SDN-enabled Adaptive and Reliable Communication in IoT-Fog Environment Using Machine Learning and Multi-Objective Optimization

Aamir Akbar, Muhammad Ibrar, Mian Ahmad Jan*, Ali Kashif Bashir, Lei Wang**

***Abstract*—The Internet of Things (IoT) has inspired the development of emerging new applications. Backed by the resourceful fog computing, the IoT devices are capable to meet the demands of tasks, even the most computationally-intensive ones. However, many existing IoT applications are unable to perform well, while communicating with the fog server due to different QoS requirements. Constantly changing traffic demands of applications is another challenge to consider. The demand for real-time applications includes communicating over a path that is less prone to delay. Moreover, applications that offload computationally-intensive tasks to the fog server need a reliable path that has lower probability of link failure. This results in a trade-off among continuously changing objectives, i.e., minimizing the end-to-end delay and maximizing the reliability of paths between IoT devices and the fog server. To tackle this problem, we propose a novel approach that considers an SDN-enabled multi-hop scenario for the IoT-Fog environment. We evaluated the reliability level of links by employing the real-life network statistics recorded for a period of over 5 months. We use a multi-objective optimization (MOO) algorithm to search the Pareto-optimal paths by considering the two conflicting objectives. Our experimental evaluation considers two different applications - a real-time application (App-1) using UDP sockets and a task offloading application (App-2) using TCP sockets. Our results show that: (i) using MOO, the trade-off between the two objectives can be optimized, and (ii) the SDN controller was able to make adaptive decision on-the-fly to choose the best path from the Pareto-optimal set. The App-1, communicating over the selected path finished its execution in 13% less time than communicating over the shortest path. The App-2 had 41% less packet loss using the selected path compared to using the shortest path.**

***Index Terms*—IoT, Multi-Objective Optimization (MOO), Machine Learning, Fog Computing, Software Defined Networks (SDN).**

I. Introduction

THE rapid use of gadgets linked with the Internet fosters the influence in different juncture of life. To connect the physical world with the digital world, the Internet of Things (IoT) [1] plays an important role. Numerous IoT applications are running in practice such as smart cars, smart industry, smart cities [2], and smart health [3], [4], [5]. The overall growth of IoT in terms of global market share in 2020 was estimated to reach $250B [6]. The application layer of IoT is comprised of custom applications that continuously generate data streams for real-time processing and analysis. However, at the things layer, the resources available are limited in terms of computation and storage. To make IoT applications virtually unbounded in terms of storage and processing power, the integration of IoT devices with cloud computing has played a vibrant role [7]. Even though cloud computing supports the execution of computationally-intensive tasks, it lacks to provide reliable communication. Therefore, techniques that involve machine learning can be used to find a reliable path and provide reliable communication for IoT devices [8], [9]. In particular, a reliable communication channel is required for delay-sensitive applications that need a quick response with extremely low latency, e.g. smart cities and smart health.

Fog computing [10] provides computing services between the IoT devices and cloud data centres and is typically located, though not exclusively, at the edge of the network. As the promises of both cloud and fog computing are similar, the latter aims to provide low latency with a wider spread and geographically distributed nodes. Although fog computing minimizes the latency and reduces the amount of data sent to the cloud, the deployment of fog nodes, i.e., fog server in ubiquitous computing incur higher cost [4]. Alternatively, IoT devices are allowed to connect to fog servers in a multi-hop manner [11]. However, in such settings, the desired performance of applications for IoT devices is hard to achieve. For example, finding a reliable path in a multi-hop IoT-Fog that has a minimum end-to-end delay.

In a multi-hop IoT-Fog scenario, the physically distributed but logically unifying fog servers can be achieved using SDN-enabled IoT-Fog [12], [4], [11]. The core idea of SDN [13] is to decouple the control plane from the data plane by delegating the former from all network devices, i.e., routers, switches, and Access Points (APs), to a centralized controller [14]. However, in SDN-enabled multi-hop IoT-Fog environment, link failures events can occur, which could result in recomputing of an alternative path for a flow. The alternative path may not be the best in terms of reliability or delay. In [15], [16], the authors have reported that a link failure in a network occurs almost every 30 minutes. The causes of failure may be due to Windows maintenance such as operating system reboot, network connection (OSPF convergence), software

Aamir Akbar and Mian Ahmad Jan are with the Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan. (email: amirakbar@awkum.edu.pk; mianjan@awkum.edu.pk)

Muhammad Ibrar and Lei Wang are with the School of Software, Dalian University of Technology, China. (email: mibrar@mail.dlut.edu.cn; lei.wang@dlut.edu.cn)

Ali Kashif Bashir is with the School of Computing and Mathematics, Manchester Metropolitan University, United Kingdom. (email: dr.alikashif.b@ieee.org)

$*$ indicates the corresponding author

$**$ indicates the co-corresponding author

(BIOS upgrade), hardware (replacement of a device or line card), and configuration (primary-back fail-over and VPN tunnelling) [17]. Therefore, an IoT application is adversely affected if the flow from the application to the fog server is dropped or delayed due to a link or a device failure.

The problem faced by IoT applications such as communication over a reliable path that has minimum end-to-end delay can be resolved. In this paper, we propose a machine learning-based approach to find the reliability of links in a multi-hop and SDN-enabled IoT-Fog settings. We define the reliability of a link as the average failure/downtime or repair-time in the specific time duration. More specifically, the frequency of downtime/repair-time shows the reliability level of a link. We use a regression-based k-nearest neighbor (k-NNR) machine learning algorithm [18], which is trained on a dataset of a real-life network captured for over a period of 5 months, to estimate the links reliability. The k-NNR considers the link parameters such as *link utilization rate*, *link failure*, and the *frequency of failures* from the dataset. Also, we measure the link delay as the sum of transmission delay (packet size/link bandwidth), processing delay and propagation delay. Furthermore, different applications have different Quality of Service (QoS) requirements. For example, a real-time application may require a path that has a minimum end-to-end delay, while compromising the objective of maximum path reliability. Therefore, we consider finding the path from the source node to the fog server, in multi-hop IoT-Fog, as a multi-objective optimization (MOO) problem.

In a MOO, several conflicting objectives are optimized simultaneously. A trade-off between objectives naturally exists that results in creating a set of Pareto-optimal solutions [19]. In this work, we consider two objectives: 1) maximize the path reliability: we find it as the minimum link reliability in all links making a path, and 2) minimize the path delay: we measure it as the average delay of links making a path.

Achieving two or more objectives at the same time may not be possible. For example, maximizing the path reliability may prevent the objective of minimizing path delay because a path having maximum reliability may happen to have one or more links with high transmission delay. Therefore, using a MOO algorithm, we will obtain an optimal path or a set of paths that optimize the trade-off between the two objectives. Furthermore, using the Pareto-optimal paths, the SDN controller will decide on-the-fly which path to select for a flow based on the packet types. Therefore, an adaptive and reliable communication for different types of applications in an IoT-Fog paradigm can be achieved. We highlight the major contributions of this paper as follow.

1) When a link failure happens due to external and uncontrollable factors, our proposed approach ensures that it is still capable to estimate the link reliability level using machine learning algorithm.
2) Our proposed approach optimizes the trade-off between two objectives: 1) maximize the path reliability, and 2) minimize the path end-to-end delay. This leads to finding the Pareto-optimal approximation of paths in the SDN controller.
3) Our proposed approach improves the performance of IoT applications in terms of minimizing the end-to-end delays and packet losses for a set of source nodes and destination nodes. This is achieved by using an adaptive switching mechanism employed in an SDN controller. In this case, the source nodes are SDN switches to which the IoT devices are linked, while the destination nodes are the fog servers.

The rest of this paper is organized as follow. Section II discusses the related work. Section III explains the system model followed by reliable communication in SDN-enabled IoT-Fog networks in Section IV. Section V presents the adaptive communication in SDN-enabled IoT-Fog networks. Section VI discusses the experimental setup and results. Finally, Section VII concludes the paper with future directions.

II. Related Work

The communication over highly reliable paths, according to QoS requirements in a network, is a hot research topic. In [20], the authors proposed a time slot-based approach in which the SDN controller regularly find multi-paths for the transfer of media-intensive applications over the Internet while meeting their QoS requirements. They used heuristics to find optimal survivable multi-paths with least cost. However, the QoS cannot always reflect the actual service quality experienced by the end-users. Authors in [21] focused more on quality of experience (QoE) that influence the interactivity of the services experienced by end-users.

To provide a better QoE, multi-objective optimization (MOO) techniques have been used, in different domains, to minimize the trade-off between conflicting objectives. For example, MOO has been used to improve coverage of wireless nodes [22] in a network, to optimize the trade-off between two conflicting objectives (battery power consumption and network usage) of mobile-cloud hybrid applications for smartphones [23] and battery-powered robots [24]. In MOO algorithms, multiple objectives are treated simultaneously, subject to a set of constraints [25]. Furthermore, MOO problems are more often solved by using multi-objective evolutionary algorithms (MOEAs) [26], which aim for searching the Pareto-optimal set of solutions in a single run. One particular MOEA algorithm, the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [27], has been widely researched in the family of MOO algorithms, due to its capability of efficiently constructing an approximate Pareto-front of non-dominated solutions in a set of solutions.

However, for different types of end-user applications, a reliable path selection based on dynamic nature of the networks such as link utilization rate, link failures and the frequency of failures must be considered [28]. Moreover, the end-to-end delay of a given path is an important criterion for a reliable path selection. The authors in [29] have observed that for $30 - 80\%$ of failures in the network, there is an alternative path with better characteristics in terms of delay and packet loss. Therefore, machine learning-based prediction algorithms can be used to provide better QoE under various network conditions. For example, ML-based approaches have

been used to provide better VoIP service under various network conditions [30], to classify YouTube QoE based on encrypted network traffic [31], and in other areas such as predicting SDN-based networks QoE [32].

Furthermore, to minimize the end-to-end delay in resource-constrained IoT devices, from the centralized virtual cloud, mobile computing has shifted towards mobile edge computing (MEC) [33] and fog computing [10]. As the long-distance in terms of network topology between a mobile device and the conventional cloud remains a significant drawback for mobile-cloud computing, MEC provides low latency due to being in proximity to mobile devices. Also, MEC promises to push computation, storage and network control to the edge of the network (e.g. wireless access points, cellular base stations). However, MEC has limited computing capabilities compared to the conventional cloud computing [34]. Instead, fog computing has the advantage to not only provide low latency but also better compute and storage services anywhere in the device-to-cloud continuum. Moreover, as fog computing provides an abstraction to the underlying communication technology, it can be useful for heterogeneity of IoT devices. In [35], [36], the authors proposed a hierarchical architecture for the task offloading (IoT-fog-cloud) to minimize the path delay and maximize the QoE. Whereas, in such a scenario, using MEC would have the drawback to provide the hierarchical architecture, because it is essentially one layer of nodes located near the end-users.

In this work, we aim to find a reliable path with minimum delay in an IoT-Fog framework. Our method is different from the aforementioned approaches because we consider each objective differently and conflicting with each other. By integrating the benefits of two established technologies, i.e., SDN [13] and fog computing [10] for IoT applications and using a machine learning algorithm, we show that our method can achieve better QoS/QoE as depicted by our experimental results section.

III. Modelling and Notations

We discuss the network modeling in this section. It is important as it will enable the SDN control plane to further exploits the potential of SDN in performing fine-grained management. Our proposed system model consists of network architecture, communication model and provisioning model. We discuss them in detail here.

A. Network Architecture

The underlying physical SDN-enabled IoT-Fog architecture consists of a set of SDN switches V, a set of fog nodes F, a set of IoT devices $\mathbb{I}$ and a set of links L. The architecture is represented by a directed graph $G = (V \cup F,\ L)$. In G, each $l_i \in L$ is associated with QoS parameters, i.e., link reliability R_{li}, bandwidth b_{li}, and transmission delay d_{li}. The SDN controller obtains the QoS-related information from the underlying network periodically, i.e., after t seconds, to optimize the routing policy for a new flow. Additionally, each $l_i \in L$ in the network has a set of associated functions. Function $R(.) : l \rightarrow (0, 1]$ is use to compute the reliability of an $l_i \in L$ and function $D(.) : L \rightarrow d_{li} > 0$ is use to compute the transmission delay.

B. Communication Model

In an IoT-Fog network, fog servers receive continuous flows from IoT end devices. For a given flow, the SDN controller needs to compute a path. The path needs to satisfy two QoS requirements, which are based on the path reliability level and transmission delay. In the proposed model, the IoT flow is represented by a triple, $\Upsilon\ (\mathbb{I}_f, b_f, d_f)$, where $\mathbb{I}_f$ signify the source IoT device of a given flow f, b_f: $\mathbb{I}_f \rightarrow \mathbb{R}^+$, and $d_f > 0$ represent the delay of the flow. For every flow f, the delay should be enforced for transmission.

A graph G that represents the network topology, consists of a set of paths $P = \{p_x\}$, where each path consists of links $p_x = (l_{x(0)}, l_{x(1)}, ..., l_{x(|p_x|)})$. In a path, $x(j)$ shows the index of j^{th} link. The QoS properties of paths and links, i.e, reliability level and transmission delay, are signified by R_{l_i}, d_{l_i}, R_{p_x}, and d_{p_x}. The link failure is often observed in a network G due to external and uncontrollable conditions. We refer these uncontrollable conditions as events E. The reliability function, $R : L \times E \rightarrow (0, 1]$, assigns the reliability level to each $l \in L$ at a specific time t_i slot. Therefore, $R(l_i|e)$ represents the reliability level of any link at a specific event $e \in E$. Furthermore, we assume that links' reliability level are independent of one another. Thus the reliability level of a link l_i can be computed using Eq. 1.

$$R_{t_i}(l_i|E) = 1 - \sum_{e \in E}(1 - R(l_i|e). \tag{1}$$

For a given flow Υ, the reliable path R_{p_x} from a source device $\mathbb{I}$ to the fog server F_n is shown in Eq. 2.

$$R_{p_x}((\mathbb{I}, F_n)|E) = \sum_{l_i \in p_x \in P|E} [I_R(\mathbb{I}, F_n) \times R_{t_i}(l_i|E)] \tag{2}$$

where, $I_R \leftarrow (0, 1]$ shows the reliability indicator of a link in the path p_x.

In the proposed model, function $D(.)$ is used to compute the end-to-end delay on a path $d_{p_x} = (l_{x(0)}, l_{x(1)}, ...)$. The associated delay with links, depend on all flows traversing the links $l_i \in L$. If we neglect the delay demand of a given flow Υ, then the delay can be computed using Eq. 3,

$$d_{p_x} = \sum_{l_i \in p_x} d(l_{x(i)}), \tag{3}$$

where, $d(l_{(i)})$ shows the end-to-end delay of i^{th} link. However, in this work, we will consider the end-to-end delay of a path p_x, which is a combination of transmission delay (packet size/link bandwidth), processing delay, and propagation delay. Therefore, the links' failure and end-to-end delay creates sequential dependence between the links status.

The SDN controller obtains the link status and compute the path status signified by s_{l_i} and s_{p_x}, respectively. The link's status s_{l_i} contains information about the reliability level, transmission delay, processing delay, and propagation delay. Additionally, the path status s_{p_x} contains the end-to-end QoS

matrix. Consequently, based on these status, i.e., s_{l_i} and s_{p_x}, we can conclude two assumptions:

- The link status s_{l_i} depends upon the status of all paths $p_x \in P$ including the link $l_i \in p_x$.
- The path status s_{p_x} depends upon the status of all links $l_i \in p_x$ in the path.

We have formulated these two assumptions in Eq. 4 and Eq. 5 as given here.

$$s_{l_i} = \imath(s_{p_{xi}}, ..., s_{p_{xj}}), \forall l_i \in p_x, x = 1, 2, ..., j, \quad (4)$$

$$s_{p_x} = \jmath(s_{lx(o)}, ..., s_{lx(|p_x|)}), \quad (5)$$

where, $\imath$ and $\jmath$ are optimization functions. However, a direct optimization of $\imath$ and $\jmath$ is not possible here. Therefore, we will use a multi-objective optimization algorithm (discuss in Section IV-C) to achieve a structure for $\imath$ and $\jmath$. The delay cost function $D(.)$ can be formulated mathematically using Eq. 6.

$$d_{p_x}((\mathbb{I}, F_n)|E)) := \sum_f [\jmath(d_f^{Tx} + d_f^{Pro} + d_f^{Prop})], \quad (6)$$

where, d_f^{Tx}, d_f^{Pro}, and d_f^{Prop} show the transmission delay, processing delay, and propagation delay, respectively.

C. Provisioning Model

As mentioned in Section III-B, a flow f is provisioned with two QoS parameters, which are the path reliability and transmission delay. Based on these parameters, we make the following explanation.

Path Reliability (Max): It is the minimum link reliability in all links making a path. The objective is to find a path that has maximum reliability. In the proposed model, we compute this parameter using Eq. 7.

$$max(min(\sum_{p_x \in P|E} R_{p_x}\ I_R(\mathbb{I}, F_n))). \quad (7)$$

Path's Delay (Min): It is the mean of total delay of all the links making a path $p_x(\mathbb{I}, F_n)$. The objective is to find a path that has minimum end-to-end delay. In the proposed model, we compute this parameter using Eq. 8,

$$min \sum_{p_x \in P} d_{p_x}(\mathbb{I}, F_n). \quad (8)$$

IV. Reliability-Aware Communication in SDN-enabled IoT-Fog

In this section, we discuss our proposed approach for a reliable communication in an SDN-enabled IoT-Fog. In the following sub-section, we start with providing an overview of a multi-hop network scenario that we have considered for the transmission of flows from IoT end-devices to the fog servers.

A. Multi-Hop SDN-Enabled IoT-Fog Architecture

In order to communicate with the fog servers in an SDN-enabled IoT network, we consider a multi-hop scenario for data transmission. Figure 1 shows the architecture of SDN-enabled IoT-Fog. As mentioned in Introduction (Section I), deploying the fog servers at the edge is costly and companies are often reluctant to do so. Therefore, fog server are deployed in such a way that end-users' devices can access them in a multi-hop fashion. The end-devices are connected to the SDN switches via access points.

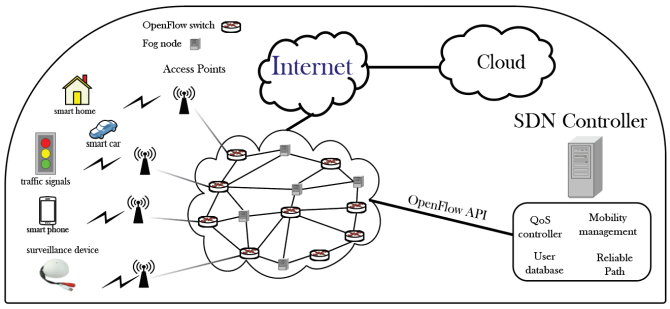

Fig. 1. A multi-hop SDN-enabled IoT-Fog architecture. The things layer of IoT communicate with fog servers in a multi-hop manner, where the intermediate nodes are the SDN switches.

B. Finding Link Reliability

In this work, we employ an ML-based algorithm to build a model that find the reliability of a link in a network. We train the ML-based model on historical statistics, which are based on link failure events of a real-life network. For this purpose, we use a regression-based k-nearest neighbor algorithm (k-NNR), which is one of the most popular supervised learning and non-parametric algorithm. Furthermore, k-NNR is used to identify the non-linear and complex relationship among observations using nearest neighbors' rules. For example, when a new link failure $l_x \in L$ event occurs, k-NNR works under the assumption that the link failure event shares a similar property with similar historical links failure events. Therefore, using the model (based on the k-NNR), the reliability level for the links are predicted. Moreover, k-NNR can use different distance metrics such as Euclidean, Manhattan, or Makowski to find the closest/nearest neighbors [18]. In this work, we consider the Euclidean distance, as shown in Eq. 9,

$$Euclidean\ distance\ (r_i, l_x) = \sqrt{(r_i, l_x)^2}, \forall l_x \in L, \quad (9)$$

where, $(r_i,\ l_x)$ shows the distance between the target reliability value of r_i.

1) Dataset - A Real-life Fog-based network: We train the k-NNR algorithm with a data set that includes a realistic data capture of links' failure of an anonymous fog-based network. The data is recorded for a time interval of over 5 months. The data set shows that a link goes down almost every 22 minutes. As our primary goal is to study link failures, we focus only on those link State Packets (SPs) that report a link failure or be reinstated after a failure. In Fig. 2, the distribution of link failure events is shown. We can see that the link failure

events are spread out reasonably well across the given period. Moreover, a zoomed view of the events highlighting only one month duration of the link's failure is shown in Fig. 3.

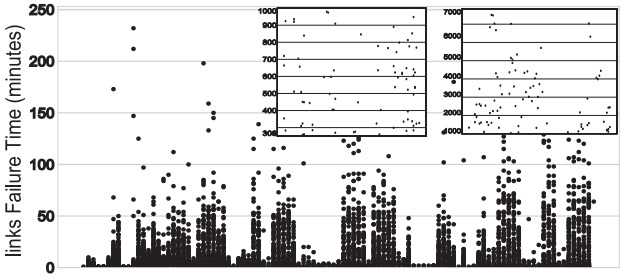

Fig. 2. Real-time Captured Notification of Link Failure Events over Timescale (April 15, September 21, 2019)

Furthermore, the data traces recorded for each link l_i of the network include parameters such as $Day_{time}, l_{i_{downtimestart}}, l_{i_{uptime}}, l_{i_{downfrequency}}$, and $l_{i_{failurereason}}$. The $l_{i_{downtimestart}}$ is the occurrence time of a link failure, and $l_{i_{uptime}}$ is when the link is recovered or become operational. The $l_{i_{downfrequency}}$ refers to the number of times a link was down in a specific time slot of a day. The $l_{i_{failurereason}}$ shows the reasons for link failure, e.g. due to power, configuration, hardware, and software malfunctioning. The total down duration of a link is $l_{i_{downtime}}$ = $l_{i_{uptime}}$ - $l_{i_{downtimestart}}$. Furthermore, we computed the reliability level of each link l_i in the data set using Eq. 1.

2) k-NNR Algorithm: Algorithm 1 shows our implementation of the k-NNR. It takes the data set as an input, we split the data into two sets, to be used for training (X_{train}) and testing (X_{test}). The K-NN regression method is based on three steps, which results in predicting the accuracy of a link's reliability level $l_x \in L$ and returns the predicted value. The distance array is calculated in step-1, which is to calculates the distance between a sample in (X_{test}) and all other samples in the training data set (X_{train}), using the Euclidean distance of Eq. 9. In step-2, the nearest k neighbors in data X_{test} is selected. Finally, in step-3, the predicted value is estimated and returned.

Algorithm 1: A Regression-based k-NN Algorithm to predicts links' reliability-level of an SDN-enabled network.

input : Link $l_x \in L$, training dataset (Z) having (m) samples, nearest neighbors (k).
Result: Return the reliability-level $pred_{rel}$ of $l_x \in L$

```
//Step-1 calculate distance array
for i = 1 to m do
    for y = 1 to m do
        E-dist[y] ← 0 // Euclidean distance
        for z = 1 to n do
            E-dist[y] ← E-dist[y] + (X_testz, X_trainyz)^2
        end
        E-dist[y] ← sqrt(E-dist[y])
    end
    // Step-2, Sort distance array
    for s = 1 to k do
        neareastSet[s] = E-dist[s]
    end
    sum ← 0 //Step-3
    for s ∈ neareastSet do
        sum ← sum + s
    end
    pred_rel ← sum/k
    return pred_rel
end
```

C. Multi-Objective Optimization

Multi-objective optimization (MOO) algorithms represent a viable alternative to finding the Pareto-optimal approximation set in one run. We will use NSGA-II [27] - an MOO algorithm to find an optimal path or set of Pareto-optimal paths that optimize the trade-off between the two objectives: 1) maximum reliability level of a path, i.e., Eq. 7, and 2) minimum end-to-end delay of a path, i.e., Eq 6.

1) Pareto optimality: Pareto optimality is a state when a solution or a set of solutions in the solution space are achieved so that no other feasible solution reduces at least one objective function without increasing another one [37]. As in multiple-objective optimization, there are more than one objective to be optimized simultaneously. During the process of optimization, one of the challenging steps occurs when the objectives are conflicting with each other. There exists a natural trade-off between the objectives, which creates a set of optimal solutions using Pareto-optimal theory [27]. These solutions are popularly known as Pareto-optimal solutions or non-dominated set of solutions. To represent a Pareto-optimal set or non-dominated solutions on a two-dimensional graph, an approximation front called Pareto front/Pareto frontier is used.

2) NSGA-II: The Non-dominated Sorting Genetic Algorithm (NSGA-II) [27] is a fast elitist population-based algorithm for multi-objective optimization. Algorithm 2 shows our implementation of the NSGA-II. Moreover, NSGA-II is based on elitism. In a population of a generation, the elites have the opportunity to be carried to the next generation. Also, the non-dominated solutions in NSGA-II algorithm are preserved.

We implemented the NSGA-II by executing it on the SDN controller. It finds the Pareto-optimal approximation of paths from source SDN switches ($r \subset V$) to the fog servers (F). As the link states of the network changes on-the-fly, the controller searches the updated paths. This enables the SDN controller to select the best path according to the requested flow for achieving the required QoS level. We will discuss more about adaptive selection of path in Section V.

V. Adaptive Communication in SDN Enabled IoT-Fog

Adaptivity enables the IoT applications running on the end-devices to achieve better QoE. Based on the type of packets, the SDN controller can choose a path from the Pareto-optimal set. In other words, the behavior of the controller will change

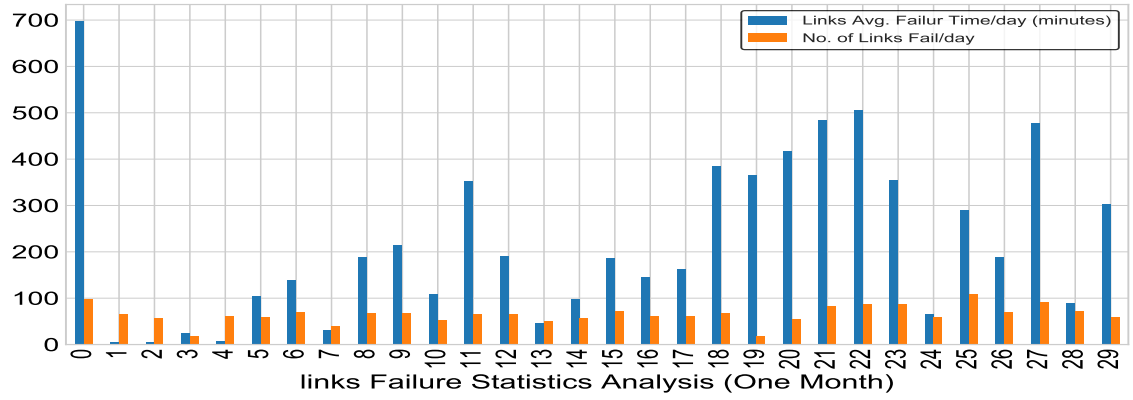

Fig. 3. The plot showing links failure statistics per day of a real-life network.

Algorithm 2: Implementation of NSGA-II to optimise tradeoff between reliability and delay of a path.

input : G: The global view of IoT-Fog topology with links information such as link reliability and delay, F:list of Fog servers, $r \subset V$: list of source SDN Switches.
Result: return the Pareto-optimal set of paths

```
for s ∈ r do
    for d ∈ F do
        paths ← Networkx.all_simple_paths(G, s, d)
        binaryEncode(paths)
        NSGA(paths, populationSize, TournamentSize,
        crossoverProb, mutationProb)
        pop ← initializePopulation()
        R_pop ← pop
        for k = 1 to numGenerations do
            pop ← findChildPop(R_pop)
            evoluePopulation(pop) // i.e., Using the
              costs functions in Eq. 6 and Eq. 7
            find_fronts(pop)
            find_Crowding_distance(pop)
            R_pop ← elitistPop(pop)
        end
        PF_Paths{s:d} ← R_pop
    end
end
```

at run-time in response to the changing QoS requirements to make better decisions on how to use the available resources.

In this work, we implemented an adaptive decision mechanism in the SDN controller. This was achieved by considering the output of NSGA-II algorithm - the Pareto-optimal paths from source SDN switches ($r \subset V$) to the fog servers. The source SDN switches are those to which host terminals are connected directly or indirectly, i.e., through wireless access points. Therefore, when a new packet arrives on any of these source switches, the SDN controller (on-the-fly) checks the packet type and decides which path should be selected. For real-time applications, which are delay-sensitive in nature and generally use UDP packets, a path that has the lowest end-to-end delay will be selected. For those applications that use TCP packets for their communications, i.e., computation offloading or HTTP requests, a path from the Pareto-optimal set that has a higher reliability level will be selected.

VI. Experimental Work and Results

A. Simulation Settings

We evaluate the performance of the proposed approach using Mininet [1] network emulator along with POX [2] SDN controller. We used Intel core i7 PC with 3.40 GHz CPU and 12 GB of RAM, running Ubuntu 16.04 OS having Linux kernel version 4.4.

We created a network topology that has a total of 38 nodes, i.e., SDN switches, and 586 links connecting them. The graph G, representing the network topology, was well connected and nearly a complete graph. The nodes in G represent the SDN switches (V) that are connected to a Pox controller. We randomly selected 10 nodes ($r \subset V$), which were connected directly (or indirectly via wireless access point) to the host terminals. We selected three fog servers placed in random and distance positions from each other. For each of the host terminals, the nearest fog server having the minimum number of hops was chosen as the offload target.

B. Results and Discussion

1) Finding links reliability level: We created a Python-based script to execute the simulation in the Mininet by employing the network topology. For the Pox controller to obtain a global view of the network with link failure statistics, we generated traffic from hosts to fog servers. We achieved this by enabling the built-in Python-based HTTP server at the fog servers and the hosts were set to send HTML page requests continuously for a random amount of time. Furthermore, we enforced to randomly make some of the intermediate links down for a random and a small amount of time. As the k-NNR algorithm was deployed on the controller, it was calculating the links reliability level inline with the failure events.

[1] http://mininet.org
[2] http://github.com/noxrepo/pox

2) Finding Pareto-optimal paths: The Pox controller executed the NSGA-II algorithm in order to find the Pareto-optimal paths between the source SDN switches (through which the hosts were connected) and the fog servers. NSGA-II starts by creating a population of individuals (i.e possible paths). We used binary representation of individuals, where *1* indicates the link is considered while *0* indicates the link is not considered. To select the parents in each generation, we used the tournament selection. To create a child, we used Uniform Crossover operator. With the crossover probability as 0.5, each gene (bit) is selected randomly from one of the corresponding genes of the parent chromosomes. We used Flip Mutation which works by flipping a gene (bit). With a very low mutation probability 0.2, it involves changing 0 to 1 and 1 to 0.

To find out the quality of solution set (Pareto-frontier) produced in each generation by the NSGA-II, we used a unary indicator called hypervolume indicator (S-metric). The hypervolume is a unary value, which is calculated as the sum of the areas formed by points on the non-dominated front and a chosen reference point (w). In Fig. 4, the hypervolume of the Pareto-front in each generation is shown. We used a maximum of 30 generations in NSGA-II. Moreover, it can be seen in cases of all the switches that the calculated hypervolume value remains constant after the 20^{th} generation. This indicates that at this point the algorithm has fully converged in finding the Pareto-optimal approximation of paths.

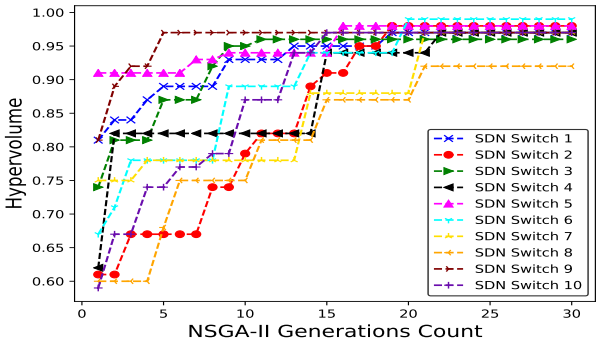

Fig. 4. The plot shows the hypervolume of the Pareto-front approximation obtained throughout the generations of NSGA-II MOO algorithm.

In Table I, we provide the results of the NSGA-II algorithm. The average time to find the paths from the source SDN switches to the fog servers was 16 seconds. The two extreme paths making the Pareto-front approximation and the shortest path are stated along with their reliability level and end-to-end delay. Furthermore, we have randomly selected the results of only three SDN switches, which we will be using to demonstrate the adaptive decision mechanism in the following subsection.

3) Adaptive communication: In order to evaluate how an IoT application will perform in terms of communicating to the fog servers using the reliable path (p_r), the shortest path (SP) and the path having less end-to-end delay (p_d), we used two different applications (Apps). The first application (App-1) sent continuous data stream to the fog server using UDP sockets for analysis. The second application (App-2) offloads a computationally-intensive task to the fog server for processing. For this simulation, we stopped updating the links failure statistics, so that no new paths are calculated. In Fig. 5, we can see that App-1 performed well (using multiple runs) when the Pareto-optimal path with the lowest delay (P_d) was chosen. Moreover, the result was the same when this app was executed on different host terminals communicating with different fog servers. This implies that for a real-time application, which relies on UDP packets for communication, the SDN controller would choose a path with low end-to-end delay, i.e., p_d.

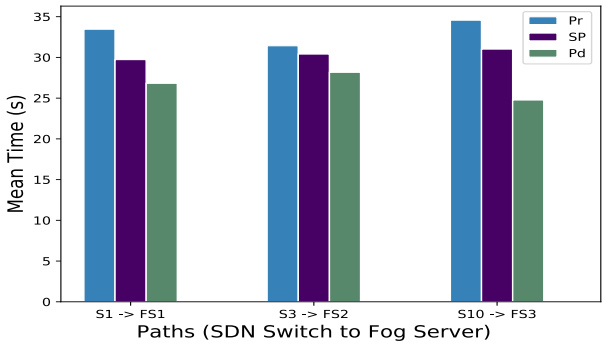

Fig. 5. A real-time application execution completion time using different paths.

For App-2, we used a controlled environment to randomly make the intermediate links down for some time (seconds) in order to emulate link failure events, based on the paths reliability. The simulation was carried out by executing the app for multiple times (ten runs). In Fig. 6, we can see that the app performed well while using the Pareto-optimal path with high reliability (p_r). The packet loss was relatively lower as compared to the app using the other two paths.

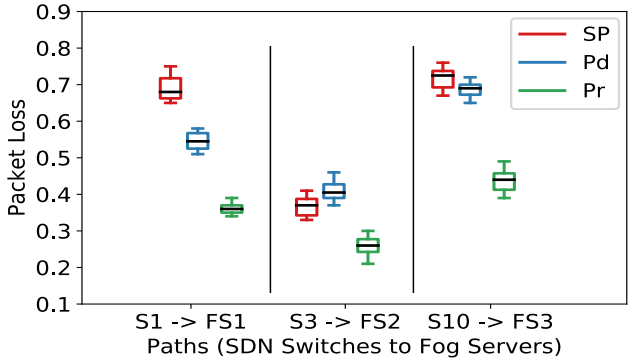

Fig. 6. Packet loss during offloading task using different paths.

Analysis of results shows that using the adaptive decision mechanism at the SDN controller, both the applications (App-1 and App-2) met the required QoS requirements. The adaptive decision to select the best path, p_d, for App-1, resulted in finishing its execution in 13% less time than if the shortest path, SP, would have been selected, and 20% less time than if the reliable path, p_r, would have been selected. Similarly,

TABLE I
THE RESULT OBTAINED FOR 3 SWITCHES AFTER EXECUTING NSGA-II ALGORITHM 2. THE EXECUTION TIME FOR FINDING THE PARETO-OPTIMAL PATHS, SHORTEST AND EXTREME PARETO-OPTIMAL PATHS ARE SHOWN ALONG WITH THEIR RELIBILITY LEVEL AND END-TO-END DELAY.

SDN Switch	Fog Server	Execution Time (s)	Paths	Path Reliablility Level	Path End-to-End Delay (s)
SDN Switch 1	Fog Server 1 (node=14)	16.48	SP = [1,2,13,14]	0.56	0.91
			P_r = [1,3,16,20,21,15,14]	0.84	1.04
			P_d = [1,2,15,13,14]	0.71	0.83
SDN Switch 3	Fog Server 2 (node=27)	17.61	SP = [3,16,20,25,27]	0.84	0.94
			P_r = [3,6,12,10,15,25,27]	0.87	0.97
			P_d = [3,16,20,21,15,23,27]	0.81	0.86
SDN Switch 10	Fog Server 3 (node=38)	16.73	SP = [10,25,33,37,38]	0.62	0.96
			P_r = [10,15,23,34,36,38]	0.80	1.1
			P_d = [10,15,23,34,37,38]	0.66	0.76

selecting a reliable path for App-2 faced less packet loss than if the their data would have been sent on the shortest path or p_d. On average, the packet loss on path p_r was 41% less than SP and 36% less than p_d.

VII. CONCLUSION

In this paper, we presented a novel approach to achieve adaptive and reliable communication in an SDN-enabled IoT-Fog. A regression-based k-nearest neighbor algorithm is used to find the reliability level of links trained with real-life network traces recorded for 5 months. A popular MOO algorithm (NSGA-II) is used to find the Pareto-optimal paths that optimizes the trade-off between two objectives. An adaptive decision mechanism is used in which the SDN controller selects the best path for different types of applications, based on the packet types (UDP and TCP).

We used two different applications to evaluate the proposed method: 1) a real-time and delay-sensitive application (App-1) that was sending data to the fog server using UDP sockets, and 2) a second application (App-2) that was offloading a computationally-intensive task to the fog server for processing. The demands of both applications were different. A path with lowest possible delay needed to be selected for the first app, while the second app needed a reliable path to communicate. THe results showed that using the adaptive decision mechanism at the SDN controller, both the applications met the required QoS requirements.

In the future, we are interested in pursuing further work on the proposed approach and related case-study applications. Particularly, by evaluating different strategies for other QoS metrics beyond the execution time of the applications such as energy efficiency and the cost of using the fog servers. Furthermore, creating case studies by conducting real-world experiments involving users that exercise the system instead of using the controlled environment.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568

[2] Y. Sun, H. Song, A. J. Jara, and R. Bie, "Internet of things and big data analytics for smart and connected communities," *IEEE Access*, 2016.

[3] L. Toka, B. Lajtha, E. Hosszu, B. Formanek, D. Gehberger, and J. Tapolcai, "A resource-aware and time-critical IoT framework," in *IEEE INFOCOM*, 2017, pp. 1–9.

[4] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *IEEE INFOCOM*, 2018, pp. 783–791.

[5] M. Barua, X. Liang, R. Lu, and X. Shen, "Peace: An efficient and secure patient-centric access control scheme for ehealth care system," in *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2011, pp. 970–975.

[6] L. Columbus, "Internet of things market to reach $ 267b by 2020," *Forbes*. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2017/01/29/internet-of-things-market-to-reach-267b-by-2020/

[7] A. Akbar and P. R. Lewis, "Towards the optimization of power and bandwidth consumption in mobile-cloud hybrid applications," in *Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 213–218.

[8] G. Dartmann, H. Song, and A. Schmeink, *Big data analytics for cyber-physical systems: machine learning for the internet of things*. Elsevier, 2019.

[9] W. G. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, 2018.

[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[11] S. Misra and N. Saha, "Detour: Dynamic task offloading in software-defined fog for IoT applications," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1159–1166, 2019.

[12] J. Liu, J. Wan, B. Zeng, Q. Wang, H. Song, and M. Qiu, "A scalable and quick-response software defined vehicular network assisted by mobile edge computing," *IEEE Communications Magazine*, 2017.

[13] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[14] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *IEEE INFOCOM*, 2013, pp. 2211–2219.

[15] C.-Y. Chu, K. Xi, M. Luo, and H. J. Chao, "Congestion-aware single link failure recovery in hybrid SDN networks," in *IEEE INFOCOM*, 2015, pp. 1086–1094.

[16] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: understanding the causes and impact of network failures," in *ACM SIGCOMM*, 2010, pp. 315–326.

[17] G. IANNACCONE, "Analysis of link failures in a IP backbone," *Internet Measurement Workshop, http://www. icir. org/vern/imw-2002/imw2002-papers/202. pdf*, 2002.

[18] O. Kramer, "Unsupervised k-nearest neighbor regression," *arXiv preprint arXiv:1107.3600*, 2011.

[19] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[20] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Adaptive and reliable multipath provisioning for media transfer in sdn-based overlay networks," *Computer Communications*, vol. 106, pp. 107–116, 2017.

[21] J. Luo, X. Deng, H. Zhang, and H. Qi, "Qoe-driven computation offloading for edge computing," *Journal of Systems Architecture*, vol. 97, pp. 34–39, 2019.

[22] Z. Sun, Y. Zhang, Y. Nie, W. Wei, J. Lloret, and H. Song, "Casmoc: a novel complex alliance strategy with multi-objective optimization of coverage in wireless sensor networks," *wireless Networks*, 2017.

[23] A. Akbar and P. R. Lewis, "The importance of granularity in multiobjective optimization of mobile cloud hybrid applications," *Transactions on Emerging Telecommunications Technologies*, 10 2018.

[24] A. Akbar, P. R. Lewis, and E. Wanner, "A self-aware and scalable solution for efficient mobile-cloud hybrid robotics," *Frontiers in Robotics and AI*, 2020.

[25] B. S. P. Reddy and C. S. P. Rao, "A hybrid multi-objective ga for simultaneous scheduling of machines and agvs in fms," *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 5, pp. 602–613, Dec 2006.

[26] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, Nov 1999.

[27] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *Parallel Problem Solving from Nature PPSN VI*. Springer Berlin Heidelberg, 2000, pp. 849–858.

[28] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, 2017.

[29] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan, "Detour: informed internet routing and transport," *IEEE Micro*, vol. 19, no. 1, pp. 50–59, 1999.

[30] P. Charonyktakis, M. Plakia, I. Tsamardinos, and M. Papadopouli, "On user-centric modular qoe prediction for voip based on machine-learning algorithms," *IEEE Transactions on Mobile Computing*, vol. 15, no. 6, pp. 1443–1456, 2016.

[31] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, "A machine learning approach to classifying youtube qoe based on encrypted network traffic," *Multimedia tools and applications*, vol. 76, no. 21, pp. 22 267–22 301, 2017.

[32] T. Abar, A. Ben Letaifa, and S. El Asmi, "Machine learning based qoe prediction in sdn networks," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 1395–1400.

[33] G. Brown, "Mobile edge computing use cases and deployment options," *Juniper White Paper*, pp. 1–10, 2016.

[34] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[35] H. Shah-Mansouri and V. W. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, 2018.

[36] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 998–1010, 2018.

[37] J. S. Arora, *Introduction to optimum design*. Elsevier, 2004.

Aamir Akbar is currently a lecturer in computer science at Abdul Wali Khan University Mardan (AWKUM) in Pakistan. He obtained a PhD degree at Aston University, UK. Before that, he was a lecturer at COMSATS University in Pakistan, and before that, he was reading for an MSc degree at Oxford Brooks University in the UK. His research is on mobile cloud computing, including but not limited to, edge/fog computing, IoT, SDN. His work leverages multi-objective optimisation, evolutionary computation, machine learning (AI), self-adaptivity and self-awareness to tackle problems.

Muhammad Ibrar is pursuing his Ph.D. degree at the School of Software, Dalian University of Technology, China. He has completed his MS degree in Telecommunication and Networking from Bahria University, Islamabad, Pakistan, in 2014. He received his BS degree in Telecommunication and Networking from COMSATS University Islamabad, Abbottabad Campus, Pakistan, in 2010. His research interests include Software-Defined Networking (SDN), fog computing, wireless ad-hoc, and sensor networks.

Mian Ahmad Jan completed his Ph.D. at University of Technology Sydney, Australia. He is an Assistant Professor in the Department of Computer Science, Abdul Wali Khan University Mardan, Pakistan. He has been a visiting faculty member at Northwestern Polytechnical University, Xian, China since October 2017. He has published more than 40 papers in highly-reputed IEEE transactions and journals. His research interests are focused mainly on energy-efficient routing, secure communication, and resource management in wireless sensor networks, Internet of things, and edge computing.

Ali Kashif Bashir is working as an Associate Professor in Department of Computing and Mathematics, Manchester Metropolitan University, UK. He received his Ph.D. degree in computer science and engineering from Korea University, South Korea. His research interests include cloud computing, NFV/SDN, network virtualization, network security, IoT, computer networks, RFID, sensor networks, wireless networks, and distributed computing. He is serving as the Editor-in-chief of the IEEE INTERNET TECHNOLOGY POLICY NEWSLETTER and the IEEE FUTURE DIRECTIONS NEWSLETTER.

Lei Wang is currently a full professor at the School of Software, Dalian University of Technology, China. He received his Ph.D. from Tianjin University, China. He was a member of technical staff with Bell Labs Research China (2001-2004), a senior researcher with Samsung, South Korea (2004-2006), a research scientist with Seoul National University (2006-2007), and a research associate with Washington State University, Vancouver, WA, USA (2007-2008). His research interests involve sensor networks, social networks, and network security.