Dynamic Adaptation of Software-defined Networks for IoT Systems: A Search-based Approach

Seung Yeob Shin
University of Luxembourg, Luxembourg
seungyeob.shin@uni.lu

Shiva Nejati
University of Ottawa, Canada
University of Luxembourg, Luxembourg
snejati@uottawa.ca

Mehrdad Sabetzadeh
University of Ottawa, Canada
University of Luxembourg, Luxembourg
msabetza@uottawa.ca

Lionel C. Briand
University of Ottawa, Canada
University of Luxembourg, Luxembourg
lbriand@uottawa.ca

Chetan Arora
SES Networks, Luxembourg
University of Luxembourg, Luxembourg
Deakin University, Australia
chetan.arora@deakin.edu.au

Frank Zimmer
SES Networks, Luxembourg
frank.zimmer@ses.com

ABSTRACT

The concept of Internet of Things (IoT) has led to the development of many complex and critical systems such as smart emergency management systems. IoT-enabled applications typically depend on a communication network for transmitting large volumes of data in unpredictable and changing environments. These networks are prone to congestion when there is a burst in demand, e.g., as an emergency situation is unfolding, and therefore rely on configurable software-defined networks (SDN). In this paper, we propose a dynamic adaptive SDN configuration approach for IoT systems. The approach enables resolving congestion in real time while minimizing network utilization, data transmission delays and adaptation costs. Our approach builds on existing work in dynamic adaptive search-based software engineering (SBSE) to reconfigure an SDN while simultaneously ensuring multiple quality of service criteria. We evaluate our approach on an industrial national emergency management system, which is aimed at detecting disasters and emergencies, and facilitating recovery and rescue operations by providing first responders with a reliable communication infrastructure. Our results indicate that (1) our approach is able to efficiently and effectively adapt an SDN to dynamically resolve congestion, and (2) compared to two baseline data forwarding algorithms that are static and non-adaptive, our approach increases data transmission rate by a factor of at least 3 and decreases data loss by at least 70%.

KEYWORDS

Search-based Software Engineering, Dynamic Adaptive Systems, Internet of Things, Software-defined Networks

SEAMS 2020, 25 – 26 May, 2020, Seoul, South Korea

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C. Briand, Chetan Arora, and Frank Zimmer. 2020. Dynamic Adaptation of Software-defined Networks for IoT Systems: A Search-based Approach. In *Proceedings of 2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing (SEAMS 2020).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The recent proliferation of sensors, actuators and inexpensive network-enabled devices in homes, workplaces, public spaces and nature provides several opportunities to build intelligent systems that can improve our lives in many different ways. These devices, when used in combination with wired and wireless connectivity, have created a surge of interest in the concept of Internet of Things (IoT). Systems enabled by IoT perform a task by connecting sensors and actuators and many previously unconnected things through the Internet [3, 10]. A notable example of an IoT-enabled system is an emergency management system that monitors a large geographical area through a network of sensors to detect potential disasters (e.g., fire, floods, hurricanes, earthquakes) as early as possible and to provide a communication platform between the responsible organizations and people to enable quick action and minimize loss of life and damages.

Successful IoT systems necessarily depend on an underlying communication system that can transmit large volumes of data in an efficient, effective and flexible way. Such a communication system should, in particular, be able to adapt to changes in the environment and maintain a reasonable quality of service when, for example, the traffic for a particular network route increases dramatically due to a massive demand from system users. Recently, software-defined networks (SDN) [33] have started to enable such flexible and effective communication systems. The idea behind SDN is to transfer the control of networks from localized fixed-behavior controllers distributed over a set of switches to a centralized and programmable software controller that can react to environment changes in a timely fashion by efficiently reconfiguring the entire network. With

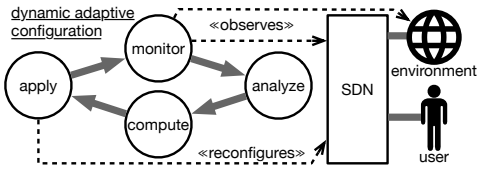


Figure 1: An overview of our *Dynamic adaptIve CongEstion control algorithm for SDN (DICES).*

software being an integral part of SDN, developing network controllers needs interdisciplinary considerations which include not only network engineering, but also *software engineering* [49].

For an IoT system that builds on SDN, the controller is responsible for ensuring that the network is configured in such a way as to maintain the quality of service at a desired level. In this paper, we focus on developing effective reconfiguration techniques for SDN to improve the quality of service in IoT systems. Such techniques should be able to continuously monitor environment changes and dynamically reconfigure the system accordingly in order to optimize multiple quality of service criteria such as minimizing data loss, communication delays and reconfiguration costs. There are a number of existing research threads on ensuring the quality of service for traditional networks [4, 14, 30, 36, 51]. Some more recent approaches study dynamic reconfiguration of SDN to maximize quality of service [24, 32, 39]. None of these lines of work, however, consider or optimize the configuration of an SDN for *multiple* quality of service criteria simultaneously. The problem of configuration for the purpose of optimizing multiple criteria has been studied in prior research threads for design-time software development [8, 61, 71]. These studies, however, are geared toward offline optimization of system design or architecture, and cannot address the challenge of online and dynamic SDN reconfiguration.

In this paper, we propose a dynamic adaptive configuration technique to resolve congestion in SDN in an online manner, while minimizing data transmission delays and reconfiguration costs. We refer to our approach as *Dynamic adaptIve CongEstion control algorithm for SDN (DICES)*. Inspired by feedback-loop control systems [43], DICES realizes the control loop shown in Fig. 1 and consisting of the following steps: (1) *monitor* the SDN to collect network information, (2) *analyze* the network to determine whether it is congested, (3) *compute* a reconfiguration if congestion is detected, and (4) *apply* the new configuration to the actual SDN. The control loop is executed periodically and may reconfigure the SDN at each period if congestion is detected. The "compute" step of DICES uses a tailored multi-objective search algorithm to optimize multiple quality of service criteria simultaneously. Specifically, the algorithm minimizes the following three objectives: *network-link utilization*, *transmission delay* and *reconfiguration cost*. In order to be executed in a real-time manner, DICES has to be efficient. Hence, instead of searching for the very best reconfiguration option, the approach aims to find good-enough solutions sufficiently quickly. Consistent with this goal, we build on the research field of dynamic adaptive search-based software engineering (SBSE) [34] to enable the computation component in charge of the reconfiguration of an SDN.

DICES has to be integrated and executed together with an actual SDN control platform. We develop DICES as an application within ONOS [13] – a widely used open-source SDN control platform.

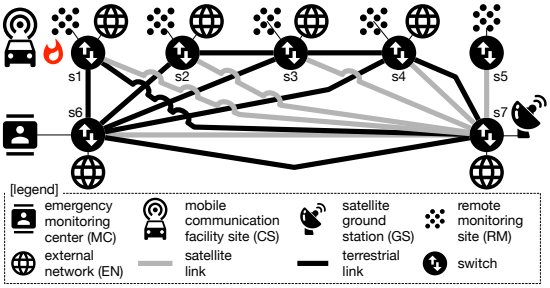


Figure 2: A conceptual view of an emergency management system (EMS).

To evaluate DICES, we rely on an open-source network emulator, Mininet [47]. Mininet enables us to create and emulate realistic virtual networks with different topologies and characteristics. Alongside Mininet, we employ an open-source network traffic flow generator, D-ITG [18], to generate IoT traffic scenarios by combining sensor, video, audio and data streams. Our implementation of DICES can be integrated in a straightforward way into an actual network system. Nevertheless, we elect to evaluate our approach based on emulation through Mininet. This choice, which follows standard engineering practice, is motivated by two main factors: First, the large-scale and systematic experiments that we perform would be prohibitively expensive to set up using real hardware. Second, we need to evaluate our approach on varying networks with different sizes and properties. With real hardware, we would not have the required flexibility.

We assess the performance of DICES on ten synthetic and one industrial SDN. The industrial SDN is a national emergency management system in Luxembourg. The information about the topology and the IoT traffic scenarios for this SDN is provided by SES, a leading satellite operator, which is in charge of assessing the infrastructure for the national emergency management system. Our results show that: (1) DICES efficiently and effectively adapts an SDN to resolve congestion, (2) the execution time of DICES scales linearly with the network size and the number of traffic flows, and (3) compared to two baseline solutions commonly used in practice [1, 5, 16, 17, 21, 58, 62], DICES leads to data transmissions that are at least 3 times faster while reducing data loss by at least 70%. Our case study data is available online [64].

Organization. The rest of this paper is organized as follows. Section 2 motivates the paper. Section 3 describes DICES. Section 4 evaluates DICES. Section 5 compares with related work. Section 6 concludes this paper.

2 MOTIVATING CASE STUDY

We motivate our work with an IoT-enabled national emergency management system, currently under study by SES, for public protection and disaster relief. We refer to this system as *EMS* in the rest of the paper. EMS is responsible for generating early warnings about potential disasters, detecting natural or man-made emergencies, and facilitating response/recovery operations by providing emergency workers or governmental bodies with a reliable and efficient communication and data transfer infrastructure.

Fig. 2 shows a conceptual view of EMS for an example topology suggested by SES. EMS employs SDN to interconnect four types

of sites, namely *remote monitoring site*, *emergency monitoring center*, *satellite ground station*, and *mobile communication facility site*. The interconnections are realized using SDN switches (s1–s7), terrestrial links (e.g., optical fiber links) and satellite links. The key characteristics of the four EMS sites are described below.

• *Remote monitoring sites (RM)* continuously monitor and gather environment data using sensor networks. In Fig. 2, switches s1–s5 are connected to remote monitoring sites.

• *Emergency monitoring centers (MC)* control and monitor the entire EMS by aggregating data from the remote sites. They further facilitate decision making for emergency handling by controlling the entire network and by processing the aggregated data. EMS has one emergency monitoring center attached to s6, as depicted in Fig. 2.

• *Satellite ground stations (GS)* are responsible for routing data streams transmitted by satellites. All satellite connections need to pass through a satellite ground station. EMS has one satellite ground station attached to s7, as shown in Fig. 2.

• *Mobile communication facility sites (CS)* are used by emergency workers and first responders for communication during an actual emergency. Unlike the other EMS sites that are operational at all times, the mobile communication facility comes into play only during or after an emergency. The mobile facility site is primarily used as a communication hotspot for audio and video transmission between a remote monitoring site and the emergency monitoring center. In our case study, we assume that an emergency situation, e.g., a natural disaster, occurs in the area close to s1. Hence, in Fig. 2, the mobile communication facility site is located at s1.

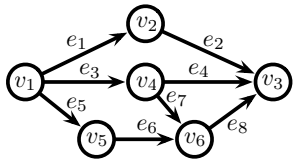
Finally, as shown in Fig. 2, the EMS network can further be connected to *external (legacy) networks (EN)* to allow access to remote monitoring sites.

During an emergency, the EMS data traffic volume increases by many folds. The remote monitoring sites transmit monitored data streams to the emergency monitoring center. The mobile communication facility site and the emergency monitoring center exchange high-bandwidth demanding streams such as high definition video and audio for real-time updates. The emergency monitoring center sends earth-observation images (i.e., maps) to the mobile communication facility site in order to help plan an appropriate recovery strategy.

EMS is highly prone to congestion during emergencies due to the increased volume of demand. Such congestion leads to increased latency, information loss and inability to communicate with one or more sites. While such congested networks are common during emergencies, critical systems such as EMS are expected to be resilient and find ways to avoid or mitigate congestion. Failing to do so can have dire consequences. EMS is thus subject to strict quality of service requirements so that it will operate through network issues without intolerable delays or information loss. To this end, SES is interested in DICES as a way to ensure that EMS can sustain emergency situations and satisfy its quality of service requirements.

3 APPROACH

The separation between software-defined data control and the physical aspects of network systems is a key feature of SDN [33]. The SDN architecture is composed of three layers: *infrastructure*, *control*, and *application*. The infrastructure layer is comprised of physical entities such as links and switches that enable data flows based on forwarding rules instructed by the control layer. The control layer hosts one or multiple SDN controllers distributed across the network. This layer is responsible for managing infrastructure entities, e.g., switches and links, based on algorithms provided by the application layer. In Section 3.1, we provide an abstract formalization of SDN concepts and use them to define the problem of network congestion.



(a) An example network

bandwidth Mbps	$c(e_1)$ 20	$c(e_2)$ 20	$c(e_3)$ 10	$c(e_4)$ 10	$c(e_5)$ 20	$c(e_6)$ 20	$c(e_7)$ 10	$c(e_8)$ 20
delay ms	$l(e_1)$ 250	$l(e_2)$ 250	$l(e_3)$ 250	$l(e_4)$ 250	$l(e_5)$ 25	$l(e_6)$ 25	$l(e_7)$ 25	$l(e_8)$ 25

(b) $c(e)$ and $l(e)$ values

Figure 3: An example network: (a) network topology and (b) $c(e)$ bandwidth and $l(e)$ delay values for each link in the network topology.

The behavior of the control layer can be modified and extended by the application layer. Users can develop their own applications to apply domain-specific data forwarding, security or failure management algorithms. Specifically, the SDN application layer includes a data-forwarding algorithm that directs data flows between any pair of switches through the weighted shortest path between the switches. This default data-forwarding algorithm is described in Section 3.2. Since SDN controller behavior is programmable through applications, we can enhance the data-forwarding function of SDN using DICES as described in Section 3.3.

3.1 Problem Description

In this section, we describe SDN topologies using directed graphs and formalize SDN traffic concepts. We then define the problem of network congestion. We define an SDN network as a tuple $G = (V, E, c, l)$, where V is a set of switches, $E \subseteq V \times V$ is a set of directed links between switches, c is a bandwidth function $c : E \rightarrow \mathbb{N}$ assigning a positive integer value $c(e)$ to every link $e \in E$, and l is a delay function $l : E \rightarrow \mathbb{N}$ assigning a positive integer value $l(e)$ to every link $e \in E$. For example, Fig. 3(a) presents an example SDN topology with six switches, $v_1, v_2, ..., v_6$, and eight directed links $e_1, e_2, ..., e_8$; and, Fig. 3(b) shows the bandwidth and delay values of each link in Fig. 3(a). The network of EMS in Fig 2 could be represented using a graph similar to that in Fig. 3(a) where every node in Fig. 3(a) corresponds to a switch in EMS and every link in Fig. 3(a) corresponds to a terrestrial or satellite link in EMS. Note that EMS terrestrial and satellite links are bidirectional and thus have to be represented as two directed graph links.

A network request q specifies a data stream that should be sent by a source switch s to a terminal switch t. Each network request q has a source switch $q.s$, a terminal switch $q.t$ and a data stream of size (or bandwidth) $q.d$. Note that $q.d$ may vary over time, but, for

notational simplicity, we capture $q.d$ as a constant. We produce a different request if $q.d$ changes and remove the old one. To process each request q, a flow f is created. A flow describes a directed path, i.e., a sequence of links, in G that is used to transmit the data stream of q. We denote by $f.q$ the request q related to a flow f, and by $f.p$ the directed path that is used to carry the data of q from $q.s$ to $q.t$. Let F be a set of flows. We denote by $links(f)$ the set of links on the directed path $f.p$ and by $links(F) = \cup_{f \in F} links(f)$ the set of all the links of the flows in F. Finally, we denote the subset of flows in F going through link e by $flows(e, F)$.

The bandwidth $c(e)$ of a network link e is a (limited) resource shared by different flows. A flow f going through a link e consumes the link's bandwidth $c(e)$ by the flow size $f.q.d$. Hence, the total size of flows going though e, i.e., the throughput of e, should be less than or equal to the bandwidth $c(e)$. Given a set F of flows, we define the throughput of e for F as follows: $throughput(e, F) = \sum_{f \in flows(e,F)} f.q.d$.

We say a network G is congested by a given set F of flows if there is some link e such that $throughput(e, F) > c(e)$. Given a network G congested by the set F of flows, we address the problem of network congestion by finding a new set $F^a = \{f_1^a, f_2^a, ..., f_n^a\}$ of flows where (1) each f_j^a processes the same request as that of the flow $f_j \in F$, i.e., $f_j^a.q = f_j.q$, and thus F^a and F have the same cardinality, i.e., $|F^a| = |F|$, and (2) G is not congested by F^a, i.e., $throughput(e, F^a) \leq c(e)$ for all $e \in links(F^a)$. The problem of resolving network congestion is NP-hard [6, 19]. Note that F^a may not exist when, for example, all the links in G are overutilized by network requests. In this case, we aim to compute F^a such that the maximum link throughput is minimized even if it is still congested (see Section 3.3).

3.2 SDN Data Forwarding

We assume that an SDN data forwarding algorithm is executed whenever a new request q arrives, i.e., the data forwarding is an event-driven (aperiodic) process. In order to handle the continuous stream of requests from network users, which are not a-priori-known, a network system must continuously respond to new requests arriving at any time – even in the middle of addressing a congestion problem. Our data forwarding algorithm, which is similar to existing baselines [17], uses weight parameters assigned to network links and computes the weighted shortest path between a pair of switches to determine the route for carrying a data stream of q between the switches. Specifically, we denote by $w(e)$ the weight value of a link e. The default weights are one (i.e., $w(e) = 1$ for all the links e in G). The weights are configurable and can be modified by application layer algorithms. In Section 3.3.2, we discuss how DICES modifies the weight parameters after detecting congestion so that the data forwarding algorithm does not send new requests through the overutilized links.

3.3 Dynamic Adaptive Congestion Control

DICES runs in parallel with the SDN data forwarding algorithm described in Section 3.2. In contrast to the SDN data forwarding algorithm, DICES is designed to execute periodically with a time period Δ. To detect congestion, DICES has to poll the network state periodically as the state is always changing due to the unpredictable environment. In addition, DICES has to ensure, when congestion happens, that the subsequent steps for congestion resolution are always deterministically executed. Therefore, we chose to design DICES as a periodic process instead of an event-driven (aperiodic) one. The period Δ should be chosen such that it is small enough to allow DICES to detect and handle congestion as quickly as possible, and at the same time, large enough for executions of DICES not to cause too much overhead and interfere with other SDN operations, e.g., the execution of the SDN data forwarding algorithm.

Let $\mathbb{T} = [0, T]$ be the time duration during which we observe the network traffic. We assume the network G is fixed over time, but the network traffic, i.e., the set Q of requests and the set F of flows handling Q, vary over time. We denote by Q_i the set of network requests received at the beginning of the time step $i{\cdot}\Delta$, and by F_i the set of flows corresponding to Q_i. At each time step $i{\cdot}\Delta$, DICES starts running by executing its "monitor" step (Fig. 1). It receives Q_i and F_i and uses these two sets in its subsequent steps, i.e., "analyze", "compute", and "apply". Requests that arrive within the interval of $[i{\cdot}\Delta, (i{+}1){\cdot}\Delta)$ or the flows generated within this interval are included in Q_{i+1} and F_{i+1}, but not in Q_i and F_i.

The "analyze" step is in charge of determining whether, or not, the network is congested. In practice, a link e is considered congested if it is utilized above a certain threshold (e.g., 80% of the link bandwidth) [2, 48]. We denote by $util(e, F_i)$ the utilization of link e by the flow set F_i and define it as follows: $util(e, F_i) = throughput(e, F_i)/c(e)$. The "analyze" step deems e to be congested if $util(e, F_i) > u$, where $0 < u \leq 1$ is the *utilization threshold*.

If the network G is congested as determined by the "analyze" step, the "compute" step addresses the congestion problem by performing the following two tasks: First, it resolves the congestion by computing a new set F_i^a of F_i that can handle the requests Q_i without congestion (see the congestion problem definition in Section 3.1). If congestion cannot be resolved, it ensures that F_i^a minimizes the maximum link utilization by Q_i. Second, it computes a set of weights for network links based on their utilization. These weights are passed to the SDN data forwarding algorithm (Section 3.2) so that the algorithm does not send new requests arriving after $i{\cdot}\Delta$ through the overutilized links. The "apply" step reconfigures flows and applies the new weights computed by the "compute" step.

In the remainder of this section, we present two algorithms addressing the two tasks of the "compute" step: A *search-based congestion control algorithm* for the first task, and a *utilization-aware weight control algorithm* for the second task.

3.3.1 Search-based Congestion Control Algorithm. Our search-based congestion control algorithm attempts to resolve an identified congestion, and if the congestion cannot be resolved, the algorithm minimizes the maximum link utilization. Specifically, given a network G congested by the set F_i of flows addressing the set Q_i of requests, our aim is to generate the set F_i^a of flows to resolve or minimize the congestion while addressing the requests in Q_i. To do so, we minimize the maximum link utilization across all the links in G (objective *O1* or *Utilization*). In addition to minimizing utilization, we aim to optimize two more objectives that are important for quality of service in network systems: We minimize the number of link updates, i.e., insertions and deletions, required

to reconfigure the network flows (objective *O2* or *Cost*) and the overall data transmission delays induced by the new set F_i^a of flows (objective *O3* or *Delay*). By minimizing the cost, we ensure that we manipulate a small number of elements at the infrastructure layer and require a small amount of time to apply F_i^a. Minimizing the network delay is critical for emergency systems to ensure that data streams are transmitted on time. Note that we have to optimize these three objectives explicitly and simultaneously since optimizing the utilization objective, *O1*, is likely to negatively impact the cost of flow reconfiguration, *O2*, or the overall delay, *O3*. This is because if the new flow paths of F_i^a are very different from those of F_i or if F_i^a uses longer but less utilized paths than those of F_i, the reconfiguration cost and the overall delay may increase. In addition, the reconfiguration cost, *O2*, and the overall delay, *O3*, are independent objectives.

Following standard practice [31], we describe our algorithm by defining the representation, the initial population, the fitness functions, and the computational search algorithm. We then discuss the output flow set F_i^a that we report as the optimal solution to be used in the "apply" step of DICES.

Representation. Given a network G and a set Q of requests, a feasible solution is a set $F = \{f_1, f_2, ..., f_l\}$ of flows where for every $f \in F$ we have $f.q \in Q$, for every $q' \in Q$ there is some $f \in F$ such that $f.q = q'$, and $|F| = |Q|$.

Initial population. Recall that the input to our search algorithm is a set F_i of flows at time $i{\cdot}\Delta$ and its corresponding set Q_i of requests. We create an initial population by randomly modifying individual flows in F_i while ensuring that the generated flow sets are able to handle the requests in Q_i.

Fitness. For the three objectives *O1*, *O2*, and *O3* described above, we formulate three quantitative fitness functions $fitUtil(F_i^a)$, $fitCost(F_i, F_i^a)$, and $fitDelay(F_i^a)$, respectively, where F_i is the set of flows given as input and F_i^a is a candidate flow set generated during the search.

The $fitUtil(F_i^a)$ fitness function is defined by equation (1) as the maximum link utilization across all the links used in F_i^a. Our approach aims to minimize equation (1).

$$fitUtil(F_i^a) = \max_{e \in links(F_i^a)} util(e, F_i^a) \tag{1}$$

The $fitCost(F_i, F_i^a)$ fitness function is defined by equation (2). In this paper, we compute the distance between a pair f and f' of flows, denoted by $dist(f, f')$, as the *edit distance* between the path of f ($f.p$) and the path of f' ($f'.p$). Our notion of edit distance is the same as computing the longest common subsequence (LCS) distance of two paths [27] and counts the number of *link insertions* and *link deletions* required to transform $f.p$ into $f'.p$. This metric matches our definition of the cost objective described earlier in this section. Our approach minimizes equation (2).

$$fitCost(F_i, F_i^a) = \sum_{(f,f') \in F_i \times F_i^a : f.q = f'.q} dist(f, f') \tag{2}$$

The $fitDelay(F_i^a)$ fitness function is defined by equation (3) which sums the delay values $l(e)$ of all the links e used in a candidate solution F_i^a. Note that the delay objective can be estimated for a flow set F_i^a only if F_i^a does not give rise to congestion, i.e., only when $fitUtil(F_i^a) \leq u$, where u is a utilization threshold. This is because, when a network is congested, actual delay values depend on various factors such as the underlying network protocol (e.g., TCP or UDP) that are not studied here. Hence, when F_i^a leads to congestion, we assign an undefined value (i.e., a large number) to $fitDelay(F_i^a)$. Our approach minimizes equation (3).

$$fitDelay(F_i^a) = \begin{cases} \sum_{e \in links(F_i^a)} l(e) & \text{if } fitUtil(F_i^a) \leq u \\ \text{UNDEF} & \text{otherwise} \end{cases} \tag{3}$$

Recall from Section 3.1, that congestion may not be resolved by our approach which is based on reassigning the flows. In this case, the objective *fitDelay*() is excluded since it is undefined and returns a large number for all the congested solutions. But the search still minimizes *fitUtil*() and *fitCost*() and returns a solution F_i^a that is minimally congested and its implementation incurs minimal cost.

Computational search. We use the Non-dominated Sorting Genetic Algorithm version 2 (NSGAII) algorithm [28] to find a near-optimal solution. NSGAII outputs a set (Pareto front) of non-dominated solutions which are equally viable and the best tradeoffs found among the given fitness functions. The dominance relation over solutions is defined as follows [44]: "A solution F_i^b dominates another solution F_i^a if F_i^b is not worse than F_i^a in all fitness values, and F_i^b is strictly better than F_i^a in at least one fitness."

```
1  Algorithm Search-based congestion control
2  Input G: Network
3  Input Q_i: Set of requests at time i·Δ
4  Input F_i: Set of flows at time i·Δ
5  Input u: Upper threshold of link utilization
6  Input psize: population and archive size
7  Input cprob: Crossover probability
8  Input mprob: Mutation probability
9  Input neval: Maximum number of evaluations
10 Output F_i^b: Best solution
11
12 // initial population
13 P ← {F_i} // P is a set of sets
14 while |P| < psize do
15 |  F_i^a ← mutate(G, F_i)
16 |  P ← P ∪ {F_i^a}
17 A ← {} // initial archive
18 for neval times do
19 |  // fitness evaluation
20 |  for each F_ik^a ∈ P do
21 |  |  fitUtil(F_ik^a) = max_{e∈links(F_ik^a)} util(e, F_ik^a)
22 |  |  fitCost(F_i, F_ik^a) = Σ_{(f,f')∈F_i×F_ik^a : f.q=f'.q} dist(f, f')
23 |  |  if fitUtil(F_ik^a) ≤ u then
24 |  |  |  fitDelay(F_ik^a) = Σ_{e∈links(F_ik^a)} l(e)
25 |  |  else
26 |  |  |  fitDelay(F_ik^a) = UNDEF
27 |  P ← P ∪ A
28 |  B ← paretoFront(P)
29 |  R⃗ ← sortNonDominatedFronts(P)
30 |  A ← {}
31 |  for each front R_k in R⃗ do
32 |  |  assignCrowdingDistance(R_k)
33 |  |  // union() returns |A| ≤ psize
34 |  |  A ← union(A, R_k, psize)
35 |  |  if |A| = psize then break
36 |  P ← breed(A, cprob, mprob)
37 F_i^b ← selectOne(B)
38 return F_i^b
```

Figure 4: An NSGAII-based congestion control algorithm.

```
1  Algorithm Flow mutation
2  Input G: Network
3  Input F_i^a: Set of flows
4  Input mprob: Mutation probability
5  Output F_i^m: Set of flows
6
7  F_i^m ← F_i^a
8  for each f_k ∈ F_i^a do
9  |  if mprob ≥ random(0,1) then
10 |  |  f_k^a ← alternativeFlow(G, f_k)
11 |  |  F_i^m ← (F_i^m \ {f_k}) ∪ {f_k^a}
12 return F_i^m
```

Figure 5: A flow mutation algorithm.

Fig. 4 presents our NSGAII-based congestion control algorithm. As shown in lines 12–16, we first create an initial population based on the input F_i. Lines 19–26 of the algorithm compute the fitness functions. Lines 27–36 describe how NSGAII selects best solutions (lines 27–28), sorts non-dominated fronts (line 29), and assigns crowding distance (line 32) to introduce diversity among non-dominated solutions [28].

As per line 36 of the listing in Fig. 4, the algorithm breeds the next population by using the following genetic operators: (1) *Selection.* We use the binary tournament selection based on non-domination ranking and crowding distance as typically used by NSGAII [28]. (2) *Crossover.* We use the standard single-point crossover which has been applied in many problems [8, 28, 37]. (3) *Mutation.* We use the mutation algorithm in Fig. 5. It replaces a randomly selected flow f_k in F_i^a (lines 8–9) with an alternative flow f_k^a for f_k such that $f_k^a.q = f_k.q$ (lines 10–11).

Choosing an optimal solution. The output of NSGAII is a set of equally viable solutions (line 28 in Fig. 4). But we have to select only one solution to be used for reconfiguring the network (lines 37–38). Researchers have proposed various alternatives for selecting an optimal solution among all the solutions on an optimal Pareto front, such as a *knee* solution [20] or the *corner* solution [56] for an objective. In our work, we use a knee solution. Fig. 6 illustrates the knee point on a two-objective Pareto front. Specifically, a knee point is the closest point on a Pareto front to the *ideal point*, a hypothetical solution with the best values for all the objectives. A knee solution is often preferred in SBSE studies [20, 23] because a small improvement in one objective by selecting other solutions on the front would lead to a large deterioration in at least one other objective [20]. In our work, we choose a knee solution because we do not want the selected flow set to be biased toward any objective; instead, we prefer a flow set that is equally optimized for all the objectives.

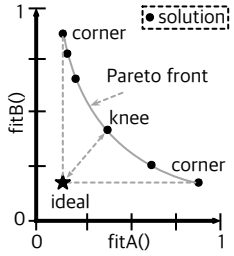


Figure 6: The concept of a knee point solution on a two-objective Pareto front.

3.3.2 Utilization-aware Weight Control. As described in Section 3.2, the SDN data forwarding algorithm constructs a flow that always routes a data stream along the weighted shortest path. A network system that uses static link weights, e.g., $w(e) = 1$ for all the links e in G, may remain congested even after applying our search-based solution in Section 3.3.1. This occurs when a highly utilized link keeps being used by the SDN data forwarding algorithm to carry new data requests because the link is located on a weighted shortest path. Note that, in general, links located on the shortest paths are more likely to be highly utilized or congested when we use the SDN data forwarding algorithm with fixed weight values. To avoid this problem, we update link weights in a way that forces the SDN data forwarding algorithm to prioritize less utilized links over highly utilized ones.

```
1  Algorithm Link weight adjustment
2  Input F_i^b: Solution of the algorithm in Fig. 4
3  Input w⃗_i: Vector of link weights at time i·Δ
4  Input u: Upper threshold of link utilization
5  Output w⃗^o: Vector of adjusted link weights
6
7  w⃗^o ← w⃗_i
8  for each e ∈ links(F_i^b) do
9  |  w(e) = l(e)·u/(u − util(e, F_i^b))
10 |  w⃗^o ← replaceWeight(w⃗^o, e, w(e))
11 return w⃗^o
```

Figure 7: A link weight adjustment algorithm.

Fig. 7 describes our utilization-aware link weight adjustment algorithm. The algorithm modifies the link weights based on the statement on line 9. Specifically, it adjusts the weights such that $w(e)$ of a link e is proportional to link delay $l(e)$, but is inversely proportional to the remaining (available) bandwidth of the link e, i.e., $u/(u - util(e, F_i^b))$. Note that the bandwidth is computed after applying the new optimized flows F_i^b generated by the search-based algorithm in Fig. 4. The weight computation thus assigns a large value to a highly utilized but lower-speed link (i.e., a link with large delay). The SDN data forwarding algorithm (see Section 3.2) then selects less utilized and higher-speed links when it creates flows to address new requests arriving after the weight adjustment, i.e., after $i \cdot \Delta$.

4 EMPIRICAL EVALUATION

In this section, we present an evaluation of DICES. Our full evaluation package is available online [64].

4.1 Research Questions (RQs)

RQ1 (efficiency and effectiveness): *Can our approach resolve congestion caused by changes in network requests over time?* In RQ1, we examine the efficiency and effectiveness of DICES by investigating whether it is able to detect congestion as we increase network requests, and whether it can compute and apply an adequate reconfiguration in a timely manner.

RQ2 (scalability): *Can our approach resolve congestion promptly for large-scale networks?* In RQ2, we investigate the scalability of DICES by studying the relation between its execution time and the network size and number of requests.

RQ3 (comparison with baselines): *How does our approach perform compared with baseline approaches?* With RQ3, we investigate whether our approach can outperform two existing packet forwarding algorithms: a reactive forwarding algorithm (RFWD) [17] and an open shortest path first algorithm (OSPF) [26]. RFWD and OSPF, discussed in Section 4.5, are commonly used for optimal data forwarding and congestion avoidance, respectively, and as baselines in recent SDN research strands [1, 5, 16, 17, 21, 58, 62].

4.2 Simulation Platform

We implemented DICES as an application for an SDN testbed at SES. Specifically, we use an open-source SDN control platform known as ONOS (Open Network Operating System) [13]. ONOS has been used extensively in research and practice [11, 17], in particular for large-scale network systems. To simulate networks, we use Mininet [47] and D-ITG [18]. Mininet is a network emulator that creates a virtual network, running real SDN-switch and application programs, on a single machine to ease prototyping and testing. D-ITG (Distributed Internet Traffic Generator) is a traffic generation and monitoring tool that supports various network protocols and traffic distributions for replicating realistic network traffic. We ran all our experiments on a computer equipped with an Intel i7 CPU with 8GB of memory.

4.3 Study Subjects

We use two types of study subjects: (1) some synthetic networks, and (2) EMS – a large-scale industrial system under study by SES (see Section 2). The synthetic networks are used to evaluate efficiency, effectiveness and scalability since, in these networks, we can freely change the size and traffic, while EMS is used to evaluate the execution time of DICES and to compare it with baselines in a realistic setting.

Our synthetic networks are characterized by two parameters: the number of network switches and the number of network requests. We assume complete graph topologies, i.e., all the switches are connected to one another using links with 100Mbps bandwidth and 25ms delay. Hence, the number of links is not an open parameter for our synthetic networks. This choice was made to reduce unnecessary complexity in our analysis. The network bandwidth and delay values and network-request profiles were suggested by SES based on the typical characteristics of terrestrial links and their data streams. We use UDP and TCP – typical protocols for Internet applications – for transmitting data. Note that graph topologies, which contain only one path from a sender to receiver, e.g., star and line topologies, are not considered in our experiments. Congestion for such topologies can only be handled by decreasing the network traffic. In contrast, DICES requires multiple flow paths from a sender to a receiver for rerouting. A partially complete graph topology is considered in the EMS network (Fig. 2).

As discussed in Section 2, the EMS network contains seven SDN switches, four site types (denoted RM, MC, CS, and GS in Fig. 2). The network further contains both terrestrial and satellite links and is connected to *external (legacy) networks (EN)*. The characteristics of the terrestrial and satellite links are as follows: 100 Mbps bandwidth, 25ms delay for terrestrial links, and 10 Mbps bandwidth, 275ms delay for satellite links.

4.4 Evaluation Metrics

To answer the RQs, we measure the following network performance metrics: *link utilization*, *packet loss*, and *packet delay*. In addition, we measure the execution time of DICES. The link utilization metric is the maximum link utilization across all the links in a network since a single overutilized link can create congestion. Specifically, given a set F of flows, we compute this metric as the maximum of $util(e, F)$ for every link e (see Section 3.3 for the definition of $util(e, F)$).

Table 1: A traffic profile for a disaster situation.

EMS entity		Request characteristics			
Sender	Receiver	Type	Protocol	Throughput	# requests
RM	MC	Sensor	TCP	100 Kbps	5
CS	MC	Audio	UDP	64 Kbps	4
CS	MC	Video	UDP	10 Mbps	2
MC	CS	Audio	UDP	64 Kbps	4
MC	CS	Video	UDP	10 Mbps	2
MC	CS	Map	TCP	30 Mbps	1
EN^N	EN^D	External	UDP	20 Mbps	5
EN^D	EN^N	External	UDP	20 Mbps	5

The EMS entities are: RM (*remote monitoring site*), MC (*emergency monitoring center*), CS (*mobile communication facility site*), EN^N (*external networks* in normal areas), and EN^D (*external network* in a disaster area). The disaster area (D) is assumed to be close to s1 in the network of Fig. 2. Other than s1, all switches in this network are in normal areas (N).

To measure the packet loss and delay metrics, we rely on an existing network monitoring tool, D-ITG, described in Section 4.2. Briefly, the packet loss metric for a flow measures the number of packets dropped associated with the flow over a time period, e.g., time interval Δ. The delay metric for a flow measures each individual packet delivery time from the sender to the receiver of the flow.

Due to random variation in DICES and the traffic generator (D-ITG), we repeat our experiments 50 times. To statistically compare our results, we use Mann-Whitney U-test [50] which determines whether two independent samples are likely or not to belong to the same distribution. We set the level of significance, α, to 0.05.

To determine correlations between the execution time of DICES and the network parameters in our study, i.e., network size and the number of requests, we use regression analysis [54]. We use R^2 [70] to evaluate the goodness of fit for our regression analysis, providing the proportion of the variance in execution time that can be explained by a regression model.

4.5 Experimental Setup

EXP1. To answer RQ1, we create a synthetic network with five switches and generate two network requests between a fixed pair of switches every 10s. Each request transmits 30Mbps of data. Hence, the volume of data that the network transmits increases over time, i.e., 60Mbps initially, 120Mbps after 10s, 180Mbps after the next 10s, and so on.

EXP2. To answer RQ2, we perform two analyses: (1) To study the correlation between the execution time of DICES and the network size, we create ten synthetic networks with 5, 10, …, 50 SDN switches, and for each network, we generate five requests simultaneously to transmit a total of 150Mbps data. (2) To study the correlation between the execution time of DICES and the number of requests, we use a network with five switches, and perform ten different experiments by issuing 5, 10, …, 50 requests simultaneously to transmit, each time, a total of 150Mbps. We also compute the execution time of DICES over the EMS network.

EXP3. To answer RQ3, we use the data traffic profile shown in Table 1 and defined by SES for the EMS network of Fig. 2. The profile characterizes anticipated traffic at the time of a disaster. It includes 28 requests which transmit sensor, audio, video, map, and external data where TCP is used for sensor and map data and UDP for the rest. In this experiment, we assume a disaster occurs in the

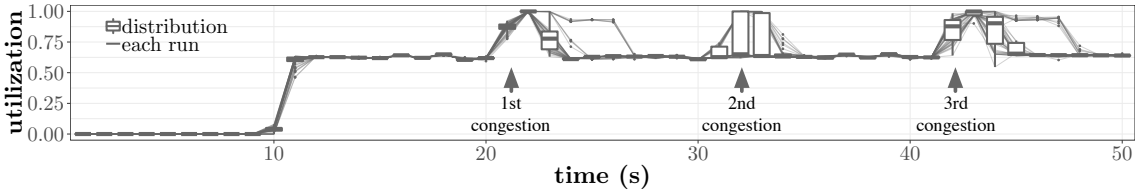


Figure 8: Network utilization values over time when DICES is used to resolve congestion for a synthetic network with five switches and when new UDP requests arrive around 10s, 20s, 30s and 40s. The boxplots (25%-50%-75%) show network utilization values obtained based on 50 executions of DICES.

area near the s1 switch in Fig. 2. Thus, the *mobile communication facility site* is connected to s1.

We compare DICES with a reactive forwarding algorithm (RFWD) [17] and an open shortest path first forwarding algorithm (OSPF) [26]. RFWD, which is the only predefined reactive data forwarding application in ONOS, routes requests through the shortest paths between the requests' ending points. It is the same as the SDN data forwarding algorithm in Section 3.2 when link weights are all equal to one and are fixed all the time. OSPF is commonly used as a baseline for SDN-based network solutions [1, 5, 21, 58, 62]. It is a prevalent protocol in legacy (non-SDN) networks while SDN is still an emerging area in both research and practice. OSPF computes weighted shortest paths to route network requests, but it does not provide the flexibility to update the link weights dynamically. We compare DICES with OSPF when the link weights for OSPF are inversely proportional to the bandwidths of the links. This is a typical use case of OSPF and can reduce the possibility of congestion since high-bandwidth links tend to be more used to carry data. To our knowledge, DICES is the first SDN application available online which addresses a congestion problem while accounting for minimizing multiple objectives: transmission delays and reconfiguration costs [64].

4.6 Parameter Tuning and Setting

We set Δ, i.e., the time period for executing DICES, to 1s since this is the minimum monitoring time period allowed by ONOS. Following the guidelines in the literature [9], we set the NSGAII parameters as follows: the population size = 100, the crossover probability = 0.8, and the mutation probability = $1/|F_i|$. We set the utilization threshold to 0.8, as instructed by SES. We set the total number of fitness evaluations to 10,000 because our initial experiments, performed on EMS, showed that, after 10,000 fitness evaluations, there is no notable improvement in the optimal solution.

4.7 Experiment Results

RQ1. Fig. 8 shows the network utilization over time when DICES is used to resolve congestion for the synthetic network described in **EXP1** (see Section 4.5). As shown in the figure, network requests cause congestion after 20s, 30s and 40s. Note that the requests arriving around 10s do not lead to any congestion and they can be handled by the network. DICES is able to resolve every congestion since utilization always comes back down to around 65% after the sudden increase caused by each congestion. DICES is further able to resolve congestion in a timely manner. Specifically, it takes DICES, on average, 439ms to execute all the four steps in its control loop. We note that it takes, on average, 2.68s for the network utilization to settle back to a desired value below the utilization threshold (i.e., 0.8) after congestion. This is due to the additional internal processing time required by ONOS to monitor the network and reconfigure the SDN control and infrastructure layers.

As suggested by its low utilization average in Fig. 8, the second occurrence of congestion around 30s is observed only in 17 out of 50 runs of DICES. More precisely, in the other 33 runs, the link weight adjustment performed by DICES at 20s is able to handle the requests at 30s without leading to any congestion. This is because the link weights adjusted by DICES at 20s can sometimes, due to luck, help the SDN data forwarding algorithm (Section 3.2) handle the requests arriving at 30s using less utilized links, hence preempting congestion. Note that Fig. 8 shows the results for UDP packet transmission. The results for TCP packet transmission are consistent with those in Fig. 8 and not shown due to space.

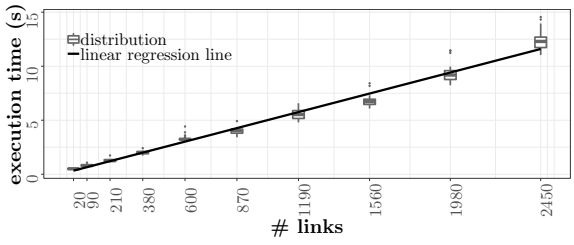
The answer to **RQ1** *is that* DICES efficiently and effectively resolves congestion. In particular, experiments performed on a realistic network transmitting large and increasing volumes of data over time show that DICES is able to maintain, most of the time, the network utilization at 65%, which is well below the utilization threshold of 80%. Further, DICES takes, on average, 439ms to execute and resolve congestion.

RQ2. Fig. 9 reports the results obtained by **EXP2**. Specifically, Fig. 9(a) shows the relation between the execution time of DICES versus network size specified as the number of links (i.e., the first study of **EXP2**), and Fig. 9(b) shows the relation between the execution time of DICES versus the number of requests (i.e., the second study of **EXP2**). Note that the x-axis of Fig. 9(a) shows the number of links instead of the number of switches since DICES mainly manipulates links, and its execution time depends on the number of links and not the number of switches (see the algorithm in Fig. 4). The linear regression lines in both Fig. 9(a) and Fig. 9(b) fit well the actual execution time of DICES with high goodness of fit (i.e., R^2=0.98 for Fig. 9(a) and R^2=0.89 for Fig. 9(b)). Hence, the execution time of DICES is linear both in the number of links and in the number of requests. Therefore, we expect DICES to scale well as the numbers of network links and requests increase. Finally, for our industrial EMS, which contains seven switches and 30 links, and has to handle 28 requests (see Section 4.5), DICES, on average, takes 1.74s to resolve congestion. This shows that DICES is able to scale to real-world systems and can resolve congestion caused by high network demands due to an emergency.

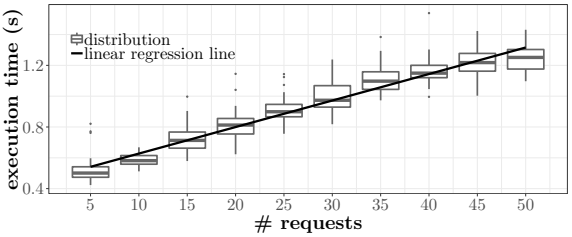
We note that our analysis above is concerned with the relation between the execution time of DICES and the number of requests, rather than the data size of network requests. Since DICES resolves

Table 2: Comparison of the average delay and packet loss of DICES against RFWD and OSPF based on 50 runs of each algorithm.

		RFWD		OSPF		DICES	
Receiver	Type	Delay (s) [p]	Packet loss (%) [p]	Delay (s) [p]	Packet loss (%) [p]	Delay (s)	Packet loss (%)
Emergency monitoring center	Sensor	0.10 [0.00]	0.00 [1.00]	0.09 [0.00]	0.00 [1.00]	0.13	0.00
	Audio	0.03 [0.00]	0.17 [0.00]	0.03 [0.49]	0.20 [0.02]	0.07	0.23
	Video	0.03 [0.00]	0.02 [0.03]	0.03 [0.30]	0.02 [0.02]	0.04	0.08
Mobile communication facility site	Audio	0.03 [0.00]	0.17 [0.01]	0.06 [0.02]	0.38 [0.00]	0.08	0.18
	Video	0.03 [0.00]	0.06 [0.53]	0.06 [0.00]	0.32 [0.00]	0.04	0.27
	Map	0.23 [0.00]	0.00 [1.00]	**1.13 [0.00]**	0.00 [1.00]	**0.10**	0.00
External network	External	0.35 [0.00]	**29.55 [0.00]**	0.05 [0.00]	0.14 [0.00]	0.06	**0.02**
Weighted average	Overall	0.29	21.81	0.17	0.13	0.06	0.04



(a) Links



(b) Requests

Figure 9: Graphs showing DICES execution time versus (a) the number of network links, and (b) the number of network requests together with regression lines showing linear correlation between DICES execution time and (a) the number of links (*exec.time*=2.391*e*-01+4.635*e*-03×|*links*|), and (b) the number of requests (*exec.time*=0.455+0.017×|*requests*|).

congestion by rerouting requests and never modifies the data size of a request, the execution time of DICES is not impacted by data size. We have confirmed this through experiments that we cannot report due to space.

The answer to **RQ2** *is that* the execution time of DICES is linear in the network size and in the number of requests. Further, DICES scales to real-world systems: it takes an average of 1.74s to resolve congestion caused by an emergency situation in our industrial case study.

RQ3. Table 2 shows the average delay and packet loss values for EMS when one uses DICES, RFWD and OSPF for handling the requests described in Table 1. As discussed in Section 4.5, each experiment was repeated 50 times. The table statistically compares DICES against RFWD and OSPF with respect to delay and packet loss by reporting p-values.

In the table, we have highlighted in gray two specific delay values of DICES and OSPF, and two specific packet loss values of DICES and RFWD. These two pairs are particularly interesting because they show significant differences between DICES and OSPF in terms of delay and between DICES and RFWD in terms of packet loss (p-value < 0.05 in both cases). These differences are significant, not just statistically but also practically. Specifically, the difference in delay values shows that the EMS network with OSPF transmits map data with a 1.13s delay on average. In contrast, with DICES, the network transmits the map data with a 0.1s delay on average. In other words, for map-data transmission, the network with DICES is 11 times faster than the network with OSPF. The difference in packet loss values shows that the EMS network with RFWD drops, on average, 29.55% packets while exchanging data between external networks (see Table. 1). When DICES is used, the network drops only 0.02% of those packets on average. This shows that DICES is considerably more effective than RFWD in transmitting external data through the EMS network during an emergency situation. We note that the behavior of DICES is independent of traffic types – sensor, audio, video, map, and external. As shown in Table 2, DICES maintains a practically acceptable level of delay and packet loss for all types of traffic even when the EMS network is congested. In contrast, the two baselines, i.e., RFWD and OSPF, fail to maintain the level of delay and packet loss at an acceptable level.

To compare the overall performance of RFWD and OSPF with DICES, we compute weighted averages of delay and packet loss. Specifically, the weighted average delay (resp., packet loss) for each algorithm is computed by multiplying the average delay (resp., packet loss) of that algorithm for each network request type with the total throughput of that request type (see the request types and throughputs in Table. 1). The weighted averages, given in the last row of Table 2, show that DICES yields lower overall delay and packet loss compared to both RFWD and OSPF. That is, the overall delay of DICES is almost five and three times better than the overall delays of RFWD and OSPF, respectively. Further, DICES loses almost 99% and 70% less packets compared to RFWD and OSPF, respectively.

We note that the improvements brought about by DICES come at the expense of reconfiguring some flows to resolve congestion, while RFWD and OSPF do not require any reconfiguration. SES found the minimized reconfiguration cost of DICES to be an acceptable tradeoff for the substantial benefits of the approach over RFWD and OSPF in terms of delay and packet loss.

The answer to **RQ3** *is that* DICES significantly outperforms the baseline algorithms: RFWD and OSPF. Specifically, results obtained by simulating emergency traffics over the EMS network show that the overall network delay of DICES is almost five and three times better than those of RFWD and OSPF, respectively. Further, DICES loses almost 99% and 70% less packets compared to RFWD and OSPF, respectively.

4.8 Threats to Validity

We evaluated DICES using both synthetic networks and an industrial IoT system. Since our current evaluation uses a network emulator (Mininet), future case studies and experiments on physical networks remain necessary for a more conclusive evaluation of DICES. In particular, there is the possibility that the physical network in our industrial case study system may sustain damage during natural disasters. DICES can operate properly as long as the underlying SDN provides accurate topology and traffic data. How accurate this data would be in the presence of network damage, and how one can counteract potential inaccuracies need to be further investigated. In addition, while motivated by IoT-enabled emergency management systems, DICES is a general congestion-control approach for SDN. Case studies in other domains, e.g., SDN-based data centers, are required in order to assess the usefulness of DICES in a broader context.

5 RELATED WORK

This section compares DICES with related work in the areas of communication protocols, SDN, IoT, self-adaptive systems and dynamic adaptive SBSE.

Standard communication protocols have been widely studied for resolving network congestion [4, 14, 30, 36, 51]. For example, the TCP congestion control algorithm is prevalently used over the Internet and has been addressed by many prior research threads [4, 30, 36, 51]. More recent work in this direction includes new application-layer protocols such as CoAP [63] and its congestion control algorithm, CoCoA [14]. These congestion control algorithms, in general, work by adjusting data transmission rates in an interconnected set of network hosts. In contrast, DICES works by controlling the data flow paths and link weights in a network.

SDN has received considerable attention in the recent literature on networks. The problem of flow reconfiguration has been already studied for SDN with the objective of exploiting the additional flexibility offered by software [1, 19, 24, 32, 38, 39, 42, 68]. Chiang et al. [24] formulate a new optimization problem to find optimal routing paths for group communication traffic. Gay et al. [32] propose a local search-based segment routing method for networks with unexpected failures. Huang et al. [39] present a dynamic routing algorithm to maximize network throughput under link-capacity and user-demand constraints. Agarwal et al. [1] present a single-objective linear programming method for improving network utilization. None of the above work strands account for the tradeoffs among the three objectives that DICES minimizes, i.e., maximum link utilization, number of link configurations, and delay. Further, unlike the above, DICES supports simultaneous dynamic control of data flow paths and link weights to both deal with the current congestion and also plan for handling future requests in a congestion-free manner. Finally, DICES is evaluated through a real case study on an emergency management system.

IoT may be realized through a variety of technologies and applied in many application domains [3, 66]. The research topics related to IoT are numerous, e.g., data models to capture highly volatile IoT data [53], model-based code generation for heterogeneous things [35], model-based testing of IoT communications [67], IoT architectures [45, 57], and self-adaptive IoT systems [12, 22, 29, 40, 55, 69]. Among these, an architecture-based adaptation framework by Weyns et al. [69] is the most related to our work. This prior work accounts for multiple quality of service criteria and represents them as quality constraints using state machines. In contrast, we formulate quality objectives as quantitative functions, thus enabling Pareto (tradeoff) analysis. Further, we use a multi-objective search algorithm to find practically acceptable solutions for dynamically reconfiguring an IoT system.

Self-adaptive systems have been studied in many domains [25, 46]. DICES relates to work on self-adaptation in the network domain, e.g., adaptive network anomaly detection [41], adaptive network monitoring [7, 15], self-adaptive multiplex networking [59], and network topology adaptation [65]. Among these, the most pertinent thread is by Stein et al. [65], where the authors propose a topology adaptation model alongside a language to specify the adaptation logic of a set of network applications. This prior work aims to adapt a topology to a set of network applications, e.g., a video streaming source and a peer. In contrast, DICES adapts the network upon which the applications rely. Further, DICES uses multi-objective search to account for optimization tradeoffs.

Dynamic adaptive SBSE [34], as noted in Section 1, is the main research field upon whose principles we build. Prior research in this field has employed search for various purposes, e.g., improving the design and architecture of self-adaptive systems [8, 52, 60] and configuring such systems [61, 71]. To our knowledge, we are the first to have addressed the problem of congestion control in the context of dynamic adaptive SBSE.

6 CONCLUSIONS

We developed a search-based approach, named DICES, to dynamically mitigate network congestion in IoT systems via network reconfiguration. Our approach is realized through a control feedback loop, whereby the traffic on an IoT network is periodically monitored and corrective action is taken at run-time when congestion is detected. The corrective action to take (i.e., the reconfiguration) is computed using a multi-objective search algorithm that simultaneously minimizes: (1) the maximum link utilization across all the links in the network, (2) the number of link updates for reconfiguration, and (3) the overall data transmission delays. We evaluated DICES on a number of synthetic networks as well as an industrial IoT-enabled emergency management system. The results indicate that DICES is able to efficiently and effectively adapt an IoT network to resolve congestion. Further, compared to two common data forwarding algorithms which we use as baselines, DICES yields data transmission rates that are at least 3 times faster while reducing data loss by at least 70%.

For future work, we plan to extend DICES by accounting for: (1) link and switch failures and (2) the policies (e.g., cost-containment policies) that govern the use of terrestrial and satellite telecommunication networks. When dealing with fast-changing network loads (i.e., when Q_{i+1} is drastically different from Q_i), the flows computed by DICES to optimize network usage based on Q_i may not be optimized for Q_{i+1}. We intend to address this issue in future by using prediction models to anticipate Q_{i+1} earlier and use it in the computation of the optimized flows for the next step. In the longer term, we would like to further validate DICES by deploying it as an integrated component of the emergency management system in our industrial case study (in-situ deployment).

ACKNOWLEDGMENTS

This project has received funding from SES, the Luxembourg National Research Fund under the grant C-16PPP/IS/11270448 and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277).

REFERENCES

[1] Sugam Agarwal, Murali S. Kodialam, and T. V. Lakshman. 2013. Traffic Engineering in Software Defined Networks. In *Proceedings of the 2013 Annual IEEE International Conference on Computer Communications (INFOCOM'13)*. 2211–2219.

[2] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. 2014. A Roadmap for Traffic Engineering in SDN-OpenFlow Networks. *Computer Networks* 71 (2014), 1–30.

[3] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys Tutorials* 17, 4 (2015), 2347–2376.

[4] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the 2010 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'10)*. 63–74.

[5] Rashid Amin, Martin Reisslein, and Nadir Shah. 2018. Hybrid SDN Networks: A Survey of Existing Approaches. *IEEE Communications Surveys and Tutorials* 20, 4 (2018), 3259–3306.

[6] Saeed Akhoondian Amiri, Szymon Dudycz, Stefan Schmid, and Sebastian Wiederrecht. 2018. Congestion-Free Rerouting of Flows on DAGs. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP'18)*. 143:1–143:13.

[7] Ivan Dario Paez Anaya, Viliam Simko, Johann Bourcier, Noël Plouzeau, and Jean-Marc Jézéquel. 2014. A Prediction-driven Adaptation Approach for Self-Adaptive Sensor Networks. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'14)*. 145–154.

[8] Sandro S. Andrade and Raimundo José de A. Macêdo. 2013. A Search-Based Approach for Architectural Design of Feedback Control Concerns in Self-Adaptive Systems. In *Proceedings of the 7th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'13)*. 61–70.

[9] Andrea Arcuri and Gordon Fraser. 2011. On Parameter Tuning in Search Based Software Engineering. In *Proceedings of the 3rd International Conference on Search Based Software Engineering (SSBSE'11)*. 33–47.

[10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805.

[11] Taimur Bakhshi. 2017. State of the Art and Recent Research Advances in Software Defined Networking. *Wireless Communications and Mobile Computing* 2017 (2017), 1–35.

[12] Jacob Beal, Mirko Viroli, Danilo Pianini, and Ferruccio Damiani. 2017. Self-Adaptation to Device Distribution in the Internet of Things. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 12, 3 (2017), 12:1–12:29.

[13] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantzand Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. 2014. ONOS: Towards an Open, Distributed SDN OS. In *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN'14)*. 1–6.

[14] August Betzler, Carles Gomez, Ilker Demirkol, and Josep Paradells. 2016. CoAP Congestion Control for the Internet of Things. *IEEE Communications Magazine* 54, 7 (2016), 154–160.

[15] Md Zakirul Alam Bhuiyan, Jie Wu, Guojun Wang, Tian Wang, and Mohammad Mehedi Hassan. 2017. e-Sampling: Event-Sensitive Autonomous Adaptive Sensing and Low-Cost Monitoring in Networked Sensing Systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 12, 1 (2017), 1:1–1:29.

[16] Andrea Bianco, Paolo Giaccone, Ahsan Mahmood, Mario Ullio, and Vinicio Vercellone. 2015. Evaluating the SDN control traffic in large ISP networks. In *Proceedings of the 2015 IEEE International Conference on Communications (ICC'15)*. 5248–5253.

[17] Andrea Bianco, Paolo Giaccone, Reza Mashayekhi, Mario Ullio, and Vinicio Vercellone. 2017. Scalability of ONOS reactive forwarding applications in ISP networks. *Computer Communications* 102 (2017), 130–138.

[18] Alessio Botta, Alberto Dainotti, and Antonio Pescapè. 2012. A Tool for The Generation of Realistic Network Workload for Emerging Networking Scenarios. *Computer Networks* 56, 15 (2012), 3531–3547.

[19] Sebastian Brandt, Klaus-Tycho Foerster, and Roger Wattenhofer. 2016. On Consistent Migration of Flows in SDNs. In *Proceedings of the 2016 Annual IEEE International Conference on Computer Communications (INFOCOM'16)*. 1–9.

[20] Jürgen Branke, Kalyanmoy Deb, Henning Dierolf, and Matthias Osswald. 2004. Finding Knees in Multi-objective Optimization. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN'04)*. 722–731.

[21] Marcel Caria, Tamal Das, and Admela Jukan. 2015. Divide and conquer: Partitioning OSPF networks with SDN. In *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM'15)*. 467–474.

[22] Ing-Ray Chen, Jia Guo, and Fenye Bao. 2016. Trust Management for SOA-Based IoT and Its Application to Service Composition. *IEEE Transactions on Services Computing* 9, 3 (2016), 482–495.

[23] Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature-Guided and Knee-Driven Multi-Objective Optimization for Self-Adaptive Software. *ACM Transactions on Software Engineering and Methodology (TOSEM'18)* 27, 2 (2018), 5:1–5:50.

[24] Sheng-Hao Chiang, Jian-Jhih Kuo, Shan-Hsiang Shen, De-Nian Yang, and Wen-Tsuen Chen. 2018. Online Multicast Traffic Engineering for Software-Defined Networks. In *Proceedings of the 2018 Annual IEEE International Conference on Computer Communications (INFOCOM'18)*. 414–422.

[25] Zack Coker, David Garlan, and Claire Le Goues. 2015. SASS: Self-adaptation Using Stochastic Search. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'15)*. 168–174.

[26] Rob Coltun, Dennis Ferguson, John Moy, and Acee Lindem. 2008. *OSPF for IPv6*. Internet Standard RFC 5340. Network Working Group.

[27] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). The MIT Press.

[28] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[29] Nathalia Moraes do Nascimento and Carlos José Pereira de Lucena. 2017. FIoT: An Agent-Based Framework for Self-Adaptive and Self-Organizing Applications based on the Internet of Things. *Information Sciences* 378 (2017), 161–176.

[30] Simone Ferlin, Özgü Alay, Thomas Dreibholz, David A. Hayes, and Michael Welzl. 2016. Revisiting Congestion Control for Multipath TCP with Shared Bottleneck Detection. In *Proceedings of the 2016 Annual IEEE International Conference on Computer Communications (INFOCOM'16)*. 1–9.

[31] Filomena Ferrucci, Mark Harman, Jian Ren, and Federica Sarro. 2013. Not Going to Take This Anymore: Multi-objective Overtime Planning for Software Engineering Projects. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 462–471.

[32] Steven Gay, Renaud Hartert, and Stefano Vissicchio. 2017. Expect the Unexpected: Sub-Second Optimization for Segment Routing. In *Proceedings of the 2017 Annual IEEE International Conference on Computer Communications (INFOCOM'17)*. 1–9.

[33] Evangelos Haleplidis, Kostas Pentikousis, Spyros G. Denazis, Jamal Hadi Salim, David Meyer, and Odysseas G. Koufopavlou. 2015. *Software-Defined Networking (SDN): Layers and Architecture Terminology*. Information RFC 7426. Internet Research Task Force (IRTF).

[34] Mark Harman, Edmund Burke, John Clark, and Xin Yao. 2012. Dynamic Adaptive Search Based Software Engineering. In *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'12)*. 1–8.

[35] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. 2016. ThingML: A Language and Code Generation Framework for Heterogeneous Targets. In *Proceedings of the 19th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS'16)*. 125–135.

[36] Keqiang He, Eric Rozner, Kanak Agarwal, Yu (Jason) Gu, Wes Felter, John B. Carter, and Aditya Akella. 2016. AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks. In *Proceedings of the 2016 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'16)*. 244–257.

[37] Hadi Hemmati, Andrea Arcuri, and Lionel C. Briand. 2011. Empirical Investigation of the Effects of Test Suite Properties on Similarity-Based Test Case Selection. In *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation (ICST'11)*. 327–336.

[38] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving High Utilization with Software-driven WAN. In *Proceedings of the 2013 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'13)*. 15–26.

[39] Meitian Huang, Weifa Liang, Zichuan Xu, Wenzheng Xu, Song Guo, and Yinlong Xu. 2016. Dynamic Routing for Network Throughput Maximization in Software-Defined Networks. In *Proceedings of the 2016 Annual IEEE International Conference on Computer Communications (INFOCOM'16)*. 1–9.

[40] Muhammad Usman Iftikhar, Gowri Sankar Ramachandran, Pablo Bollansée, Danny Weyns, and Danny Hughes. 2017. DeltaIoT: A Self-Adaptive Internet of Things Exemplar. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'17)*. 76–82.

[41] Dennis Ippoliti, Changjun Jiang, Zhijun Ding, and Xiaobo Zhou. 2016. Online Adaptive Anomaly Detection for Augmented Network Flows. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 11, 3 (2016), 17:1–17:28.

[42] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. 2014. Dynamic Scheduling of Network Updates. In *Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'14)*. 539–550.

[43] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *Computer* 36, 1 (2003), 41–50.

[44] Joshua D. Knowles and David W. Corne. 2000. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation* 8, 2 (2000), 149–172.

[45] Heiko Koziolek, Andreas Burger, and Jens Doppelhamer. 2018. Self-Commissioning Industrial IoT-Systems in Process Automation: A Reference Architecture. In *Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA'18)*. 196–205.

[46] Christian Krupitzer, Martin Breitbach, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. 2018. *A Survey on Engineering Approaches for Self-Adaptive Systems (Extended Version)*. Technical Report. University of Mannheim. 1–33 pages. Preprint published in Pervasive and Mobile Computing Journal, vol. 17, 2015, pp. 184–206.

[47] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets'10)*. 19:1–19:6.

[48] Ying-Dar Lin, Hung-Yi Teng, Chia-Rong Hsu, Chun-Chieh Liao, and Yuan-Cheng Lai. 2016. Fast Failover and Switchover for Link Failures and Congestion in Software Defined Networks. In *Proceedings of the 2016 IEEE International Conference on Communications (ICC'16)*. 1–6.

[49] Felipe A. Lopes, Marcelo Santos, Robson Fidalgo, and Stenio F. L. Fernandes. 2016. A Software Engineering Perspective on SDN Programmability. *IEEE Communications Surveys and Tutorials* 18, 2 (2016), 1255–1272.

[50] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Annals of Mathematical Statistics* 18, 1 (1947), 50–60.

[51] Matthew Mathis and Jamshid Mahdavi. 1996. Forward Acknowledgement: Refining TCP Congestion Control. In *Proceedings of the 1996 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'96)*. 281–291.

[52] Daniel A. Menascé, Hassan Gomaa, Sam Malek, and Jo ao P. Sousaa. 2011. SASSY: A Framework for Self-Architecting Service-Oriented Systems. *IEEE Software* 28, 6 (2011), 78–85.

[53] Assaad Moawad, Thomas Hartmann, François Fouquet, Grégory Nain, Jacques Klein, and Yves Le Traon. 2015. Beyond discrete modeling: A continuous and efficient model for IoT. In *Proceedings of the 18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS'15)*. 90–99.

[54] Frederick Mosteller and John Wilder Tukey. 1977. *Data Analysis and Regression: A Second Course in Statistics* (first ed.). Pearson.

[55] Sharief M.A. Oteafy and Hossam S. Hassanein. 2017. Resilient IoT Architectures Over Dynamic Sensor Networks With Adaptive Components. *IEEE Internet of Things Journal* 4, 2 (2017), 474–483.

[56] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2015. Reformulating Branch Coverage as a Many-Objective Optimization Problem. In *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST'15)*. 1–10.

[57] Henrique Brittes Pötter and Alexandre Sztajnberg. 2016. Adapting Heterogeneous Devices into an IoT Context-aware Infrastructure. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'16)*. 64–74.

[58] Konstantinos Poularakis, George Iosifidis, Georgios Smaragdakis, and Leandros Tassiulas. 2019. Optimizing Gradual SDN Upgrades in ISP Networks. *IEEE/ACM Transactions on Networking* 27, 1 (2019), 288–301.

[59] Evangelos Pournaras, Mark Ballandies, Dinesh Acharya, Manish Thapa, and Ben-Elias Brandt. 2018. Prototyping Self-managed Interdependent Networks: Self-healing Synergies against Cascading Failures. In *Proceedings of the 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'18)*. 119–129.

[60] Andres J. Ramirez and Betty H. C. Cheng. 2010. Design Patterns for Developing Dynamically Adaptive Systems. In *Proceedings of the 2010 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'10)*. 49–58.

[61] Andres J. Ramirez, David B. Knoester, Betty H.C. Cheng, and Philip K. McKinley. 2009. Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems. In *Proceedings of the 6th International Conference on Autonomic Computing (ICAC'09)*. 97–106.

[62] Albert Rego, Sandra Sendra, José Miguel Jiménez, and Jaime Lloret. 2017. OSPF routing protocol performance in Software Defined Networks. In *Proceedings of the 2017 4th International Conference on Software Defined Systems (SDS'17)*. 131–136.

[63] Zach Shelby, Klaus Hartke, and Carsten Bormann. 2014. *The Constrained Application Protocol (CoAP)*. Internet Standard RFC 7252. Internet Engineering Task Force (IETF).

[64] Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C. Briand, Chetan Arora, and Frank Zimmer. 2020. [Case study data] Dynamic Adaptation of Software-defined Networks for IoT Systems: A Search-based Approach. https://figshare.com/s/b4f6b58da221341989dc.

[65] Michael Stein, Alexander Frömmgen, Roland Kluge, Frank Löffler, Andy Schürr, Alejandro Buchmann, and Max Mühlhäuser. 2016. TARL: Modeling Topology Adaptations for Networking Applications. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'16)*. 57–63.

[66] Antero Taivalsaari and Tommi Mikkonen. 2017. A Roadmap to the Programmable World: Software Challenges in the IoT Era. *IEEE Software* 34, 1 (2017), 72–80.

[67] Martin Tappler, Bernhard K. Aichernig, and Roderick Bloem. 2017. Model-Based Testing IoT Communication via Active Automata Learning. In *Proceedings of the 10th International Conference on Software Testing, Verification and Validation (ICST'17)*. 276–287.

[68] Wen Wang, Wenbo He, Jinshu Su, and Yixin Chen. 2016. Cupid: Congestion-free Consistent Data Plane Update In Software Defined Networks. In *Proceedings of the 2016 Annual IEEE International Conference on Computer Communications (INFOCOM'16)*. 1–9.

[69] Danny Weyns, M. Usman Iftikhar, Danny Hughes, and Nelson Matthys. 2018. Applying Architecture-Based Adaptation to Automate the Management of Internet-of-Things. In *Proceedings of the 12th European Conference on Software Architecture (ECSA'18)*. 49–67.

[70] Sewell Wright. 1921. Correlation and Causation. *Journal of Agricultural Research* 20, 7 (1921), 557–585.

[71] Parisa Zoghi, Mark Shtern, Marin Litoiu, and Hamoun Ghanbari. 2016. Designing Adaptive Applications Deployed on Cloud Environments. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10, 4 (2016), 25:1–25:26.