Dynamic Adaptive Network Configuration for IoT Systems: A Search-based Approach

Seung Yeob Shin*, Shiva Nejati*, Mehrdad Sabetzadeh*, Lionel C. Briand* Chetan Arora*†, Frank Zimmer†

*SnT Centre, University of Luxembourg, Luxembourg
†SES Networks, Luxembourg

{seungyeob.shin,shiva.nejati,mehrdad.sabetzadeh,lionel.briand}@uni.lu, {chetan.arora,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. In this paper, we propose a dynamic adaptive network 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 relies on the research field of dynamic adaptive search-based software engineering (SBSE) to reconfigure an IoT network 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 IoT network to dynamically resolve congestion, and (2) compared to two baseline data forwarding algorithms that are static and non-adaptive, our approach increases the data transmission rate by a factor of at least 3 and decreases data loss by at least 70%.

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

I. 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 [1], [2]. 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 help them act quickly and effectively to prevent loss of lives and minimize damages.

Successful IoT systems necessarily depend on an underlying communication system that can transmit a large volume 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) [3] 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. By enabling software-defined control, SDN brings a whole set of software engineering considerations related to, among other things, scalability, performance and reliability [4].

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 [5], [6], [7], [8], [9]. Some more recent approaches study dynamic reconfiguration of SDN to maximize their quality of service [10], [11], [12]. None of these lines of work, however, consider or optimize a communication system 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 [13], [14], [15]. These studies, however, are geared toward offline optimization of system design or architecture, and cannot address the challenge of online and dynamic network 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 the reconfiguration cost. We refer to our approach as Dynamic adaptive CongEstion control algorithm for SDN (DICES). Inspired by feedback-loop control systems [16], 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 system 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, it 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) [17] 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. We develop DICES as an application in a widely used open-source SDN control platform, ONOS [18]. We combine this platform with a virtual network emulator, Mininet [19], to capture the topology and the characteristics of the network underlying an industrial national emergency management system in Luxembourg. For performing experiments, we use an open-source network traffic flow generator, D-ITG [20], in order to create various IoT traffic scenarios by combining sensor, video, audio and data streams. The information about the IoT traffic scenarios was provided by our industry partner, SES, 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 [21, 22], 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 [23].

Organization. The rest of this paper is organized as follows. Section II motivates the paper. Section III describes DICES. Section IV evaluates DICES. Section V compares with related work. Section VI concludes this paper.

II. 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.

III. APPROACH

The separation between software-defined data control and the physical aspects of network systems is a key feature of SDN [3]. 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 III-A we provide an abstract formalization of SDN concepts and use them to define the problem of network congestion.

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 III-B. Since SDN controller behavior is programmable through applications, we can enhance the data-forwarding function of SDN using DICES as described in Section III-C.

A. 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 \( G \) as \( (V,E,c,l) \), 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, \( v_1,v_2,v_3,v_4,v_5,v_6 \), and eight directed links \( e_1,e_2,\ldots,e_8 \); 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(b) 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 \( \text{links}(f) \) the set of links on the directed path \( f.p \) and by \( \text{links}(F) = \bigcup_{f \in F} \text{links}(f) \) 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 \( \text{flows}(e,F) \).

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 through \( e \), 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: \( \text{throughput}(e,F) = \sum_{f \in \text{flows}(e,F)} f.q.d. \)

We say a network \( G \) is congested by a given set \( F \) of flows if there is some link \( e \) such that \( \text{throughput}(e,F) > c(e) \). Given a network \( G \) congested by the set \( F \) of flows, we address the problem of network congestion by finding a new set \( F^a = \{ f^a_1,f^a_2,\ldots,f^a_n \} \) of flows where (1) each \( f^a_j \) processes the same request as that of the flow \( f_j \in F \), i.e., \( f^a_j.q = f_j.q \) 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., \( \text{throughput}(e,F^a) \leq c(e) \) for all \( e \in \text{links}(F^a) \). 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 III-C).

B. 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 [21], 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 III-C2 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.

C. Dynamic Adaptive Congestion Control (DICES)

DICES runs in parallel with the SDN data forwarding algorithm described in Section III-B. In contrast to the SDN data forwarding algorithm, DICES is designed to execute periodically with a time period \( \Delta \). 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 \( \Delta \) 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 \( T = [0, T] \) 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) [24, 25]. We denote by \( \text{util}(e, F_i) \) the utilization of link \( e \) by the flow set \( F_i \) and define it as follows: \( \text{util}(e, F_i) = \text{throughput}(e, F_i)/c(e) \). The “analyze” step deems \( e \) to be congested if \( \text{util}(e, F_i) > u \), 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 \( F^a \) of \( F_i \) that can handle the requests \( Q_i \) without congestion (see the congestion problem definition in Section III-A). If congestion cannot be resolved, it ensures that \( F^a_i \) 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 III-B) 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.

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 \( F^a_i \) 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 \( F^a_i \) 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 \( F^a_i \). 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 \( F^a_i \) are very different from those of \( F_i \) or if \( F^a_i \) 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 [26], 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 $F^q_i$ 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 $F = \{f_1, f_2, ..., f_i\}$ of flows where for every $f \in F$ we have $f.q \in Q$, for every $f' \in Q$ 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 $i\Delta$ 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 $\text{fitUtil}(F^q_i)$, $\text{fitCost}(F_i, F^q_i)$, and $\text{fitDelay}(F^q_i)$, respectively, where $F_i$ is the set of flows given as input and $F^q_i$ is a candidate flow set generated during the search.

The $\text{fitUtil}(F^q_i)$ fitness function is defined by equation (1) as the maximum link utilization across all the links used in $F^q_i$. Our approach aims to minimize equation (2).

$$\text{fitUtil}(F^q_i) = \max_{e \in \text{links}(F^q_i)} \text{util}(e, F^q_i)$$

The $\text{fitCost}(F_i, F^q_i)$ fitness function is defined by equation (2).

$$\text{fitCost}(F_i, F^q_i) = \sum_{(j,f') \in F_i \times F^q_i : f.q = f'.q} \text{dist}(j,f')$$

The $\text{fitDelay}(F^q_i)$ fitness function is defined by equation (3) which sums the delay values $l(e)$ of all the links $e$ used in a candidate solution $F^q_i$. Note that the delay objective can be estimated for a flow set $F^q_i$ only if $F^q_i$ does not give rise to congestion, i.e., only when $\text{fitUtil}(F^q_i) \leq u$, 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 $F^q_i$ leads to congestion, we assign an undefined value (i.e., a large number) to $\text{fitDelay}(F^q_i)$. Our approach minimizes equation (3).

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

Recall from Section III-A that congestion may not be resolved by our approach which is based on reassigning the flows. In this case, the objective $\text{fitDelay}()$ is excluded since it is undefined and returns a large number for all the congested solutions. But the search still minimizes $\text{fitUtil}()$ and $\text{fitCost}()$ and returns a solution $F^q_i$ 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 [29]: “A solution $F^p_i$ dominates another solution $F^q_i$ if $F^p_i$ is not worse than $F^q_i$ in all fitness values, and $F^p_i$ is strictly better than $F^q_i$ in at least one fitness.”

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 [14], [30], [28]. (3) Mutation. We use the mutation algorithm in Fig. 5. It replaces a randomly selected flow $f_k$ in $F^i_o$ (lines 8–9) with an alternative flow $f'_k$ for $f_k$ such that $f'_k, q = f_k, q$ (lines 10–11).

Choosing an optimal solution. The output of NSGAII is a set of equally viable solutions (line 28 in Fig. 4). Researchers have proposed various alternatives for selecting an optimal solution among all the solutions on an optimal Pareto front, such as a knee solution [31] or the corner solution [32] for an objective. In our work, we use a knee solution.

![Flow mutation algorithm](image1.png)

Fig. 5. A flow mutation algorithm.

![Link weight adjustment algorithm](image2.png)

Fig. 7. A utilization-aware link weight adjustment algorithm.

IV. EMPIRICAL EVALUATION

In this section, we present an evaluation of DICES. Our full evaluation package is available online [23].

A. 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 (scability): 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) [21] and an open shortest path first algorithm (OSPF) [22]. RFWD and OSPF, discussed in Section V-E, are commonly used for optimal data forwarding and congestion avoidance, respectively [21], [54], [55], [56].
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) [18]. ONOS has been used extensively in research and practice [21], [23], in particular for large-scale network systems. To simulate networks, we use Mininet [19] and D-ITG [20]. 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.

### C. Study Subjects

We use two types of study subjects: (1) some synthetic networks, and (2) EMS – a large-scale industrial system under study by Company X (see Section II). 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 our analysis. The network bandwidth and delay values in normal areas, and EN external networks (in a disaster area). The delay metric for a flow measures the number of packets dropped associated with the flow over a time period, e.g., time interval $\Delta$. The delay metric for a flow measures the number of packets dropped associated with the flow over a time period, e.g., time interval $\Delta$. The delay metric for a flow measures the number of packets dropped associated with the flow over a time period, e.g., time interval $\Delta$. The delay metric for a flow measures the number of packets dropped associated with the flow over a time period, e.g., time interval $\Delta$. The delay metric for a flow measures the number of packets dropped associated with the flow over a time period, e.g., time interval $\Delta$.

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 [III-C] for the definition of $util(e, F)$).

To answer RQ3, we use the data traffic profile shown in Table I 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 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) [21] and an open shortest path first forwarding algorithm (OSPF) [22]. 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 III-B when link weights are all equal to one and are fixed all the time. 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.

F. Parameter Tuning and Setting

We set $\Delta$, 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 [41], we set the NSGAII parameters as follows: the population size = 100, the crossover probability = 0.8, and the mutation probability = 1/|$F_1$|. 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.

G. 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 IV-E). 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 III-B) 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 realistic networks 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 IV-E), 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 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 II shows the average delay and packet loss values for EMS when one uses DICES, RFWD and OSPF for handling the requests described in Table I. As discussed in Section IV-E, 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 I). 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.

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 I). The weighted averages, given in the last row of Table II, 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. Our industry partner (SES) found the minimized congestion, while RFWD and OSPF do not require any reconfiguration.
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.

H. 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.

V. 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 [5], [6], [7], [8], [9]. For example, the TCP congestion control algorithm is prevalently used over the Internet and has been addressed by many prior research threads [5], [6], [7], [8]. More recent work in this direction includes new application-layer protocols such as CoAP [42] and its congestion control algorithm, CoCoA [9]. 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 [43], [44], [45], [46], [10], [47], [11], [12]. Chiang et al. [10] formulate a new optimization problem to find optimal routing paths for group communication traffic. Gay et al. [11] propose a local search-based segment routing method for networks with unexpected failures. Huang et al. [12] present a dynamic routing algorithm to maximize network throughput under link-capacity and user-demand constraints. 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 [11], [48]. The research topics related to IoT are numerous, e.g., data models to capture highly volatile IoT data [49], model-based code generation for heterogeneous things [50], model-based testing of IoT communications [51], IoT architectures [52], [53], and self-adaptive IoT systems [54], [55], [56], [57]. To our knowledge, the problem of dynamically reconfiguring an IoT system in an unpredictable environment, as tackled in our work, has not been studied before.

Self-adaptive systems have been studied in many domains [59], [60]. DICES relates to work on self-adaptation in the network domain, e.g., adaptive network anomaly detection [61], adaptive network monitoring [62], [63], self-adaptive multiplexing [64], 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 [17], as noted in Section I, 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 [66], [14], [67] and configuring such systems [15], [13]. To our knowledge, we are the first to have addressed the problem of congestion control in the context of dynamic adaptive SBSE.

VI. 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. 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).

ACKNOWLEDGMENT
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] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and H. Mohammadi, “Internet of things: A survey on enabling technologies, protocols, and applications,” IEEE Communications Surveys Tutorials, vol. 17, no. 4, pp. 2347–2376, 2015.
[2] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” Computer Networks, vol. 54, no. 15, pp. 2787–2805, 2010.
[3] E. Haleplidis, K. Pentikousis, S. G. Demazis, J. H. Salim, D. Meyer, and O. G. Koufopavlou, “Software-defined networking (SDN): Layers and architecture terminology,” Internet Research Task Force (IRTF), Information RFC 7426, 2015.
[4] F. A. Lopes, M. Santos, R. Fidalgo, and S. F. L. Fernandes, “A software engineering perspective on SDN programmability,” IEEE Communications Surveys Tutorials, vol. 18, no. 2, pp. 1255–1272, 2016.
[5] M. Mathis and J. Mahdavi, “Forward acknowledgment: Refined TCP congestion control,” in Proceedings of the 1996 ACM Conference on Special Interest Group on Data Communication (SIGCOMM’96), 1996, pp. 281–291.
[6] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center TCP (DCTCP),” in Proceedings of the 2010 ACM Conference on Special Interest Group on Data Communication (SIGCOMM’10), 2010, pp. 63–74.
[7] S. Ferlin, O. Alay, T. Dreiholz, D. A. Hayes, and M. Welzl, “Revisiting congestion control for multipath TCP with shared bottleneck detection,“ in Proceedings of the 2016 Annual IEEE International Conference on Computer Communications (INFOCOM’16), 2016, pp. 1–9.
[8] Z. Xiong, B. Lantz, B. Heller, and N. McKeown, “TCP Cubic congestion window adaptation on a 100Gb/s link,” in Proceedings of the 2016 IEEE Conference on Computer Communications (INFOCOM’16), 2016, pp. 1–9.
[9] A. Betzler, C. Gomez, I. Demirkol, and J. Paradells, “CoAP congestion control for the internet of things,” IEEE Communications Magazine, vol. 54, no. 7, pp. 154–160, 2016.
[10] S. Cheng, J. Kao, S. Shen, Y. Yang, and W. Chen, “Online multicast traffic engineering for software-defined networks,” in Proceedings of the 2018 Annual IEEE International Conference on Computer Communications (INFOCOM’18), 2018, pp. 414–422.
[11] S. Gay, R. Hartert, and S. Vissicchio, “Expect the unexpected: Sub-second optimization for segment routing,” in Proceedings of the 2017 Annual IEEE International Conference on Computer Communications (INFOCOM’17), 2017, pp. 1–9.
[12] M. Huang, W. Liang, Z. Xu, W. Xu, S. Guo, and Y. Xue, “Dynamic routing for network throughput maximization in software-defined networks,” in Proceedings of the 2016 Annual IEEE International Conference on Computer Communications (INFOCOM’16), 2016, pp. 1–9.
[13] P. Zoghi, M. Shtern, M. Litoiu, and H. Ghanbari, “Designing adaptive applications deployed on cloud environments,” ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 10, no. 4, pp. 25:1–25:26, 2016.
[14] S. S. Andrade and R. J. de A. Macêdo, “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), 2013, pp. 61–70.
[15] A. J. Ramirez, D. B. Knoester, B. H. Cheng, and P. K. McKinley, “Applying genetic algorithms to decision making in autonomic computing systems,” in Proceedings of the 6th International Conference on Autonomic Computing (iCAC’09), 2009, pp. 97–106.
[16] J. O. Keenan and D. M. Chess, “The vision of autonomic computing,” Computer, vol. 36, no. 1, pp. 41–50, 2003.
[17] M. Harman, E. Burke, J. Clark, and X. Yao, “Dynamic adaptive search based software engineering,” in Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESE’12), 2012, pp. 1–8.
[18] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. L. O’Connor, P. Radoslavov, W. Snow, and G. Panoulas, “ONOS: Towards an open, distributed SDN OS,” in Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN’14), 2014, pp. 1–6.
[19] 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 (HotNets’10), 2010, pp. 19:1–19:6.
[20] A. Botta, A. Dainotti, and A. Pesca˘p, “A tool for the generation of realistic network workload for emerging networking scenarios,” Computer Networks, vol. 56, no. 15, pp. 3531–3547, 2012.
[21] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Verrcelone, “Scalability of ONOS reactive forwarding applications in ISP networks,” Computer Communications, vol. 102, pp. 130–138, 2017.
[22] J. Moy, “OSPF version 2,” Network Working Group, Internet Standard RFC 2328, 1998.
[23] S. Y. Shin, S. Nejati, M. Sabatezadeh, L. C. Briand, C. Arora, and F. Zimmer, “[case study data] dynamic adaptive network configuration for IoT systems: A search-based approach,” https://figshare.com/s/59907721c190a9091b655, 2019.
[24] Y. Lin, H. Teng, C. Hsiu, C. Liao, and Y. Lai, “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), 2016, pp. 1–6.
[25] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Zhou, “A roadmap for traffic engineering in SDN/OpenFlow Networks,” Computer Networks, vol. 71, pp. 1–30, 2014.
[26] F. Ferrucci, M. Harman, J. Ren, and F. Sarro, “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), 2013, pp. 462–471.
[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. The MIT Press, 2009.
[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, 2002.
[29] J. D. Knowles and D. W. Corne, “Approximating the nondominated front using the pareto archived evolution strategy,” Evolutionary Computation, vol. 8, no. 2, pp. 149–172, 2000.
[30] H. Hemmati, A. Arcuri, and L. C. Briand, “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), 2011, pp. 327–336.
[31] J. Branke, K. Deb, H. Dierolf, and M. Osswald, “Finding knees in multi-objective optimization,” in Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN’04), 2004, pp. 722–731.
[32] A. Panichella, F. M. Kifetew, and P. Tonella, “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), 2015, pp. 1–10.
[33] T. Chen, K. Li, R. Bahsoon, and X. Yao, “FEMOSAA: Feature-guided and knee-driven multi-objective optimization for self-adaptive software,” ACM Transactions on Software Engineering and Methodology (TOSEM’18), vol. 27, no. 2, pp. 5:1–5:50, 2018.
[34] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Verrcelone, “Evaluating the SDN control traffic in large ISP networks,” in Proceedings of the 2015 IEEE International Conference on Communications (ICC’15), 2015, pp. 5248–5253.
