future internet

MDPI

Article

Smart Site Diversity for a High Throughput Satellite System with Software-Defined Networking and a Virtual Network Function

Gandhimathi Velusamy * and Ricardo Lent *

College of Technology, University of Houston, Houston, TX 77204, USA

* Correspondence: gvelusamy@uh.edu (G.V.); rlent@uh.edu (R.L.); Tel.: +1-713-743-4239 (R.L.)

Received: 31 October 2020; Accepted: 2 December 2020; Published: 7 December 2020

Abstract: High Throughput Satellite (HTS) systems aim to push data rates to the order of Terabit/s, making use of Extremely High Frequencies (EHF) or free-space optical (FSO) in the feeder links. However, one challenge that needs to be addressed is that the use of such high frequencies makes the feeder links vulnerable to atmospheric conditions, which can effectively disable channels at times or temporarily increases the bit error rates. One way to cope with the problem is to introduce site diversity and to forward the data through the gateways not affected, or at least less constrained, by adverse conditions. In this paper, a virtual network function (VNF) introduced through reinforcement learning defines a smart routing service for an HTS system. Experiments were conducted on an emulated ground-satellite system in CloudLab, testing a VNF implementation of the approach with software-defined networking virtual switches, which indicate the expected performance of the proposed method.

Keywords: reinforcement learning; learning automata; Q-Learning; SDN; VNF; routing; high throughput satellites; smart gateway diversity

1. Introduction

The networking industry has been dominated by the use of proprietary hardware appliances built with application-specific integrated circuits (ASIC). Network hardware appliances deliver one or more specific network functionality, including routing, intrusion detection, and traffic shaping. Network Function Virtualization (NFV) aims to replace this traditional approach to networking infrastructure by decoupling network functions from the hardware appliances, with software functions running as modular software on commercial off-the-shelf (COTS) servers with virtualization. The modular approach allows implementing virtualized network functions (VNF) as connectable blocks that can be chained to define complex services. NFV brings several benefits such as elasticity, availability, reliability, shorter deployment times, and reduced capital and operational expenditures compared to the traditional methods [1].

While less than a decade old, the NFV concept has gained rapid acceptance to support diverse applications but mostly for the terrestrial domain. Recent works have started the discussion of the possible benefits of NFV and Software-Defined Networking (SDN) for satellite communication networks, for example, to achieve high rates in satellite–5G applications with wide-scale coverage and high availability. It is expected that the use of Extremely High Frequencies (EHF) or free-space optical (FSO) in the feeder links will push the data rates to the order of Terabit/s in the future High Throughput Satellite (HTS) systems. However, one limitation is that the resulting performance may not be consistent because, with such high frequencies, atmospheric attenuation due to clouds, fog, and rainfall can cause severe time-varying channel impairments [2]. Even with the use of Adaptive

Coding Modulation (ACM) or improved error correction, codes can disable or significantly reduce the performance of the satellite link in the event of bad atmospheric conditions [3].

This work investigates an intelligent mitigation approach that is defined as a VNF to deal with the channel impairments of an HTS system exploiting spatial diversity for the feeder link. This approach allows forwarding the feeder link communications through selected gateways from a set of geographically distributed alternatives that are interconnected through high-speed terrestrial links, with the gateways separated by at least 20 km. While this possibility has been discussed in the literature [2,3], the prevailing approach to the problem is static, which involves manual interventions to change the forwarding policy with the knowledge of the system and transmission medium conditions. This work introduces a reinforcement learning (RL) approach that autonomously decides how to set the data forwarding policy without requiring the knowledge of weather conditions. The proposed method not only allows quick routing adaptation to the changes to precipitation fades in the Q/V bands but also to the traffic carried by the different feeder links. We implemented the smart routing method as an SDN controller that runs as a virtual network function (VNF). The deployment of the proposed VNF for onboard controllers helps to improve the reliability and availability of the feeder links.

2. Background

Bandwidth scarcity is one of the technical challenges to be addressed in future HTS systems. The entire Ka frequency band was allocated to the user link to prevent the bandwidth limitation, moving the feeder link to EHF (or FSO) with spatial diversity of the gateways to combat the channel impairments that may randomly appear due to atmospheric conditions [4]. Various approaches to gateway redundancy were proposed to prevent the need for doubling the infrastructure required for the ground segments. These approaches assumed that all gateways are interconnected by high-speed terrestrial links, and the traffic of a given user can be redirected to another gateway. These approaches, denoted as *Smart Gateway Diversity*—SGD [5,6] use Network Control Centers to make the routing decision of serving each user beam by, 1. carriers from all (N+0 site diversity), 2. a subset of the gateways using P redundant gateways (N+P site diversity) so that the traffic of a gateway experiencing an outage can be routed through an alternative gateway, or 3. the concurrent operations of all gateways, including the redundant ones (++N+P site diversity).

While most of the works about SDN/NFV integration focus on traffic steering towards VNF instances that implement network functions, only limited works exist for implementing an SDN controller as a VNF. A systematic literature review [7] lists possible controller functions implemented as VNF in existing works. An SDN controller that functions as a traffic load balancer was implemented as a VNF [8]. The controller was implemented as a virtual network service to deploy additional controllers based on the network traffic to improve the overall network performance. An SDN controller that load balances traffic between two snort (intrusion prevention systems) VNF instances based on a control-theoretic approach was proposed [9]. A deep learning-based traffic classifier was implemented as a VNF directs an SDN controller to route traffic with application awareness [10]. An elastic routing service implemented as a Ryu controller is deployed as a VNF to load balance the network traffic across dynamically provisioned switches in a framework called UNIFY ESCAPE [11].

On the other hand, a few works have explored the application of Software Defined Networking, Virtualization, and Network Function Virtualization to satellite networks. For example, Bertaux et al. [12] suggested likely broadband communication scenarios for these technologies, including an inter-hub handover with site diversity case, enhancements for virtual network operator (VNO) services, and the integration of satellite and terrestrial networks. In another work, Gardikis et al. [13] emphasized the use of NFV to ensure the competitiveness of the satellite communications sector. Several approaches are certainly possible for this integration. Li et al. [14] focused on the use of SDN/NFV to orchestrate the control, forwarding, access, service, and management planes operating at different orbits of the space segment. A system architecture for the Internet of Space Things/CubeSats (IoST) was proposed based on SDN and NFV to improve

network resource utilization and simplify network management [15]. Prior works such as the one by Cai et al. [16] have formalized the optimization problem to be handled by the VNF. However, these approaches require global knowledge of the link states.

3. SDN/VNF HTS Architecture

In this work, the number of required gateways N is increased with additional P gateways, which provide redundant paths and capacity. As with the ++N+P site diversity scheme, the $n = N + P$ gateways are concurrently active. Routing occurs at the payload level (i.e., via packet switching) as decided by a VNF. No particular constraints are assumed about the symmetry of the links, so the system model applies to diverse applications. The user beams and the gateway beams are assumed to occur in different bands, with the latter using either EHF (Q/V band) or FSO. The user beams are located at a lower frequency band (e.g., Ka-band), so it is less affected by atmospheric attenuation than the feeder links.

For each user, two routing decision elements are required for each communication direction (i.e., data from or towards the user). Examples of the services include high-definition video streaming (i.e., data flowing mainly towards the user) and a large userbase generating and reporting data to an Internet-of-things cloud server (i.e., data flowing from the user to the gateways). Figure 1 depicts the main components of the system. Each routing decision element consists of an intelligent agent that learns how to select the optimal routing decision just by observing the performance of the past transmissions. As a result, one distinct feature of the proposed approach compared to the existing routing methods considered for the HTS is that it does not require the knowledge of the current system state, including the atmospheric conditions at the gateway sites or the bit-error-rates of the channels. Another advantage is that the agent does not need to be aware of physical-layer changes, such as those handled by adaptive-coding modulation. Through the learning mechanism, the agent observes the outcome of those changes and modifies its forwarding policy. This functionality is achieved by defining agents as VNF that implement reinforcement learning.

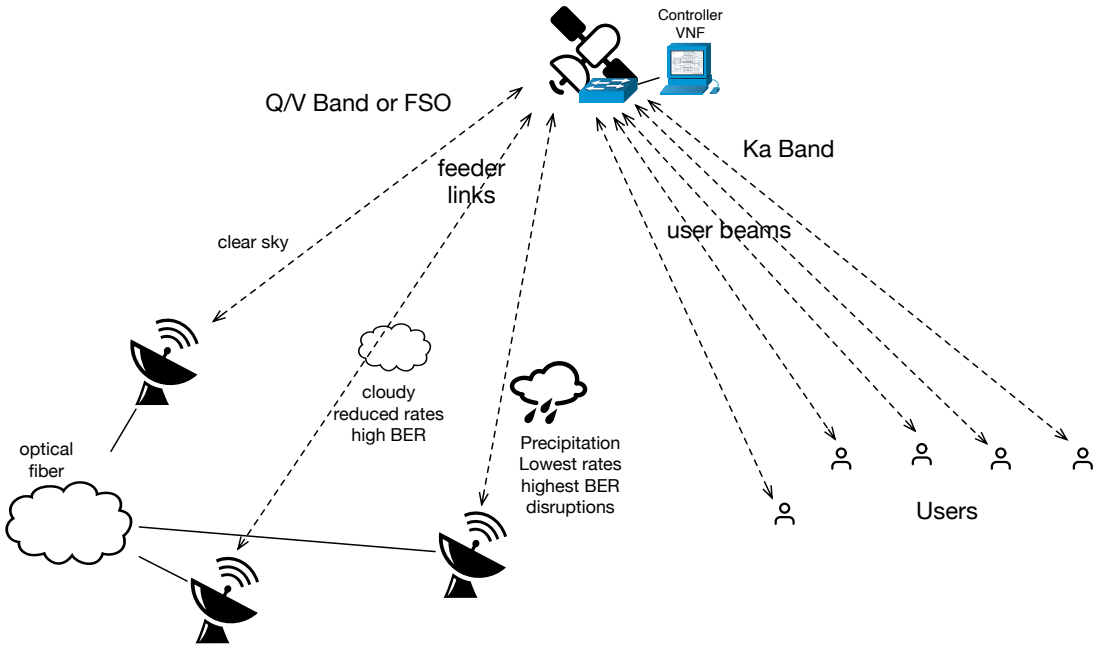


Figure 1. System architecture with a virtual network function (VNF) providing a routing function for a High Throughput Satellite (HTS) system.

Agent Learning Goal

User communications occur as a result of requests originated from the clients with data flowing in either or both directions. The routing goal is to minimize the completion time of these requests (i.e., the response time) and their loss ratio. The agents need not be aware of the actual path of the communications but can determine the resulting performance of each routing decision. This is achieved by testing the state of the feeder links at regular intervals. The problem is formulated as a Markov decision process (MDP), which models the feeder link selection process. There are two variants of reinforcement learning investigated for this task: Q-Learning and actor-critic (learning automata—LA).

Learning Automata (LA) is a decision-making agent working in a random environment that selects an optimal action from a set of actions based on their performance [17]. The environment reacts to the selected action either with a favorable response or with an unfavorable response. The output from the environment is feedback to the LA as an input to assist in deciding the next action. When an action is selected, it has a certain penalty probability c_i to produce an unfavorable response and a reward probability $d_i = 1 - c_i$ to produce a favorable response. The reward and penalty probabilities vary over time and depend on the environment. Each action is selected based on a probability distribution. If the action resulted in a favorable response, its selection probability will be increased, otherwise decreased.

An LA is described by a six-tuple, $\{\phi, X, \lambda, P, A, G\}$. Table 1 indicates the model parameters. The number of r actions is always less than or equal to the the number of internal states s.

Table 1. Entities and descriptions.

Entities	Description
Automaton	$\{\phi, \lambda, X, Pr, G, A\}$
Input	$X \in \{0, 1\}$
Internal state-ϕ	$\{\phi_1, \phi_2, \ldots, \phi_s\}$
Output-λ	$\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$
Pr	state probability vector at each instant t
Output function	$G : \phi \rightarrow \lambda$
algorithm **A**	computes $\phi(t+1)$ from $\phi(t)$

The diverse types of LA are classified based on the kind of input provided to the agent:

1. *P-model*: binary input (either 0 or 1),
2. *Q-model*: finite values in the interval $[0, 1]$,
3. *S-model*: continuous values in the interval $[0, 1]$.

The S-model has been applied to network communication problems due to its adaptive learning nature (see for example [18–20]) and is the model adopted in this work.

With Q-Learning, each action is linked to a certain reward (or Q-value). The agent's goal is to maximize the long-term reward associated with its selected actions: at state s, the agent receives an immediate reward by choosing an action a, based on a policy π. The agent's long-term reward is then given by the total discounted reward from that state s [21]. Q-Learning has been applied to different routing problems in the past (see for example [22,23]).

4. Feeder Selection with Reinforcement Learning

Central to the concept of RL is the notion of action rewards, which needs to be formulated for the problem context. Considering that N links are available to forward packets, the VNF emits periodic requests at regular intervals (dt seconds) to the switch to evaluate those links through a request/reply packet exchange, the time difference between which indicates the response time $\bar{d}_i^{dt}$ of each link i, $i = 1, 2, \ldots, N$ at the interval dt. The difference between the number of request departures and reply arrivals indicates the packet loss rates on the links. The packet loss rate, $pkt\_loss\_rate_i^{mt}$ is measured

on the links at an interval of mt seconds. Algorithms 1 and 2 indicate the proposed procedure for updating the delay and loss metrics by separate threads. The moving exponential average values of both metrics are combined to produce the expected link cost, $cost_i^{ft}$ at each flow modification interval ft by another thread as described in Equation (1).

$$cost_i^{ft} = k\, pkt\_loss\_rate_i^{ft} + (1 - pkt\_loss\_rate_i^{ft})\, d_i^{ft}\ ,\ \forall i = 1 \ldots N \tag{1}$$

where k is an arbitrary penalty constant, e.g., it may include the retransmission time [24]. At each interval ft, the average cost, $a\_cost_i^{ft}$ is updated via exponential smoothing with the hyperparameter α, $0 \leq \alpha \leq 1$ as given in Equation (2).

$$a\_cost_i^{ft} \leftarrow \alpha\, cost_i^{ft} + (1 - \alpha)\, a\_cost_i^{ft-1}\ ,\ \forall i = 1 \ldots N \tag{2}$$

This cost (i.e., a negative reward) is used to modify the routing policy of the feeder links as explained in the next sections.

Algorithm 1 Delay Computation

```
while true do
    for i = 1 to N do
        d_i^dt ← α d̄_i^dt + (1 − α) d_i^dt−1
    end for
    Pause for dt seconds
end while
```

Algorithm 2 Packet-loss Computation

```
while true do
    for i = 1 to N do
     pkt_loss_i^mt ← tx_pkts_i^mt − rx_pkts_i^mt
     avgpkt_loss_i^mt ← α pkt_loss_i^mt + (1 − α) avgpkt_loss_i^mt−1
     avgtx_pkts_i^mt ← α tx_pkts_i^mt + (1 − α) avgtx_pkts_i^mt−1
     pkt_loss_rate_i^mt ← avgpkt_loss_i^mt / avgtx_pkts_i^mt
    end for
    Pause for mt seconds
end while
```

4.1. Routing with SLA

At each flow modification interval, the expected costs (or negative rewards) of the most recently selected feeder link is compared to that of the other links to update the routing policy. As a result, the selection probability of the most recently selected link either increases or decreases. The value of the reward or penalty is calculated through parameter β^{ft} as defined by the Equation (3)

$$\beta^{ft} = \frac{a\_cost_v^{ft} - x_1^{ft}}{x_2^{ft} - x_1^{ft}}, \tag{3}$$

where

$a\_cost_v^{ft}$ is the most recent average cost obtained for feeder link v at time ft,
x_1^{ft} is the minimum average cost, and

x_2^{ft} is the maximum average cost.

Parameter β^{ft} represents the normalized reward or penalty awarded to the selected feeder link with respect to the observed performance of the other links. The routing policy is updated using [17]:

$$Pr[s_v]^{ft} \leftarrow Pr[s_v]^{ft-1} + \left(1 - \beta^{ft}\right) a \left(1 - Pr[s_v]^{ft-1}\right) - b\, \beta^{ft} Pr[s_v]^{ft-1} \tag{4}$$

where v is the index of the most recent link. The probability of the other links are updated as follows:

$$Pr[s_j]^{ft} \leftarrow Pr[s_j]^{ft-1} - (1 - \beta^{ft}) a\, Pr[s_j]^{ft-1} + b\, \beta \left(\frac{1}{N-1} - Pr[s_j]^{ft-1}\right) \quad ;\ j \neq v \tag{5}$$

where a, b are the reward and penalty parameters. With ergodic assumptions, both parameters are identical.

The feeder links are selected randomly according to the probability distribution Pr as described in Algorithm 3.

$Pr[s_i] = \frac{1}{N},\ i = 1, ...N,$ where N is Number of links $ft >> mt >> dt$

Algorithm 3 Latency and Packet-loss Optimized SLA Feeder Link Selection

Initialize:
$0 > a = b < 1;\ 0 < \alpha < 1, k = 100$
while true **do** ▷ Repeat every ft
$delays = [d_1, d_2, \ldots, d_N]$
$pkt\_loss\_rate = [pkt\_loss\_rate_1, pkt\_loss\_rate_2, \ldots, pkt\_loss\_rate_N]$
$g = 0$
for $i = 1$ to N **do** ▷ computing cost and its moving average
$cost_i \leftarrow k\ pkt\_loss\_rate_i + (1 - pkt\_loss\_rate_i)\ d_i$
$a\_cost_i \leftarrow \alpha\ cost_i + (1 - \alpha)\ a\_cost_i$
end for
▷ Probability reinforcement
$x_1 \leftarrow \min\{a\_cost_i\}\ \forall\, i = 1, 2, \ldots, N$
$x_2 \leftarrow \max\{a\_cost_i\}\ \forall\, i = 1, 2, \ldots, N$
$\beta \leftarrow \frac{a\_cost_v - x_1}{x_2 - x_1}$ where v is the index of selected link
for $i = 1$ to N **do**
$Pr[s_v] \leftarrow Pr[s_v] + (1 - \beta)\, a\, (1 - Pr[s_v]) - b\beta Pr[s_v]$
$Pr[s_i] \leftarrow Pr[s_i] - (1 - \beta) a Pr[s_i] + b\beta(\frac{1}{N-1} - Pr[s_i])\ \forall\, i \neq v,$
end for
▷ Cumulative Probability computation
for $i = 1$ to N **do**
$g = g + Pr[i]$
$cum\_Pr[i] = g$
end for
▷ Stochastic selection
for $i = 1$ to N **do**
if $random.random() < cum\_Pr[s_i]$ **then**
$bestindex \leftarrow i$
end if
end for
return $bestindex$
end while

4.2. Routing with Q-Learning

The agent updates the Q-values based on the combined cost computed by the measured packet-loss ratio and round trip time (RTT) delays as given in (2). At time ft, the agent observes the state of the environment s^{ft}, selects an action (feeder link) a^{ft} based on the policy and receives an immediate payoff $cost^{ft}$, observes the subsequent state y^{ft}, and adjusts its Q-values based on a learning rate α:

$$Q^{ft}(x,a) = \begin{cases} (1-\alpha)Q^{ft-1}(x,a) + \alpha(cost^{ft} + \gamma \min_{a} Q(y^{ft},b)), \text{where, } x = x^{ft}, a = a^{ft}. \\ Q^{ft-1}(x,a), \quad \text{otherwise.} \end{cases} \tag{6}$$

where the constant γ is a learning rate and α is an averaging constant. The feeder link selection uses an ϵ-*greedy* approach, where a feeder link is selected at random with probability of ϵ. Otherwise the feeder link with the lowest Q-value is selected, since the Q-values represent costs. By selecting the cost as the payoff, the agent tries to minimize both the average latency and packet-loss ratio at every state. The Q-Learning approach is described in Algorithm 4.

$ft >> mt >> dt$

Algorithm 4 Latency and Packet-loss Optimized Q-Learning Link Selection

Initialize:
    $0 < \alpha, \gamma, \epsilon < 1$
while true **do** ▷ Repeat every ft
    $delays = [d_1, d_2, \ldots, d_N]$, where N is Number of links
    $pkt\_loss\_rate = [pkt\_loss\_rate_1, pkt\_loss\_rate_2, \ldots, pkt\_loss\_rate_N]$
    for $i = 1$ to N **do**
        $cost_i \leftarrow k\ pkt\_loss\_rate_i + (1 - pkt\_loss\_rate_i)\ d_i$
    end for
    $minqval \leftarrow min(qvals)$
    for $i = 1$ to N **do**
        $qvals_i \leftarrow qvals_i + \alpha\ (cost_i + \gamma\ minqval - qvals_i)$
    end for
    if $random.random() < \epsilon$) **then**
        $bestindex \leftarrow random.randint(0, N-1)$
    else
        $bestindex \leftarrow$ argmin $\{qvals\}$
    end if
return $(bestind)$
end while

4.3. Routing Using Hierarchical SLA (HSLA)

We explored the use of hierarchical SLAs besides the above two methods as a means to improve the efficacy of the routing decisions. To illustrate, consider that four feeder links s_1, s_2, s_3, and s_4 are available to route packets. Two LAs, LA1 and LA2, located at level-1 of the hierarchy handle half of the links each, i.e., LA1 selects among s_1 and s_2 whereas, LA2 handles the other two links s_3, s_4. At level-2 of the hierarchy, LA3 selects the optimal link among the outcomes of LA1 and LA2. Figure 2 depicts the general concept behind hierarchical SLAs.

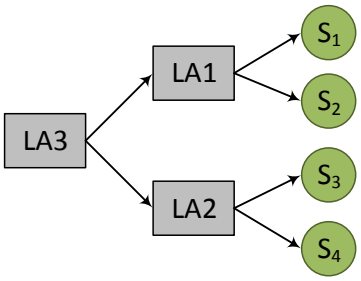


Figure 2. HSLA as a VNF controller.

5. Proof of Concept

A system prototype was developed and tested on an emulated satellite network in CloudLab [25]. The prototype uses Open vSwitch (OVS) to emulate the gateway handover with the intelligent agent-based routing methods implemented as a VNF. The implementation is based on the Ryu SDN framework [26]. For testing purposes, the user requests are modeled as REpresentational State Transfer (REST).

5.1. Flow Rules to Optimize Response Times

The reinforcement learning agent running on the controller optimizes the feeder link decisions by updating the flow rules on the OVS based on its adaptive policy. It makes decisions based on the monitored service delay and packet-loss ratios of the links, which are observations obtained by the monitor module.

5.1.1. Monitor Module

A separate thread from the Ryu controller sends specially crafted Internet Control Message Protocol (ICMP) echo packets at fixed intervals (dt seconds) through each of the feeder links using $OFPPacketOut$ messages. It then receives the ICMP replies through $OFPPacketIn$ messages, computes the delay, and stores them in a global data structure. The ICMP packet has a payload of size 44 bytes that contain the following information:

- switch ID,
- target IP address of a given server associated with the feeder link,
- creation time stamp.

The delay is computed as the time difference between the time at which an ICMP response is received from a server and the time stamp in the payload of the response. The method used for delay computation is described in Algorithm 1. The number of echo requests sent to and received from each server is updated in the respective global data structures. Likewise, another thread handles the estimation of the packet losses based on the number of echo requests sent and received at a certain rate (every mt seconds) and computes the packet-loss ratio using Algorithm 2.

5.1.2. TCP Packet Handling

The handover is implemented by defining a replicated service that is handled by back servers connected to the gateways. Clients send requests addressed to a Virtual IP address (VIP), which are forwarded to servers according to a selected policy by the VNF controller. The IP addresses of the back servers are hidden from the outside world. In the beginning, the SDN controller receives the requests destined for the VIP and adds a flow rule using a tuple of five match fields to one of the back servers, usually to the server at index 0. While adding a flow rule, it sets the destination IP and destination MAC address fields of the packets to the IP address and MAC address of the chosen back server and set the out_port ($OFPActionOutput$) to the port, the back server is connected to the switch. It also adds

a flow rule to forward the TCP packets in the reverse direction to send back the TCP responses to the clients. In the reverse flow rule, it sets source IP address, source MAC address fields to VIP and Virtual MAC (VMac) addresses and sets out_port ($OFPActionOutput$) to the port, the client is connected to the switch.

5.1.3. Dynamic Flow Modification

A server is selected at the beginning of each flow modification interval (ft seconds) based on the chosen algorithm by a separate thread. The value of ft is chosen to be greater than the values of the intervals dt and mt. In addition to Q-Learning and the SLA algorithms, results obtained with Round-Robin (RR) have been included as reference performance. The agent performs the following steps to modify the flow entries of the switch at the beginning of each flow modification interval:

1. Selects a server based on a policy.
   - SLA—the agent selects a server based on the performance of the server from the previous flow modification interval as in Algorithm 3.
   - Q-Learning—the agent selects a server that has the minimum Q-value as in Algorithm 4.
   - RR—the destination server is selected in sequential order.

   In the former two methods (i.e., learning methods), the forward flow rule of the switch is not modified if the selected server in an interval ft is the same in the previous interval $ft - 1$.
2. Looks up the flow table for a match with the VIP as the destination IP address and the client IP address as the source IP address. If it finds a matching entry, it modifies the flow rule using the OFPFC_MODIFY_STRICT command with actions to set the destination IP address and destination MAC address fields to that of the selected server, and the out_port to the port, the server is connected to the switch.
3. It adds a new flow rule to handle the reverse traffic from the newly selected server if there is no matching rule exists in the flow table.

5.2. Testbed Setup

The experimental network built from Clemson's cluster on the CloudLab testbed is depicted in Figure 3. The topology consists of four back servers, one Open vSwitch (OVS), a client (emulating the aggregated traffic flow of multiple users), and the learning agent-based SDN controller as a VNF. The client is connected to the back server via the OVS. The learning agent decides the forwarding of HTTP requests from the clients to the servers based on their observed performance as measured by the response times in ICMP echo/reply packet exchange and the packet-loss ratios on the links connecting OVS to the back servers.

The back servers s_1, s_2, s_3, and s_4 run on Xen Virtual Machines (VM), each configured with two CPU cores—Intel Xeon 2.0 GHz. Apache-2.4.18 was installed on the back servers to provide the simulated network service that users need to access to either retrieve or send information. Open vSwitch runs from the machine *switch* and forwards the HTTP requests and responses between the client and back servers. The machine has 56 processors, each having 14 cores, and its model is Intel Xeon 2.40 GHz. The SDN controller is running from the Xen VM *controller*, which has four CPU cores. Artificial delays and packet losses were introduced to the links connecting back servers to the switch using Linux's NetEm tool (Traffic Control) to emulate propagation latencies and the impact of atmospheric channel impairments. A Poisson traffic generator was developed with various configurable sending rates (λ) to emulate aggregated traffic from clients. The client in the Figure 3 emulates the users in Figure 1, the switch represents a router on the satellite, and the back servers represents the links connecting to the ground stations.

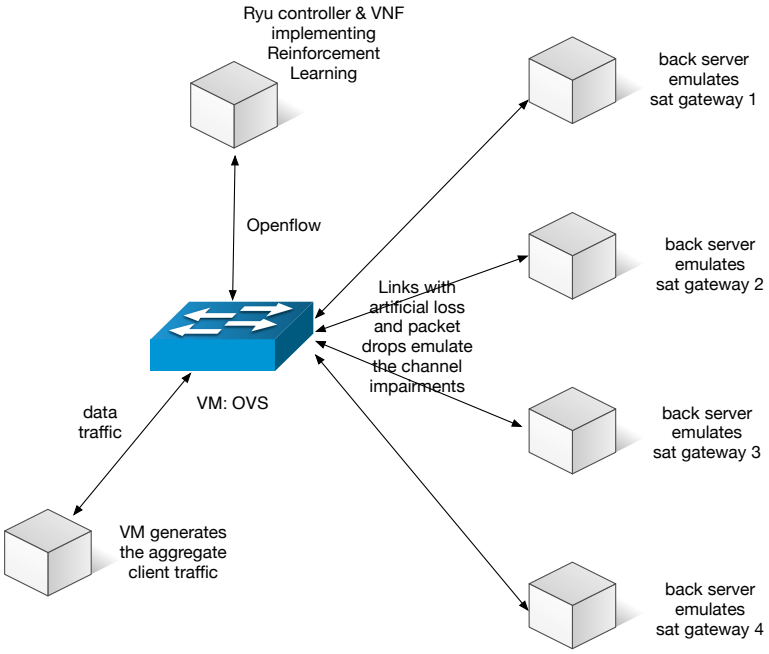


Figure 3. Emulated HTS system with VNF providing reinforcement learning-based routing.

6. Results

Our experiments were conducted with a satellite-like environment by creating emulated delays and packet-losses between clients and servers. The experiments were conducted with different assumptions for the packet-loss of the links emulating different weather conditions. A delay of 8 ms was configured on the links connecting servers to the OVS, besides a 300 ms delay imposed by the CloudLab while provisioning the resources. Packet-loss ratios used in our test scenarios are listed in Tables 2–8. The performance of each algorithm was measured by considering HTTP GET request transmissions for static files of varying sizes between 100 KB to 1 MB and sending HTTP POST requests with a data payload of 10 KB from the client machine. Each experiment was run for 10 min and repeated six times to get statistical averages.

6.1. Scenario-1

In test Scenario-1, the links connecting servers to the switch are configured with packet-loss percentages as given in Table 2. These packet losses are emulated using Linux's Traffic Control which can drop packets at a selected rate before they can reach the IP network stack. In this scenario, server s_3 is the fastest one. However, if too many requests are sent to this server, it will increase the response times and making other links appropriate, i.e., server s_1.

Table 2. Emulated packet-loss percentages for Scenario-1.

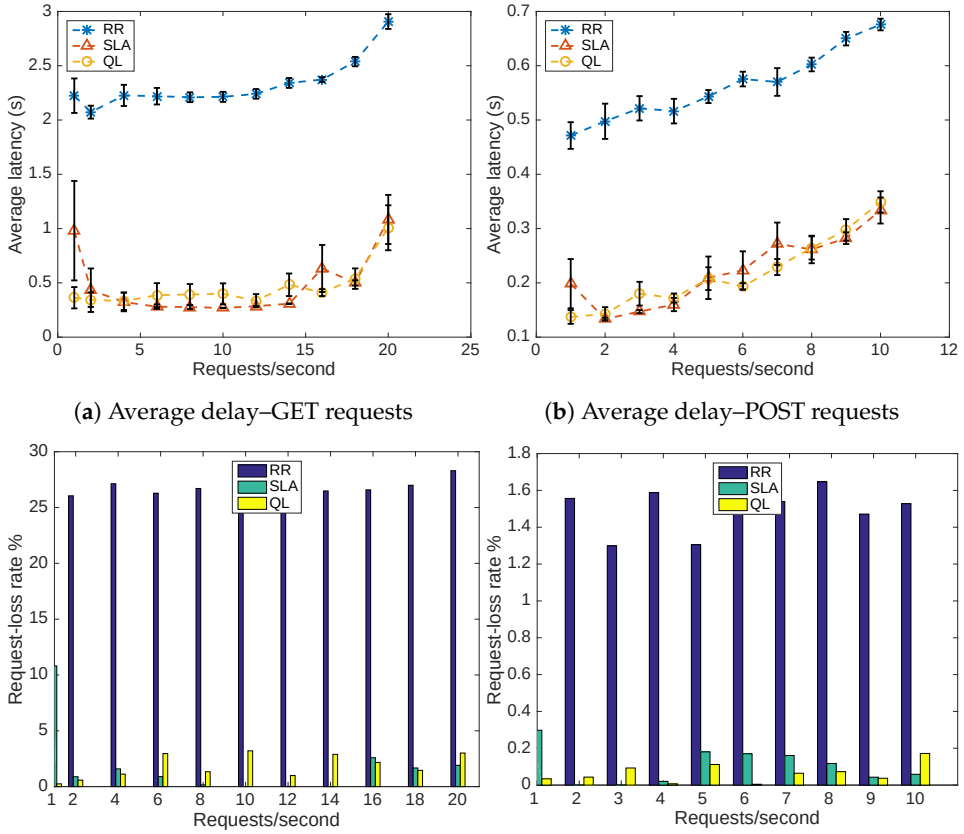
s1	s2	s3	s4
2	20	1	40

Figure 4a,b depict the performance of the forwarding algorithms while handling HTTP GET and POST requests sent from the client to the back servers. The average delay (i.e., the response time) in serving files is reported. Average delays with both learning algorithms are ≈80% lower than those achieved with RR with GET requests and ≈60% lower with POST requests.

The requests-loss ratios are depicted in Figure 4c,d. The request-loss is the ratio of requests not serviced by the system to the total number of requests sent by the client/(s). The request loss ratio is ≈90% better with both learning algorithms compared to RR with both GET and POST requests.

Both SLA and Q-Learning algorithms show better average response time and request-loss ratios than RR. The variations observed between SLA and Q-Learning is due to the stochastic nature of the server selection.

Tables 3 and 4 show the number of times each server is selected by the algorithms while serving GET and POST requests in Scenario-1. The learning algorithms selected the non-optimal servers s_2 and s_4 a lower number of times than RR and utilized the server s_3 more frequently, whereas the link usage was split equally with RR as expected.



(**a**) Average delay–GET requests (**b**) Average delay–POST requests

(**c**) Request-loss ratios–GET requests (**d**) Request-loss ratios–POST requests

Figure 4. Routing performance (Scenario-1).

Table 3. Server (feeder link) selection count for Scenario-1 (GET).

Algorithm	s1	s2	s3	s4
RR	440	440	439	439
QL	266	22	1143	22
SLA	157	31	1147	22

Table 4. Server (feeder link) selection count for Scenario-1 (POST).

Algorithm	s1	s2	s3	s4
RR	325	325	325	324
QL	384	14	881	15
SLA	144	38	1093	21

6.2. Scenario-2

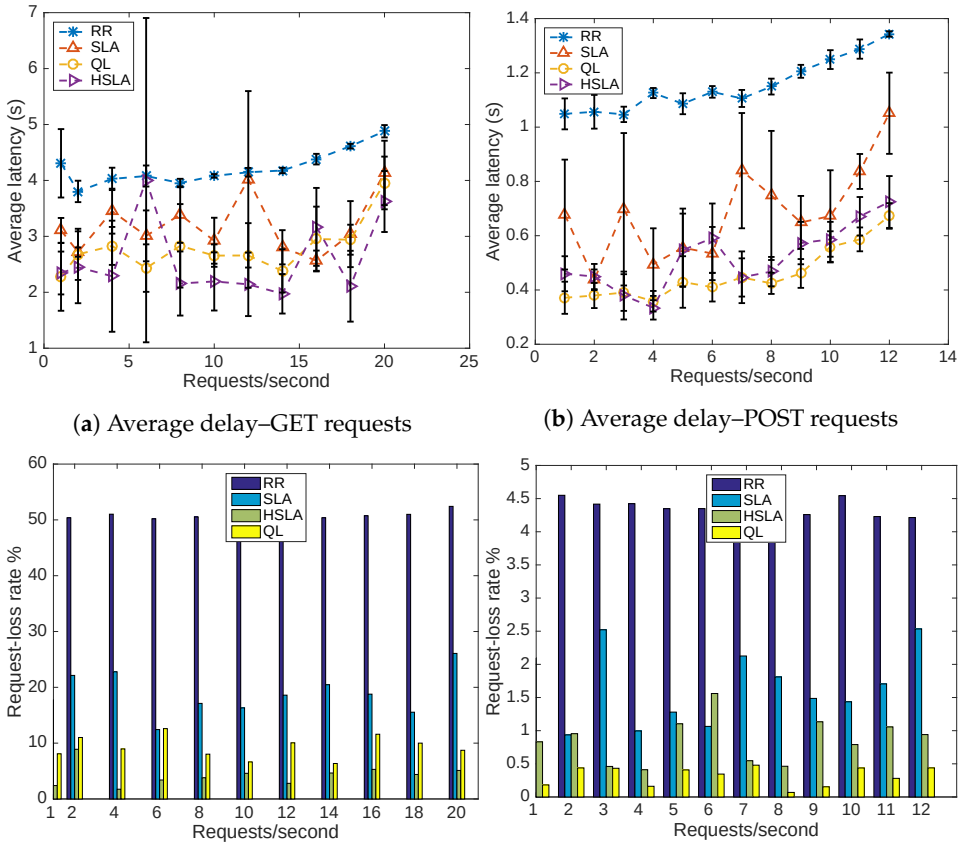
In the second scenario, a dynamic environment is considered. To this end, the packet-loss percentages of the links are changed every 5 min as given in Table 5. The servers s_1 and s_3 are assigned the lowest (emulated) packet-loss percentages.

Table 5. Emulated packet-loss percentages for Scenario-2.

Packet-Loss %	**s1**	**s2**	**s3**	**s4**
First 5 **min**	20	40	1	40
second 5 **min**	1	50	20	50

Figure 5a,b show an increase in the average delays with SLA as a function of the load λ compared to Q-Learning since the SLA algorithm selects servers using a probability distribution. It takes a certain amount of time for the SLA to adjust the policy and find the best link after a change in the packet-loss configuration. However, the Q-Learning performed better than both RR and SLA. With HSLA, the learning occurred faster as each agent was limited to decide between two choices, which translates into improved performance for both the average delay and the request loss ratios in serving HTTP GET requests. With GET requests, the average delays of HSLA, Q-Learning, and SLA were ≈20%–50%, ≈20%–40%, and ≈15%–35% better than RR. HSLA was ≈10%–40% better than Q-learning, and Q-Learning was ≈2%–20% better than SLA. With POST requests, the average delays of HSLA and Q-Learning were ≈60% better than RR, whereas SLA was ≈50% better than RR. Q-learning was ≈5%–20% better than HSLA, and ≈20%–40% better than SLA.

Figure 5c,d show that the request loss ratio percentages were better with all three learning algorithms than with RR. The request-loss ratios of HSLA, Q-Learning, and SLA were ≈ 80%, ≈90%, and ≈60% better than RR, and Q-Learning was ≈75% and ≈55% better than SLA and HSLA when serving HTTP GET requests. The request-loss ratios of HSLA, Q-Learning, SLA were ≈80%, ≈90%, and ≈40%–75% better than RR, and Q-Learning was ≈ 50%–80% and ≈10%–80% better than SLA and HSLA when serving POST requests.



(**a**) Average delay–GET requests

(**b**) Average delay–POST requests

(**c**) Request-loss ratio–GET requests

(**d**) Request-loss ratio–POST requests

Figure 5. Performance measurements—Scenario-2.

Tables 6 and 7 show the number of times each server is selected by the algorithms while serving GET and POST requests in Scenario-2. All of the learning algorithms selected the non-optimal servers s_2 and s_4 a lower number of times. Q-Learning and HSLA used them a lower number of times than SLA, whereas RR used all of the servers equally. The learning algorithms chose the servers s_1 and s_3 almost an equal number of times since the quality of the links connecting them alternates between good and worse every 5 min.

Table 6. Server (feeder link) selection count for Scenario-1 (GET).

Algorithm	s1	s2	s3	s4
RR	515	515	515	514
QL	518	16	841	18
SLA	598	171	666	176
HSLA	581	49	757	59

Table 7. Server (feeder link) selection count for Scenario-2 (POST).

Algorithm	s1	s2	s3	s4
RR	416	415	413	413
QL	776	24	727	24
SLA	526	159	749	148
HSLA	687	68	727	68

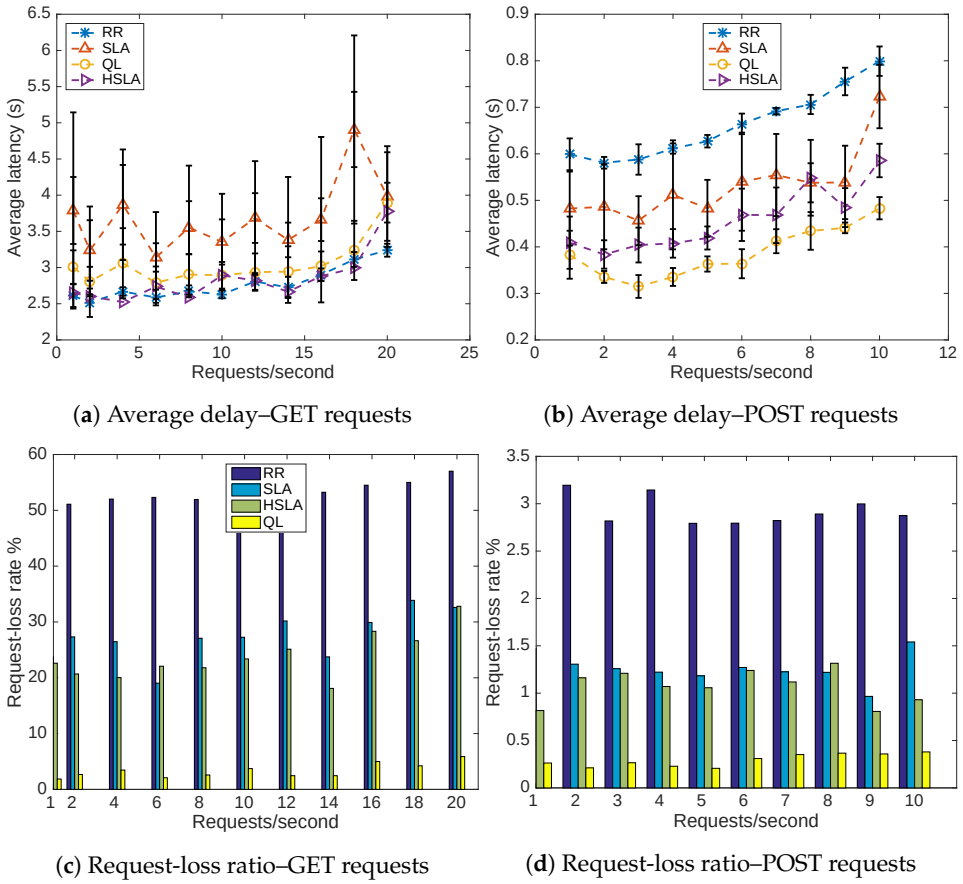
6.3. Scenario-3

In Scenario-3, the efficiency of the learning algorithms was tested in a dynamic environment, where packet-loss changes occur at a higher rate, i.e., every 2 min. Packet-loss percentages on the links connecting servers to the client are given in Table 8. Servers s_1 and s_3 were configured with the lowest packet-losses. Figure 6a shows that the average delay obtained with SLA was higher than the other three algorithms in serving GET requests, and except SLA, all of them showed equivalent performances. Figure 6b indicates that Q-Learning produced the lowest average delay when serving POST requests. With GET requests, the average delay of RR was ≈15% and ≈25%–40% better than Q-Learning and SLA. The average delay with HSLA was similar to RR, whereas Q-learning was ≈20% better than SLA, and HSLA was ≈10% better than Q-Learning. When sending POST requests, average delays of HSLA, Q-Learning, and SLA were ≈30%, ≈40%, and ≈20% better than RR, and Q-learning was ≈20% better than HSLA and ≈30% better than SLA.

Figures 6c,d depict the request-loss ratios obtained with the algorithms. All learning algorithms show better performances, and Q-Learning shows the best performance. When sending GET requests, the request-loss ratios of HSLA, Q-Learning, and SLA were ≈60%, ≈90%, and ≈50% better than RR, and Q-Learning was ≈90% better than both SLA and HSLA. When sending POST requests, the request-loss ratios of HSLA, Q-Learning, and SLA were ≈60%, ≈90%, and ≈60% better than RR. Q-Learning was ≈75% better than both SLA and HSLA.

Table 8. Emulated packet-loss percentages for Scenario-3.

Packet-Loss %	s1	s2	s3	s4
First 2 min	20	30	1	30
second 2 min	1	40	20	40



(**a**) Average delay–GET requests (**b**) Average delay–POST requests

(**c**) Request-loss ratio–GET requests (**d**) Request-loss ratio–POST requests

Figure 6. Performance measurements—Scenario-3.

Tables 9 and 10 show the number of times each server was selected by the algorithms in Scenario-3 while serving GET and POST requests. All of the learning algorithms chose the non-optimal servers s_2 and s_4 for the least number of times. Among all, Q-Learning was the lowest, whereas RR used all the servers equally. Since LA algorithms involve updating selection probabilities and making stochastic server selection, learning was not efficient when compared to Q-Learning. When the quality of the links changes more rapidly, the LA algorithms find it harder to choose the optimal link due to the learning delay.

Table 9. Server (feeder link) selection count for Scenario-3 (GET).

Algorithm	s1	s2	s3	s4
RR	898	895	895	894
QL	1668	26	1021	34
SLA	1185	412	1119	435
HSLA	994	326	1412	346

Table 10. Server (feeder link) selection count for Scenario-3 (POST).

Algorithm	s1	s2	s3	s4
RR	654	654	654	654
QL	1330	40	1175	44
SLA	971	338	944	366
HSLA	1014	221	1189	195

7. Discussion

All the learning algorithms showed exceptionally better performance in the static environment (Scenario-1). The reason being is that limited link switching was needed, since s_3 was the only fastest option. However, with a dynamic environment, where the fastest option alternates between s_1 and s_3, the link switching occurs at a higher rate, which is a desirable feature for HTS to be able to adapt to weather condition changes. A drawback of this mechanism is linked to an implementation issue rather than the technique itself because the dynamic flow modifications may produce packet drops when the flow rules are being modified in the switch. However, despite the additional delay involved in TCP retransmissions, the average response times were observed to improve. The performance penalty caused by the dynamic modification of flows in OpenFlow switches is a known issue [27–29]. It was experimentally verified that OpenFlow SDN hardware switches would cause 3 ms–30 ms latency due to flow rule modifications [27]. Updating of the selection probabilities with the SLA algorithm requires a certain amount of time to find the best link when the environmental conditions change. There are possibilities for non-optimal servers to be selected during the transient period due to the delay in learning. If the frequency of the changes in the environment increase, the switching between servers is also expected to increase, causing additional packet losses at the switch ports. With HSLA, the learning happens faster as the selection probabilities are adjusted in a shorter time since only two actions are available at each SLA. Another possible way to reduce these issues is through priority-based flow [30], which will be investigated in the future.

8. Conclusions

With the experimental results obtained with an emulated multi-site satellite network, it was shown that an SDN/VNF approach could dynamically switch transmissions among the feeder links of an HTS system to alleviate the performance degradation brought by the adverse atmospheric conditions, which can temporarily affect one or more of the gateway sites and cause network congestion. The use of reinforcement learning-based algorithms was explored for this task to reduce both packet losses and the average latency. Since reinforcement learning does not need prior training, the proposed mechanisms help to improve the autonomy of HTS systems.

Author Contributions: Conceptualization, R.L. and G.V.; methodology, R.L. and G.V.; software, R.L. and G.V.; validation, G.V.; formal analysis, R.L. and G.V.; investigation, R.L. and G.V.; resources, R.L. and G.V.; data curation, R.L. and G.V.; writing—original draft preparation, R.L. and G.V.; writing—review and editing, R.L. and G.V.; visualization, R.L. and G.V.; supervision, R.L.; project administration, R.L.; funding acquisition, R.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Aeronautics and Space Administration (NASA) grant #80NSSC17K0525.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SDN	Software-Defined Networking
VNF	Virtual Network Function
NFV	Network Function Virtualization
HTS	High Throughput System
EHF	Extremely High Frequency
ACM	Adaptive Coding Modulation
SLA	S-model Learning Automaton
RL	Reinforcement Learning
HSLA	Hierarchical S-model Learning Automaton
QL	Q-Learning algorithm
ICMP	Internet Control Message Protocol

References

1. Hwang, J.; Ramakrishnan, K.K.; Wood, T. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. *IEEE Trans. Netw. Serv. Manag.* **2015**, *12*, 34–47. [CrossRef]
2. Malik, A.; Singh, P. Free Space Optics: Current Applications and Future Challenges. *Int. J. Opt.* **2015**, *2015*, 945483. [CrossRef]
3. Kyrgiazos, A.; Evans, B.; Thompson, P.; Jeannin, N. Gateway diversity scheme for a future broadband satellite system. In Proceedings of the 2012 6th Advanced Satellite Multimedia Systems Conference (ASMS) and 12th Signal Processing for Space Communications Workshop (SPSC), Vigo, Spain, 5–7 September 2012; pp. 363–370.
4. de Cola, T.; Ginesi, A.; Giambene, G.; Polyzos, G.C.; Siris, V.A.; Fotiou, N.; Thomas, Y. Network and Protocol Architectures for Future Satellite Systems. *Found. Trends Netw.* **2017**, *12*, 1–161. [CrossRef]
5. Jeannin, N.; Castanet, L.; Radzik, J.; Bousquet, M.; Evans, B.; Thompson, P. Smart gateways for terabit/s satellite. *Int. J. Satell. Commun. Netw.* **2014**, *32*, 93–106. [CrossRef]
6. Aurizzi, M.M.; Milana, S.; Rossi, T.; Cianca, E.; Ruggieri, M. SDN for Smart Gateway Diversity Optimization in High Throughput Satellite Systems. In Proceedings of the 2019 IEEE Aerospace Conference, Big Sky, MT, USA, 2–9 March 2019; pp. 1–5.
7. Bonfim, M.S.; Dias, K.L.; Fernandes, S.F. Integrated NFV/SDN architectures: A systematic literature review. *ACM Comput. Surv. (CSUR)* **2019**, *51*, 1–39. [CrossRef]
8. Ejaz, S.; Iqbal, Z.; Shah, P.A.; Bukhari, B.H.; Ali, A.; Aadil, F. Traffic load balancing using software defined networking (SDN) controller as virtualized network function. *IEEE Access* **2019**, *7*, 46646–46658. [CrossRef]
9. Akhtar, N.; Matta, I.; Wang, Y. Managing NFV using SDN and control theory. In Proceedings of the NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, Turkey, 25–29 April 2016; pp. 1113–1118.
10. Xu, J.; Wang, J.; Qi, Q.; Sun, H.; He, B. Deep neural networks for application awareness in SDN-based network. In Proceedings of the 2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP), Aalborg, Denmark, 17–20 September 2018; pp. 1–6.
11. Van Rossem, S.; Tavernier, W.; Sonkoly, B.; Colle, D.; Czentye, J.; Pickavet, M.; Demeester, P. Deploying elastic routing capability in an SDN/NFV-enabled environment. In Proceedings of the 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, USA, 18–21 November 2015; pp. 22–24.
12. Bertaux, L.; Medjiah, S.; Berthou, P.; Abdellatif, S.; Hakiri, A.; Gelard, P.; Planchou, F.; Bruyere, M. Software defined networking and virtualization for broadband satellite networks. *IEEE Commun. Mag.* **2015**, *53*, 54–60. [CrossRef]
13. Gardikis, G.; Costicoglou, S.; Koumaras, H.; Sakkas, C.; Kourtis, A.; Arnal, F.; Contreras, L.M.; Gutierrez, P.A.; Guta, M. NFV applicability and use cases in satellite networks. In Proceedings of the 2016 European Conference on Networks and Communications (EuCNC), Athens, Greece, 27–30 June 2016; pp. 47–51.
14. Li, T.; Zhou, H.; Luo, H.; Xu, Q.; Ye, Y. Using SDN and NFV to Implement Satellite Communication Networks. In Proceedings of the 2016 International Conference on Networking and Network Applications (NaNA), Hakodate, Japan, 23–25 July 2016; pp. 131–134.
15. Akyildiz, I.F.; Kak, A. The Internet of Space Things/CubeSats: A ubiquitous cyber-physical system for the connected world. *Comput. Netw.* **2019**, *150*, 134–149. [CrossRef]
16. Cai, Y.; Wang, Y.; Zhong, X.; Li, W.; Qiu, X.; Guo, S. An approach to deploy service function chains in satellite networks. In Proceedings of the NOMS 2018—2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018; pp. 1–7.
17. Narendra, K.S.; Thathachar, M.A. *Learning Automata: An Introduction*; Dover Publications Inc: Mineola, NY, USA, 2012.
18. Saleem, A.; Afzal, M.K.; Ateeq, M.; Kim, S.W.; Zikria, Y.B. Intelligent learning automata-based objective function in RPL for IoT. *Sustain. Cities Soc.* **2020**, *59*, 102234. [CrossRef]
19. Nicopolitidis, P.; Papadimitriou, G.I.; Pomportsis, A.S. Using learning automata for adaptive push-based data broadcasting in asymmetric wireless environments. *IEEE Trans. Veh. Technol.* **2002**, *51*, 1652–1660. [CrossRef]

20. Tanwar, S.; Tyagi, S.; Kumar, N.; Obaidat, M.S. LA-MHR: learning automata based multilevel heterogeneous routing for opportunistic shared spectrum access to enhance lifetime of WSN. *IEEE Syst. J.* **2018**, *13*, 313–323. [CrossRef]
21. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
22. Dudukovich, R.; Hylton, A.; Papachristou, C. A machine learning concept for DTN routing. In Proceedings of the 2017 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE), Montreal, QC, Canada, 10–12 October 2017; pp. 110–115.
23. Boyan, J.A.; Littman, M.L. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems*; Morgan Kaufmann: Burlington, MA, USA, 1993; pp. 671–678.
24. Gelenbe, E.; Gellman, M.; Lent, R.; Liu, P.; Su, P. Autonomous smart routing for network QoS. In Proceedings of the First International Conference on Autonomic Computing, New York, NY, USA, 17–19 May 2004; pp. 232–239.
25. Duplyakin, D.; Ricci, R.; Maricq, A.; Wong, G.; Duerig, J.; Eide, E.; Stoller, L.; Hibler, M.; Johnson, D.; Webb, K.; et al. The Design and Operation of CloudLab. In Proceedings of the Proceedings of the USENIX Annual Technical Conference (ATC), Renton, WA, USA, 10–12 July 2019; pp. 1–14.
26. Nippon Telegraph and Telephone Corporation. Ryu: Component-Based Software Defined Networking Framework. Available online: https://ryu-sdn.org (accessed on 1 November 2020).
27. He, K.; Khalid, J.; Gember-Jacobson, A.; Das, S.; Prakash, C.; Akella, A.; Li, L.E.; Thottan, M. Measuring control plane latency in SDN-enabled switches. In Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, Santa Clara, CA, USA, 17–18 June 2015; pp. 1–6.
28. Rotsos, C.; Sarrar, N.; Uhlig, S.; Sherwood, R.; Moore, A.W. OFLOPS: An open framework for OpenFlow switch evaluation. In Proceedings of the International Conference on Passive and Active Network Measurement, Vienna, Austria, 12–14 March 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 85–95.
29. Kuźniar, M.; Perešíni, P.; Kostić, D.; Canini, M. Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches. *Comput. Netw.* **2018**, *136*, 22–36. [CrossRef]
30. Oh, B.H.; Vural, S.; Wang, N.; Tafazolli, R. Priority-based flow control for dynamic and reliable flow management in SDN. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1720–1732. [CrossRef]