as below.

$$\tau_{proc} = \sum_{s} \tau^{i}_{proc}(t) \times \sum_{f} (t^{e}_{f} - t^{s}_{f}) \times \psi_{f,s_i}(t) \tag{5}$$

where, $\tau^{i}_{proc}(t)$ denotes processing delay by i^{th} switch, t^{en}_{f} is end time of f^{th} flow, and t^{st}_{f} is start time of f^{th} flow.

Now, some times, a queue consisting of *n* packets, may migrate the packets to another queue (inter-queue or intra queue). In such a case, an additional delay (τ_{mig}) is incurred that is defined as below.

$$\tau_{mig} = \ell_{net} \varphi_i(t) \tag{6}$$

where, $\varphi_i(t)$ is the migration rate.

The inter-arrival time of incoming packets has a strong impact on overall latency. In this direction, jitter is also considered and is given as below.

$$\tau_{jit} = \tau_{n2n} + 2 \times \delta \times \Re \tag{7}$$

where, τ_{p2p} is the deterministic jitter, $\Re$ is the random jitter, δ depends on bit error rate required of the link.

Using the above concept, the objective function is formulated as as below.

$$min\,(\tau_n) \tag{8}$$

subject to the following constraints

$$0 < \sum_{n} q_n(t) S_n \leq P_s \lambda(t) \tag{9}$$

The proposed scheme works in three phases (described in subsequent sections) in order to achieve the above objective.

III. Phase I: Linear Ordering

The SDN Controller is the network brain which establish a TCP connection with the Openflow switches via a southbound interface in order to maintain the updated information of the number of active switch ports along with their switch ID and ethernet address. The controller sends the $packet\_out$ messages at each Openflow switches which contains the specific rule matching information how to forward the packets from the source port to the destination port. The independent $packets\_out$ messages is simply matched with the flow tables at every switches and after exact matching the packet is forwarded to the neighbor switch port number but if the packet does not matches then the $packet\_in$ message with the switch information is send to the controller for rebuilding a new flow rule. The independent packet can be easy routed towards the destination node through multiple switches available at that particular time but the major issue arises with the dependency packets. The issue that arise with the dependent packets is that the packets may departs out of order which may incurs significant delay in the packet queuing. To understand the issue, let us consider a small case study as given below.

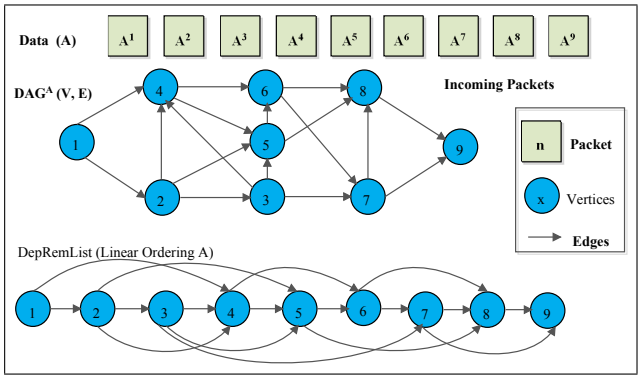


Fig. 2. Linear Ordering of arrived packets

Case Study: Consider a flow switch packet scheduling consists of a message which is divided into four packets say (A, B, C, D). The D packets is an independent packet while the (A, B, C) is a dependency packet list in which the dependency among packets is $(A \rightarrow B), (B \rightarrow C), and\ (B \rightarrow C)$. Therefore, three cases may be formed in packet transmission which are listed as follows.

- *Case 1*: Suppose, the packet A is transmitted first then packet B proceeded by packet C. At the receiving port, the packet A is received but the packet B, C is on the way. The task is incomplete without B, C because A is dependent on B and B is dependent on C. Therefore, the resources are totally wasted as A has to wait till B, C packet arrives.
- *Case 2*: For every message, first, the packet dependencies is checked and then they are forwarded based on their linear ordering. The packet C is forwarded first followed by B and then A and at last. Now, the packet D is forwarded. The packet C is totally independent hence, the task is completed after receiving the independent packets at the output port. But, again the issue arises as packet D is also an independent packet and has to wait for a long duration till the dependent packet are transmitted.
- *Case 3*: For every message, the independent as well as dependent packet are first analyzed then only initiates the packets reordering process. Therefore, the packet D is to be transmitted first instead of waiting till the dependent packets are served in the linear order.

For every dependency packets, the scheduling of a sequence of packet is reordered with the help of a directed acyclic graph (DAG): $G(V, E)$. Fig. 2 shows two different types of data as: data (A) and data (B). The data (A) is partitioned into nine packets and is to be transmitted from the source port to the destination port through multiple routes. The packets $\{A_1, A_2, ..A_9\}$ is first represented with the help of a DAG based on their dependencies. Then, a packet reordering scheme is used to return a final stack in a linear sequence of the arrived packets. Algorithm 1 shows the working of the proposed linear ordering scheme where *n* represents the number of packets of a data. The steps followed in the algorithm are given as below.

- Initialize a two-dimensional matrix as $PacketDAG[n][n]$ and assign the degree of each node row-wise by counting the out_degree of a node.

- Column-wise sum of every node from $\{1, 2, .., n\}$.
- Now, check column-wise in $D[packet]$ array if any node contains a value zero then insert that node into an empty stack S. Hence, build a stack in a sorted sequence.
- The dependent nodes is now removed from the stack S and the process continues till the stack is not empty. Remove (pop) the top node from the stack and push it into a new list named as $DepRemList$.
- Set as zero on all the elements row-wise in the poped node. Recompute the sum again column-wise and now check if the node which contains the value zero then again push the node into the stack S. Repeat the procedure till the dependency list is being removed and finally returns the output ordered stack as DepRemList[].

Algorithm 1 Dependency Removal Scheme

Input: $PacketDAG[n][n]$.
Output: $DepRemList$.

procedure FUNCTION
  D[packet] = $\sum_{packet=1}^{n} PacketDAG[Col]$;
  for $(packet = 1; packet \leq n; packet$++) **do**
    if $(D[packet] == `Null')$ **then**
      $PUSH : S \leftarrow packet$;
      return packets inserted;
    end if
  end for
  while (S is not empty) **do**
    $POP : DepRemList[\,] \leftarrow S[Top]$;
    Set 0 all elements of $PacketDAG[row]$;
    temp = $\sum_{packet=1}^{n} PacketDAG[Col]$;
    if temp = 0 **then**
      $PUSH : S \leftarrow packet$;
    end if
  end while
end procedure

IV. Phase II: Application Specific Classification

In this section, a decision tree based application specific classification scheme is designed. In this scheme, the incoming traffic flow is classified on the basis of various features such as–order index, port number, packet size, etc. The incoming packets are classified as per different priorities based on above mentioned features. The classified packets vary from 1 to *k* on the basis of priorities defined by decision tree. Initially, the decision tree is trained according to the selected features. The trained decision tree is implemented in the SDN controller for identification of different packet classes. It is assumed that the trained data set of packets *P* comprise of *k* type of application types. On this basis, the entropy of *P* is defined as below.

$$\varepsilon(P) = -\sum_{i=1}^{k} \rho_i log_2 \rho_i \tag{10}$$

where, ρ_1, ρ_2,..., ρ_k denotes the probability of each type of application appearing in the dataset of *P*.

The value of entropy directly effects the information uncertainty. The small value of entropy depicts a lower information uncertainty. This depicts that *P* is concentrated in few application types. Now, *P* is divided into *m* subsets as per attribute *x*, and the expected entropy of *x* for *P* is as.

$$\varepsilon(P|x) = -\sum_{i=1}^{k} p(P_i)\varepsilon(P_i) \tag{11}$$

Now using the entropy and expected entropy, the information gain is defined as below.

$$\text{Ģ}(P, x) = \varepsilon(P) - \varepsilon(P|x) \tag{12}$$

Now, using the value of Ģ(P,x), an information gain ratio is defined as below.

$$\text{Ģ}_{ratio}(P, x) = \frac{\text{Ģ}(P, x)}{\text{Ş}(P, x)} \tag{13}$$

where, Ş(P,x) denotes split information of *x* with respect to *P*.

The split information, Ş is given as below.

$$\text{Ş} = -\sum_{i=1}^{j} p(P_i) log_2 p(P_i) \tag{14}$$

Using these aspects, the application specific classification algorithm is designed as below.

Algorithm 2 Packet Classification Scheme

Input: P: set of training dataset, x: set of candidate attributes, A(x): all attributes of x.
Output: Decision_Tree($T\{\}$).

procedure FUNCTION
  root[Tree] = $Null$;
  q = 0;
  extract A(x);
  if (*S*== Null || tuples in *S* are all of the same class) **then**
    return *L* as a leaf node which belongs to the same class;
  end if
  while (A(x) ¿ 1) **do**
    Compute $\varepsilon(P)$;
    for $(p = 1; p \leq A(x);$ p++) **do**
      Compute $\varepsilon(P|x), \text{Ģ}(P, x), \text{Ģ}_{ratio}(P, x)$;
      if $(\text{Ģ}_{ratio}[P, x] > q)$ **then**
        $q \leftarrow \text{Ģ}_{ratio}[P, x]$;
        $r\{max\} = p$;
      end if
    end for
    $r\{max\} = Max(\text{Ģ}_{ratio}(P, x))$;
    Build $r\{max\}$ as parent node;
    $S_{subset} \leftarrow$ split S into k decision attributes;
    for $(p = 1; p \leq k;$ p++) **do**
      $T_k \leftarrow (S_{subset} - r\{max\})$
      Join T_k with the corresponding *T*
    end for
  end while
  return $T\{\}$;
end procedure

V. Phase III: Queue Management

In the section, the queuing model is formulated for an OpenFlow switch based on M/G/1 priority queue model for reducing the waiting time of the incoming packets.

A. Preliminaries and Assumptions

The preliminaries and assumptions for proposed queuing model are given as below.

- Packets are partitioned into k priority classes ($k = \{1, 2, ..., K\}$) at each queue (q) of a switch (s).
- The arrival rate of different classes are $\{\lambda_1, \lambda_2, ..., \lambda_k\}$, respectively and they follow Poisson process.
- Class k packets follow general distribution for the service rate with mean S_k and second moment ${S_k}^{(2)}$.
- Class *1* and K have highest and lowest priorities.
- The service is non-preemptive, i.e, the current packet is served, even if a higher priority packet arrives.
- Probability density function (*PDF*) for service rate of a packet of class i is assumed as $g_i(x)$

TABLE I
NOTATIONS

Notation	Description
k	Number of priority classes
λ^k	Arrival rate of class *k* packets
S_k	Service rate of class *k* packets
R	Mean residual service time for arriving labeled packets
τ_{wt}^k	Mean waiting time of class *k* packets
$N_{i,k}$	Mean number of waiting class *k* packets in the queue
$M_{i,k}$	Mean higher class *i* packets arriving in class *n* packets τ_{wt}^n
ρ_k	Load of class *k*
X_i	Service time
$g_i(x)$	Probability density function for service time of class *i* packets

Theorem: To calculate the average waiting time for packets of class k, denoted by W_k for $1 \leq k \leq K$. The average waiting time for newly labeled packet is defined as follows:

$$\tau_{wt}^k = R + \sum_{i=1}^{K} s_i \times (N_{i,k} + M_{i,k}) \; where \; 1 \leq k \leq K. \tag{15}$$

Proof: The stability condition of the queue is: $\rho_1 + \rho_2 + ... + \rho_k < 1$. For this, a newly arriving packet of class k is considered as *labeled packet*. Now, τ_{wt}^k of this packet consists of the following components:

- Time taken by the packet already in service when the labeled packet arrives.
- Time taken by the packets in the queue waiting to receive service before the labeled packet.
- Time taken by the packets which arrive at the switch after the labeled packet but are serviced before it.

Firstly, R i.e. the mean residual service time at the switch upon arrival of the labeled packets of class i is calculated using Kleinrock's method. The probability of arrival of a packet during the service of a class i packet with duration x is proportional to both x and *PDF* $g_i(x)$. The *PDF* $\hat{g}_i(x)$ for the service time $\hat{X}_i$ of a packet of class i is usually different from $g_i(x)$. This is due to the reason that the longer service times cover more of time axis than shorter service times. This arises more chances of the arrival of an arbitrary call during longer service time which is given as below.

$$\hat{g}_i(x) = cx \times g_i(x) \tag{16}$$

where, c is a constant.

Now, under the normalization condition,

$$1 = \int_0^\infty \hat{g}_i(x)dx = c\int_0^\infty x g_i(x)dx = cs_i, where \; c = 1/s_i \tag{17}$$

$$\hat{g}_i(x) = \frac{x g_i(x)}{s_i} \tag{18}$$

Here, the labeled packet arrives at the time when a class i packet is being served having duration x. The point of arrival is uniformly distributed over the time interval $[0, x]$. Using this, the *PDF*, $h_i(y)$ for the remaining service time Y_i of class i packet is given as below.

$$P[Y_i \leq y] = \frac{y}{x}, \quad 0 \leq y \leq x \tag{19}$$

The joint density of $\hat{X}_i$ and Y_i is given as below.

$$P[y < Y_i \leq y + dy, x < \hat{X}_i \leq x + dx] = \frac{dy}{x}\frac{x g_i(x)dx}{s_i} = \frac{g_i(x)dydx}{s_i} \tag{20}$$

where, $0 \leq y \leq x$.

We obtain the *PDF* for Y_i by integrating over x as listed below.

$$h_i(y) = \int_{x=y}^{x=\infty} \frac{g_i(x)dx}{s_i} = \frac{1}{s_i}\int_0^\infty g_i(x)dx = \frac{1 - G_i(y)}{s_i} \tag{21}$$

where, $G_i(y) = \int_0^y g_i(x)dx$ is the distribution function of X_i.

The expectation of Y_i mentioned as below.

$$E(Y_i) = \int_0^\infty y h_i(y)dy = \frac{1}{s_i}\int_0^\infty y[1 - G_i(y)]dy = \frac{{s_i}^{(2)}}{2s_i} \tag{22}$$

Now, $\rho_i = \lambda_i \times s_i$, the load or the fraction of time the switch is occupied by class i packets. It indicates the probability that the labeled packet finds a class i packet in service and the mean time till the service is completed is given by $\frac{{s_i}^{(2)}}{2s_i}$. Therefore, R is given as below.

$$R = (1 - \sum_{i=1}^{K} \rho_i) \times 0 + \sum_{i=1}^{K} \rho_i \frac{{s_i}^{(2)}}{2s_i} = \frac{1}{2}\sum_{i=1}^{K} \lambda_i \times {s_i}^{(2)} \tag{23}$$

where, $(1 - \sum_{i=1}^{K} \rho_i)$ is the probability that there are no packets in service when the labeled packet arrives.

Here, it is considered that the packets of classes $\{k+1, k+2, ..., K\}$, i.e., lower priority than the labeled packet are served after the labeled packet. Also, the other packets of classes $\{1, 2, .., k\}$, i.e., higher priority that are already in the queue are served before the labeled packet.

$$N_{i,k} = 0, for \; k+1 \leq i \leq K \tag{24}$$

Now, according to little's law, the mean number of class i packets already in the queue at an arbitrary time is $\lambda_i \tau_{wt}^i$. So, $N_{i,k} = \lambda_i \tau_{wt}^i$, where, $1 \leq i \leq k$. Moreover, to calculate $M_{i,k}$, the packets of lower priority class that arrive while the labeled packet is waiting, do not delay the labeled packets, i.e., $M_{i,k} = 0$, for $k+1 \leq i \leq K$. The packets of classes $\{1, 2, .., (k-1)\}$ (higher priority) that arrive in the queue, while the labeled packet is waiting are served before it. Since, the average time that the labeled packets spends in the waiting queue is τ_{wt}^n, the average arrivals of class i packets is $\lambda_i \tau_{wt}^k$. Hence, $M_{i,k} = \lambda_i \tau_{wt}^k, 1 \leq i \leq k$. Now using Eqs. 23 and 24, τ_{wt}^k is given as:

$$\tau_{wt}^k = R + \sum_{i=1}^{k} s_i \lambda_i \tau_{wt}^i + \sum_{i=1}^{k} s_i \lambda_i \tau_{wt}^k \tag{25}$$

Hence, proved.

Now, expanding the above equation again, we get.

$$\tau_{wt}^k = R + \sum_{i=1}^{k-1} \rho_i \tau_{wt}^i + \tau_{wt}^k \sum_{i=1}^{k} \rho_i, \quad 1 \leq k \leq K \tag{26}$$

and can be formulated as:

$$\tau_{wt}^k = \frac{R + \sum_{i=1}^{k-1} \rho_i \tau_{wt}^i}{1 - \sum_{i=1}^{k} \rho_i}, \quad 1 \leq k \leq K \tag{27}$$

This set of simultaneous equations can be solved to find $\{\tau_{wt}^1 + \tau_{wt}^2 + .. + \tau_{wt}^k\}$ listed as follows:

$$\tau_{wt}^1 = \frac{R}{1 - \rho_1}, \tau_{wt}^2 = \frac{R}{(1 - \rho_1)(1 - \rho_1 - \rho_2)} \tag{28}$$

$$\tau_{wt}^k = \frac{R}{(1 - \sum_{i=1}^{k-1} \rho_i)(1 - \sum_{i=1}^{k} \rho_i)} \tag{29}$$

$$\tau_{wt}^{k} = \frac{\sum_{i=1}^{K} \lambda_i s_i^{(2)}}{2(1-\sum_{i=1}^{k-1} \lambda_i s_i)(1-\sum_{i=1}^{k} \lambda_i s_i)}, \quad 1 \le k \le K \tag{30}$$

For M/G/1 queue having multiple classes that follow non-preemptive discipline, the relationship is defined as follows:

$$\sum_{k=1}^{K} \rho_k \tau_{wt}^{k} = \frac{\rho R}{1-\rho} \tag{31}$$

where, $\rho = \sum_{k=1}^{K} \rho_k$ is the total load of the system.

This is called as Kleinrock's conservation law.

B. Queue Migration Scheme

In the proposed queue management scheme, in case when the waiting time in a queue is large, then the packet is migrated to another queue. For this purpose, a queue migration scheme is designed. Two types of migrations; inter-queue migration (migration of packets to queues at different switch), and intra-queue migration (migration of packets between queues at same switch). The proposed algorithm 3 is designed for the proposed migration scheme. In this algorithm, the average waiting time is calculated and compared with the average waiting time of the queues. If the sum of average waiting time and migration time at a n^{th} queue at a switch is less than the host queue, then the packet is migrated to the n^{th} queue.

Algorithm 3 Queue Migration

Input: p, q_1.
Output: q_n.

```
for (p=1; p ≤ n; p++) do
    for (q=1; q ≤ n; q++) do
        Compute τ^k_wt                          ▷ Waiting time all available queues q2
        Compute τ_mig(q1 → qn)                  ▷ Migration time from q1 to q2
        τ^k_wt(qn) + τ_mig(q1 → qn)
        Sort the total time in ascending order.
        if (τ^k_wt(q1) > τ^k_wt(q2) + τ_mig(q1 → q2)) then
            Migrate p to q2
        else
            Wait in q1
        end if
    end for
end for
```

C. Probability Shifting Scheme

There may be case when a packet of lower priority class gets blocked in the queue in order to serve the packets of higher priority classes. Such packet may not be served for a long period of time. To handle such a situation, a probability shifting scheme for breaking the blockage of a lower priority packet is designed. In this scheme, the average waiting time of n^{th} lower probability packet is compared with a threshold time. The threshold time is calculated as below.

$$\tau_{thr} = \frac{\tau_{wt}}{n} \tag{32}$$

where, τ_{wt} is the average waiting time of a queue.

If the average waiting time of n^{th} lower priority packet is more than the threshold, then the priority of the packet is reduced by *1*. This step is repeated till this packet reaches the top of the queue and scheduled thereby.

Algorithm 4 Priority Shifting

Input: p, Priority(P_k).
Output: Top priority.

```
for (p=1; p ≤ n; p++) do
    while p=k do
        if (τ^n_wt > τ_thr then
            p_k = p − 1 return p_k
        end if
    end while
end for
```

VI. Results and Discussion

The proposed scheme is evaluated over an experimental setup with one OpenFlow Switch (3.3 GHZ, 8 cores, 8GB RAM, 5×100 Mbps NIC), and an SDN controller (3.3 GHZ, 8 cores, 8GB RAM, 1×100 Mbps NIC). The proposed setup is tested using four nodes (3.3 GHZ, 8 cores, 8GB RAM, 1×100 Mbps NIC) with variable traffic load having different sending rates. A variable traffic (10-100 Mbps) is generated from node *1* having frame size of 1000 Bytes. The experimental setup is tested over three types of flows; 1) Voice (high-priority), 2) Video (Medium priority), and 3) Data (lowest priority) flows. A cross sequence pattern is followed to sent the three types of flows. The generated traffic flows are ordered linearly for dependency removal and then classified using decision tree scheme. Finally, the queuing scheme is used to reduce the waiting time of the packets. The proposed queuing scheme scheme is also compared with FIFO queue scheme and a case of the proposed scheme without linear ordering.

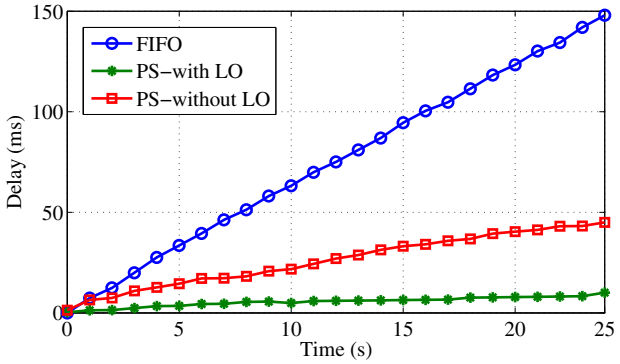
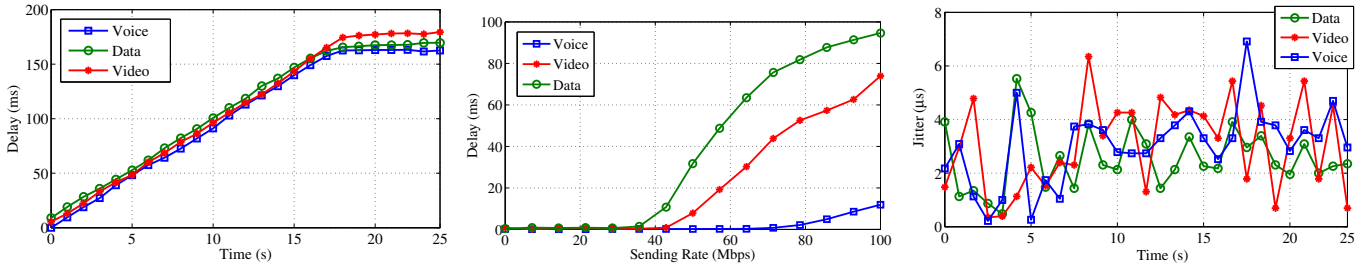


Fig. 3. Comparative analysis

Fig. 3 shows the delay incurred for the incoming traffic for proposed scheme (PS-with LO), FIFO queuing scheme, and proposed scheme without linear ordering (PS-without LO). It is clearly visible, that the proposed scheme out date the other two counterparts with a high margin. After this, Fig. 4(a) shows that the delay incurred for each flow for the designed experimental setup. The voice flow shows lowest delay as the it it assigned a highest priority by the proposed classification scheme. Similarly, the variation of delay for all three flows with respect to sending rate is shown in Fig. 4(b). The data flow shows highest delay as it is assigned lowest priority by classification scheme. Now, the variation of jitter with respect to all the three flows is shown in Fig. 4(c). Finally, the throughput for all the three traffic flows is evaluated. Fig. 5 shows the variation of throughput with respect to time for all the three types of flows.



(a) Delay (b) Delay vs Sending rate (c) Jitter

Fig. 4. Evaluation results

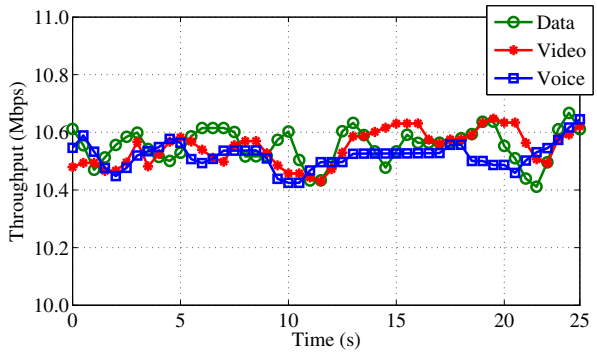


Fig. 5. Throughput

VII. Conclusion

In this paper, an QoS-aware SDN-based traffic flow management scheme is designed. The proposed scheme is divided into three phases wherein each phase aims to reduce the delay for the incoming flow. In first phase, the incoming packets are linearly ordered to remove inter-packet dependency. This avoids the waiting time of the packets at the destination. In second phase, the ordered packets are classified with respect to application type, priority, and packet size. In the third phase, a priority-based queuing model is formulated to manage the waiting time of the packets. In this queuing model, queue migration and priority shifting schemes also deigned in order to the congestion and starvation issues. The proposed scheme is evaluated on an experimental setup. The results obtained clearly verifies the effectiveness of the proposed scheme with respect to various QoS parameters. The proposed scheme shows lower delay in contrast to other schemes.

Acknowledgement

This work was supported by the National Funding from the FCT - Fundação para a Ciência e a Tecnologia through the UID/EEA/50008/2013 Project, by the Government of Russian Federation, Grant 074-U01, and by Finep, with resources from Funttel, Grant No. 01.14.0231.00, under the Centro de Referência em Radiocomunicações - CRR project of the Instituto Nacional de Telecomunicações (Inatel), Brazil.

References

[1] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-defined industrial internet of things in the context of industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, Oct 2016.

[2] Cisco global cloud index: Forecast and methodology, 2015-2020 white paper. Cisco Public. Accessed on: March 2017. [Online]. Available: http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf

[3] D. Li, Z. Aung, J. Williams, and A. Sanchez, "P3: Privacy preservation protocol for automatic appliance control application in smart grid," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 414–429, 2014.

[4] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "UbiFlow: Mobility management in urban-scale software defined IoT," in *IEEE Conference on Computer Communications*, 2015, pp. 208–216.

[5] Y. Yan, Y. Qian, H. Sharif, and D. Tipper, "A survey on smart grid communication infrastructures: Motivations, requirements and challenges," *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 5–20, 2013.

[6] G. S. Aujla and N. Kumar, "SDN-based energy management scheme for sustainability of data centers: An analysis on renewable energy sources and electric vehicles participation," *Journal of Parallel and Distributed Computing*, 2017, doi: 10.1016/j.jpdc.2017.07.002.

[7] G. S. Aujla, A. Jindal, N. Kumar, and M. Singh, "SDN-based data center energy management system using RES and electric vehicles," in *IEEE Global Communications Conference*, 2016, pp. 1–6.

[8] G. S. Aujla, R. Chaudhary, N. Kumar, J. J. Rodrigues, and A. Vinel, "Data offloading in 5G-enabled software-defined vehicular networks: A stackelberg-game-based approach," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 100–108, 2017.

[9] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Rajan, "Optimal decision making for big data processing at edge-cloud environment: An SDN perspective," *IEEE Transactions on Industrial Informatics*, 2017, doi: 10.1109/TII.2017.2738841.

[10] G. S. Aujla and N. Kumar, "MEnSuS: An efficient scheme for energy management with sustainability of cloud data centers in edge-cloud environment," *Future Generation Computer Systems*, 2017, doi: 10.1016/j.future.2017.09.066.

[11] S. Al-Rubaye, E. Kadhum, Q. Ni, and A. Anpalagan, "Industrial internet of things driven by SDN platform for smart grid resiliency," *IEEE Internet of Things Journal*, 2017, doi: 10.1109/JIOT.2017.2734903.

[12] S. Wang and X. Huang, "Aggregation points planning for software-defined network based smart grid communications," in *IEEE Conference on Computer Communications*, 2016, pp. 1–9.

[13] A. Aydeger, K. Akkaya, M. H. Cintuglu, A. S. Uluagac, and O. Mohammed, "Software defined networking for resilient communications in smart grid active distribution networks," in *IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.

[14] K. Sood, S. Yu, Y. Xiang, and H. Cheng, "A general qos aware flow-balancing and resource management scheme in distributed software-defined networks," *IEEE access*, vol. 4, pp. 7176–7185, 2016.

[15] H. Jin, D. Pan, J. Liu, and N. Pissinou, "Openflow-based flow-level bandwidth provisioning for cicq switches," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1799–1812, 2013.

[16] H. Xu, Z. Yu, X.-Y. Li, L. Huang, C. Qian, and T. Jung, "Joint route selection and update scheduling for low-latency update in sdns," *IEEE/ACM Transactions on Networking*, 2017.