applied sciences

MDPI

Article

MPResiSDN: Multipath Resilient Routing Scheme for SDN-Enabled Smart Cities Networks

Sarah L. Aljohani and Mohammed J. F. Alenazi *

Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia; 437203235@student.ksu.edu.sa
* Correspondence: mjalenazi@ksu.edu.sa

Abstract: The number of smart cities is increasing rapidly around the world with the continuous increase of governments' interest in exploiting Information and Communication Technologies (ICT) to solve issues arising from rapid urbanization. Most smart city services rely fundamentally on ubiquitous sensing, enabled by Wireless Sensor Network (WSN) technologies. However, WSNs in smart cities are naturally vulnerable to unavoidable external challenges like storms, fires, and other natural disasters. Such challenges pose a great threat to smart city infrastructure, including WSNs, as they might affect network connectivity or result in complete blockages of network services. However, some particular smart city services are critical, to the point where they must remain available in all situations, especially during disasters; to monitor the disaster and obtain sensory information needed for controlling it, limiting its danger, or for decision-making during rescue operations. Thus, it is crucial to design a smart-city network to maintain connectivity against such challenges. In this paper, we introduce MPResiSDN, a MultiPath Resilient routing system based on Software Defined Networking (SDN). The system introduced exploits SDN's capabilities and aided-multipath routing to reactively provide connectivity in smart city networks in the presence of challenges. We evaluated our proposed system under simulations of different natural disasters. The results demonstrate that the system improved data delivery under the challenges by as much as 100% compared to the Spanning Tree Protocol when a suitable value for k diverse paths was selected.

Keywords: resilienc; routing protocols; smart city; Software Defined Networking (SDN); Wireless Sensor Networks (WSN)

Citation: Aljohani, S.L.; Alenazi, M.J.F. MPResiSDN: Multipath Resilient Routing Scheme for SDN-Enabled Smart Cities Networks. *Appl. Sci.* **2021**, *11*, 1900. https://doi.org/10.3390/app11041900

Academic Editor: Eusebi Calle

Received: 27 December 2020
Accepted: 15 February 2021
Published: 22 February 2021

1. Introduction and Motivations

In the last few decades, the number of people who live in cities has increased dramatically. In 2018, 55% of the world's population lived in cities, and that percentage is expected to reach 68% by 2050 [1]. Rapid urban growth poses tangible challenges for the implementation of government development plans to deliver safe, resilient, and sustainable services to an increasing number of citizens [2]. In response, many governments have launched smart city projects to address urbanization issues and challenges [3]. According to the Cities In Motion Index, globally, more than 180 cities in more than 80 countries have been evaluated as smart cities [4]. These numbers increase each year as the number of people who live in urban areas increases. Keeping pace with this development, governments' interest and investments in smart city projects also increases.

Smart cities are considered one of the most important applications of the vision of the Internet of Things (IoT) [5] The IoT connects physical items with embedded sensors and/or software to send or receive information over the Internet. Access to the Internet enables "smart" things, i.e., IoT objects, to make decisions and determine actions based on real-time information. The IoT vision may sound aspirational in view of the fact that connecting everything directly to the Internet is not technically feasible due to problems related to current Internet protocols and inadequate infrastructure. However, this is not

the complete story, because other existing technologies, such as Wireless Sensor Networks (WSN) could play a major role in achieving the vision.

WSNs are considered suitable technology to enable the IoT [6,7]. Ubiquitous sensing enabled by WSNs could help achieve the "everything connected to the Internet" vision [8] in smart cities. WSNs are preferable and play an important role in many smart city applications [9], because they are relatively inexpensive, scalable, and easy to implement [10]. Important smart city applications that utilize WSNs include disaster monitoring and management, surveillance applications, smart transportation systems, traffic management, and smart healthcare systems, as well as atmospheric monitoring, pollution reduction, and energy-saving applications [11,12].

WSNs function effectively in ordinary stable environmental conditions; however, they have difficulty handling challenging environments [13]. In fact, WSNs are vulnerable to various challenges when implemented in real time in smart city environments. Here, a challenge is defined as any event that impacts the normal operation of the network [14]. Challenges trigger faults that may permanently or temporarily block the transmission of sensor information. Therefore, failure management is a significant aspect to be considered when developing any smart city project [5].

Challenges in a smart city environment include various natural disasters, such as severe storms, fires, floods, tornadoes, earthquakes, and volcanic eruptions. Such challenges can affect network connectivity due to fluctuations of wireless channel attenuation [15]. Some challenges, e.g., fires, can disrupt network services completely. However, some smart city services are critical and should remain available, particularly during disasters, to obtain the sensory information required to control them, limit their impact, and provide critical information for decision-making during rescue operations. Thus, improving the resilience of these underlying networks in smart cities is crucial [16].

Resilience refers to the ability of the network to adapt and retain basic functionality when errors, failures, and environmental changes occur [17]. Challenges to WSNs can cause link or node failures [18]. Under challenging conditions, the ability to respond quickly to changes in network topology can significantly improve data delivery. WSNs use a multi-hop routing mechanism; thus, alternative routes may be available. However, the nature of the distributed routing implemented in WSNs makes it difficult to update routing tables quickly and effectively. This problem can be solved by utilizing Software-defined Networking (SDN). Integrating SDN into WSNs in smart cities contributes to the programmability and central management requirements that must be satisfied to provide efficient routing in case of failures.

This paper introduces an SDN-based multipath routing scheme, which we refer to as MPResiSDN, that aims to improve the resilience of WSNs in smart cities. The proposed scheme implements a multipath-aided strategy to achieve better routing in case of natural challenges. Implementing this scheme would help rescue efforts, save people's lives, and limit property damage in case of natural disasters, because it enhances the availability of current and correct sensory information that is used by survivability applications in smart cities to make decisions.

The primary contributions of this paper are as follows. We introduce the MPResiSDN scheme and its components, implement the scheme using an RYU controller, emulate a sensor network, and apply different challenges to the emulated network under operational conditions. In addition, we evaluate network performance under each challenge in terms of network throughput, actual data delivery, end-to-end delay, and overhead. We compare and analyze the results obtained with and without the proposed scheme.

The remainder of this paper is organized as follows. Background information is provided in Section 2. Related work is discussed in Section 3. The proposed system model is described in Section 4. The evaluation method is explained in Section 5, and evaluations of the results are discussed in Section 6. Conclusions and suggestions for future work are presented in Section 7.

2. Background

This section provides definitions and background information related to smart cities, WSNs, SDN, and network resilience.

2.1. Smart Cities

Smart cities is a general term that can be considered from various perspectives. Exploring the key layers of a smart city will clarify where this research is positioned. Figure 1 shows a general overview of smart city architecture [19].

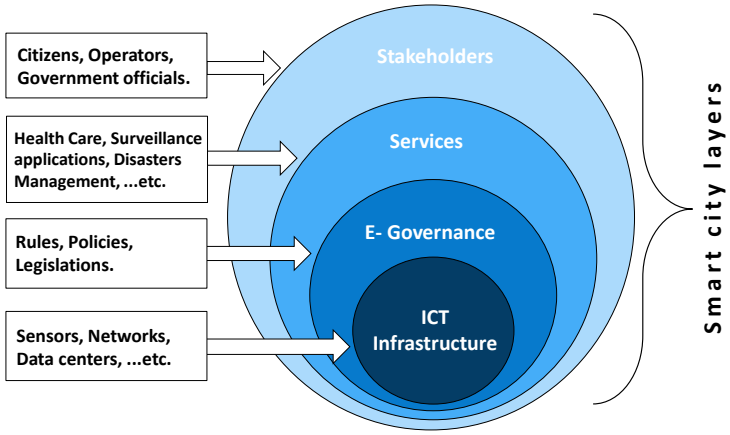


Figure 1. Smart city architecture.

As can be seen, smart city architecture comprises four different layers. The first is the Information and Communication Technologies (ICT) infrastructure layer. This layer includes all essential network infrastructure, such as data centers, network devices, sensors, and actuators. This layer is considered the key driver of smart cities. The second layer is the E-governance layer. E-governance establishes policies related to the deployment of existing ICT infrastructure to provide services to citizens. The third layer is the Services layer that includes the various public services offered to citizens and other stakeholders. Finally, the Stakeholders' layer represents all beneficiaries of smart cities services, including citizens, government officials, and commercial operators. This research focuses on the ICT infrastructure layer.

2.2. Wireless Sensor Networks

In smart cities, WSNs are a major source of sensory information [12]. A general example of the infrastructure and technologies used in a smart city crowd monitoring system is shown in Figure 2 [9]. The WSN is used along with crowd sensing devices, such as smartphones and smart wearables. Then, different communication technologies, such as ZigBee, Wi-Fi, VANET, LTE, and 5G, are used to transmit the sensed data to a data center/cloud server for analysis. In the computing layer, data from different sources are merged and analyzed to extract meaningful information to be used later for decision-making by smart city services.

A single WSN generally consists of multiple sensor nodes and a sink node. Sensor nodes are deployed over a specified sensing area to sense information about the surroundings and communicate to the sink node through a multi-hop mechanism [20,21]. The sink node is responsible for collecting the sensed information from the sensor nodes, aggregating the information, and sending the aggregated information to higher-level network devices for further processing and information extraction.

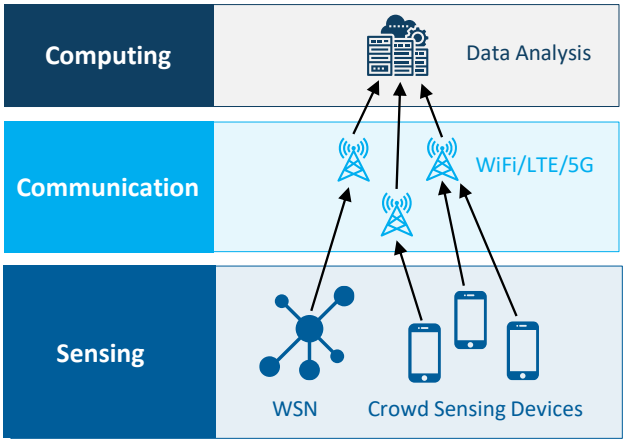


Figure 2. An example of smart city monitoring.

It is worth noting that, for smart city sensing applications, wired sensor nodes can be incorporated into the WSN. These wired sensor nodes could have better communications, computation, and storage capabilities than normal wireless sensor nodes. Although the network may include some wired sensors, it is still called a WSN because most of the sensor nodes are wireless [9].

2.3. Software Defined Networking

Software-Defined Networking (SDN) is an emerging technology that employs network architecture that differs from the architecture used in traditional networks. SDN-enabled networks decouple the control plane from the data plane by separating the routing decisions and the forwarding decisions. This separation is the key factor that enables all SDN capabilities. Figure 3 represents a logical view of SDN architecture [22]. The application layer deals with end-user applications that utilize SDN services. The control layer is the main layer in the architecture and comprises a set of software-enabled SDN controllers that centralize the network intelligence and maintain a global view of the network. Network administrators use the controllers to apply custom policies to the devices of the infrastructure layer; the infrastructure layer consists of physical devices, such as switches and routers. In SDN architecture, routers are no longer responsible for routing. Instead, they only forward packets based on flow tables that are obtained from SDN controllers. SDN controllers connect to routers through southbound APIs, and applications connect to SDN controllers through northbound APIs [23].

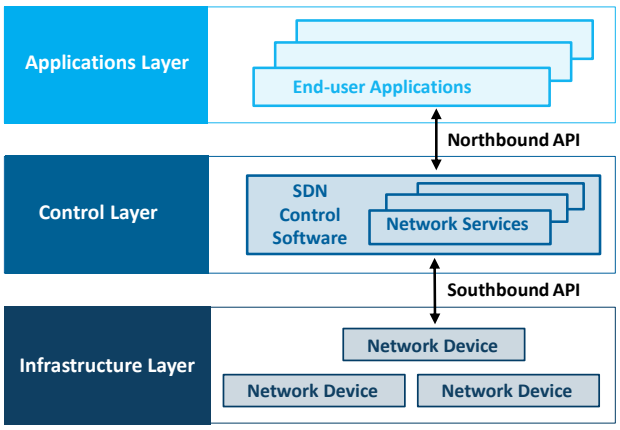


Figure 3. SDN architecture.

Implementing SDN has numerous advantages [24,25]. First, SDN provides centralized network management because an SDN controller has a centralized view of the entire net-

work status, which makes network management much easier. In addition, SDN enhances the efficiency of network administration, as it permits network characteristics to be changed remotely. Moreover, SDN offers programmability features such that policies and protocols can be deployed to any network device. SDN also facilitates configuration and reconfiguration processes. Furthermore, implementing SDN is cost-efficient because most SDN products are open source. In addition, SDN supports up to layer-3; therefore, enterprises do not need to purchase expensive hardware [26]. SDN also provides fine-grained security to end-devices [27].

2.4. Network Resilience

Network resilience refers to a network's ability to defend against and maintain an acceptable level of service in the presence of various faults and challenges to normal operation [28]. Resilience is a general term that applies to several areas, such as cybersecurity, fault tolerance, software dependability, and network survivability [29]. The different network resilience disciplines [30] are mapped in Figure 4. In this study, network resilience falls under Challenge Tolerance, specifically, under Disruption Tolerance, which is highlighted in blue in the figure.

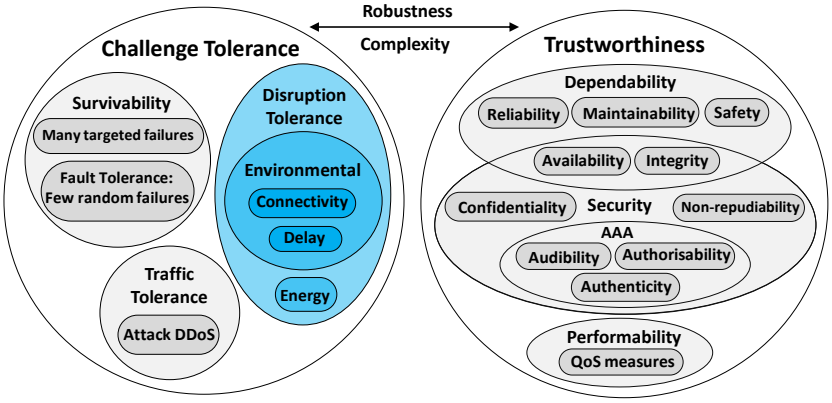


Figure 4. Resilience disciplines.

A challenge is defined as an event that impacts normal network operations [14]. In general, a challenge can be caused by malicious or non-malicious sources. A major non-malicious challenge in a communication network is a disruption [30]. Disruptions come from challenges in the communication environment that make it difficult for the network to maintain stable end-to-end connections between nodes. Disruption tolerance is the ability of a network to tolerate disruptions in connectivity among its nodes. Examples of disruptions are weak signal connectivity, unpredictably long delays, and power or battery issues [30]. This study addresses two types of disruption, which, in this context, are called challenges. The first is a relatively trivial challenge that causes weak signal connectivity, i.e., the storm challenge. The second is a severe challenge that disconnects some nodes completely, i.e., the fire challenge.

3. Related Work

SDN is a revolutionary technology that provides advanced features to infrastructure networks. Opportunities to exploit SDN to improve smart city networks are endless. Researchers have investigated exploiting SDN to improve different aspects of smart city networks, such as network security, network performance, big data processing, load balancing, traffic management, and surveillance applications.

For example, various studies [31–34] have discussed how SDN can be utilized to improve network security in smart cities, in particular, to improve resilience to DDoS attacks. Bawany et al. [31] introduced an SDN-based attack defense framework to detect and mitigate application level DDoS attacks on smart city application servers. Then, to

achieve the same goal more effectively, they improved their work and introduced a secure and agile adaptive framework [32]. This framework protects smart city applications against DDoS attacks by leveraging the advantages of SDN, such as central management and global visuality. In addition, Chen et al. [33] presented a lightweight approach to trace back DDoS attacks on SDN-based smart cities, based on an anomaly tree. This approach benefits from SDN's hierarchical control structure. Xu et al. [34] proposed a defense strategy against DDoS attacks based on SDN and Network Function Virtualization (NFV).

SDN has been also utilized to achieve different targets in smart city networks, such as to improve performance. In this regard, Han et al. [35] studied integrating SDN into smart city networks to improve 5G performance. They introduced a cell-less communications model and architecture to improve 5G network convergence. In addition, Usman et al. [36], introduced a software-defined device-to-device communication architecture for public safety applications in 5G networks in smart cities.

Furthermore, SDN has been utilized to develop mechanisms to transfer and process big data in smart cities. Bi et al. [37] proposed a strategy to overcome time-constrained big data transfer scheduling (TBTS) problems based on SDN. The strategy uses an SDN controller to achieve dynamic flow control and implements fast multipath transfer scheduling to overcome the TBTS problem. The results show that the strategy reduces big data transfer delay and improves bandwidth utilization. SDN has also been used to investigate new ways to improve big data processing in smart cities. Khan et al. [38] introduced three-tier architecture to implement IoT in smart cities. The architecture comprises data collection, data management, and application levels with two intermediate levels that work on SDN principles. The architecture was evaluated, and the evaluation results demonstrated that efficient transfer of big data for real-time applications had been achieved.

Other studies also attempted to integrate SDN in WSNs in smart cities. Those studies exploited SDN capabilities to balance the load on the sensors. Cui et al. [39] introduced a load balancing algorithm that was specifically designed to improve average bandwidth utilization and reduce network link load jitter; thus, the performance of the entire network was improved.

Utilizing SDN has also been proposed for surveillance applications in smart cities. Rametta et al. [40] presented a smart video surveillance platform for smart cities based on SDN and NFV. Kunst et al. [41] attempted to improve network resource allocation in a heterogeneous video surveillance network in smart cities by exploiting the advantages of SDN.

SDN has also been exploited to improve traffic management in smart cities. Raja et al. [42] introduced an SDN-enabled traffic alert system for the Internet of Vehicles (IoV) in smart cities. The system uses SDN controllers to update vehicles about non-line-of-sight information by broadcasting alarms that are produced automatically, either by another vehicle or by roadside units. The system also detects accidents, based on vehicle information, and broadcasts necessary alarms to all vehicles.

SDN technology has also been utilized to improve routing in IoV environments. Abbas et al. [43] introduced road-aware routing based on SDN for IoV networks in smart cities. The introduced routing mechanism improved network performance in terms of end-to-end delay, packet delivery ratio, and routing overhead. Bhatia et al. [44] introduced an innovative approach to real-time traffic analysis in Vehicular Ad-Hoc Network (VANET) environments based on SDN. The proposed approach relies on the programmability of SDN to implement a combination of different machine learning algorithms to model traffic flow in SDN-enabled smart cities.

SDN has been proposed to manage traffic in smart cities in emergency situations, such as natural disasters, traffic accidents, and terrorist attacks. Rego et al. [45] proposed an SDN-based architecture for smart cities that aims to manage traffic efficiency under disaster conditions. In that study, SDN controllers were utilized by the proposed architecture to collect information from different sensory networks in smart cities, such as traffic lights and surveillance cameras. The information collected is combined to obtain the best and fastest

evacuation routes and access roads for emergency services units. The authors further tested their proposed architecture in a follow-up study [46]. The results showed that the architecture reduced network delay by 33%. In addition, energy consumption by the nodes in the sensory networks was modeled and evaluated. The results proved the scalability of the architecture, as they demonstrate that energy consumption increases linearly with the number of nodes.

Other techniques have been proposed to handle emergency situations in smart cites. García et al. [47] proposed a system based on citizens' smartphones to detect emergency situations in smart city environments. The proposed system utilizes data from accelerometers in smartphones to detect changes in user behavior. Different tests were performed using real devices to show the possibility of identifying different behaviors, including walking, running, falling, and remaining on the floor. Also, Fragkos, G., et al. [48] proposed a multi-agency disaster management framework for unmanned aerial vehicles (UAV)-assisted public safety systems that can be used for smart cities. The framework they introduce exploits the principles of game theory and reinforcement learning to support connectivity during disasters and under challenging communication situations. Furthermore, Huang et al. [49] investigated the use of a machine learning approach to improve network connectivity. The approach addresses the connectivity holes that may exist within a network to achieve undisturbed connectivity.

Applications that deal with emergency situations and surveillance applications are critical in smart cities. The availability of these applications relies fundamentally on network resilience. Note that our research targets improving network resilience in the face of natural challenges in order to maintain the availability of these important applications. To that end, related studies that address improving network resilience are reviewed.

SDN is also utilized to achieve network resilience in smart cities. Ahmed and Adel A. [50] presented a lightweight SDN architecture for resilient real-time IoT that can be used in smart cities. The architecture achieves resilience by optimizing the regular control plane and data plane to introduce lightweight versions. The control plane was optimized by stopping the two least important control functions based on an application-specific requirement and by reducing the duty cycle of the control plane based on the demand of the data plane. The data plane is optimized by implementing real-time routing protocols, load disruption, and adaptive traffic shaping. The lightweight SDN architecture was evaluated using Mininet-IoT. The results showed that the lightweight SDN architecture outperformed traditional architecture in terms of delivery ratio, latency, and packet overhead. Here, the researchers attempted to improve resilience by improving network performance, which leads to improved availability of network services. However, they only considered normal network operation when no external challenges were presented.

Our work differs in that it aims to improve the resilience of smart city networks against external challenges, such as natural disasters. We utilize SDN's programmability feature to implement multipath routing to increase the availability of network services in smart cities, even under difficult circumstances, such as natural disasters. Multipath mechanisms, such as path diversification, are considered effective solutions to obtain high communication reliability between disconnected pairs [51]. The multipath concept has been used by researchers to improve various aspects of smart city networks at different network layers.

Singh et al. [52] introduced the potential of integrating Multipath TCP (MPTCP) with SDN in smart cities to improve Vehicle-to-Infrastructure (V2I) communication. They evaluated the performance of MPTCP for V2I connectivity in SDN-controlled small cells of DSRC and Wi-Fi. The results showed that MPTCP improved network resilience to internal failures caused by delays in flow setup or handovers.

Other researchers studied utilizing SDN along with multipath mechanisms to achieve resilience outside of the context of smart cities. Ai et al. [53] attempted to improve the resilience of SDNs against security attacks by implementing an algorithm that utilized a network-coding technique together with multipath routing. The results showed that the

proposed algorithm improved network resilience to passive security attacks by approximately 20%.

In addition, Cheng et al. [54] introduced a cross-layer resilient routing protocol stack for survivable network communication during regional challenges. The protocol, which they call GoeDivRP, operates based on SDN, collects network statistics, and calculates multipath communications. The protocol was implemented in the NS-3 network simulator and compared to Multipath TCP. The results showed that the protocol stack provided higher throughput and resilience against regional challenges compared to Multipath TCP.

Multipath routing has also been used with SDN to enhance QoS. Rezende et al. [55] proposed a general SDN-based framework to route multi-stream transport traffic over multipath networks. The system provides an interface for applications to specify multi-stream rules. Then, it utilizes SDN to ensure that the multiple streams follow the multiple paths based on the specified rules. The framework improved QoS in terms of delay and throughput.

To summarize, SDN technology has been used as a foundation to improve different aspects of smart city networks. It has been used to improve smart city network security [31–34], 5G network performance [35,36], transferring and processing of big data [37,38], and to balance sensor loads [39]. In addition, SDN technology has been used to improve surveillance systems in smart cities [40,41,56]. It has also been utilized to manage traffic effectively [42–46].

However, none of these studies addressed utilizing SDN to improve network resilience under challenging external conditions. We note that Ahmed and Adel A. [50] addressed using SDN to improve network resilience; however, they only considered internal challenges, and they only evaluated their system under normal operating conditions with no external challenges present. Moreover, they investigated using a lightweight SDN architecture to achieve network resilience, whereas our study uses multipath routing.

Multipath routing has been used to improve resilience with SDN by Singh et al. [52]; however, they did not consider evaluating network performance under external natural challenges. Similarly, Ai et al. [53] used multipath routing to improve resilience against security attacks.

Cheng et al. [54] considered regional challenges and implemented their work on a different type of network, Sprint. Rezende et al. [55] implemented a system that works in a somewhat similar way to ours; however, they did not test it under external challenges.

To the best of our knowledge, our proposed MPResiSDN system is the first to address utilizing SDN and multipath routing to improve network resilience against external challenges. We implemented our scheme on a modeled smart city environment and applied external challenges dynamically while the SDN controller was running. Then, we compared the scheme performance under the applied challenges in terms of throughput, goodput, overhead, and delay to its performance under normal operating conditions.

4. MPResiSDN System

In this section, the proposed MPResiSDN routing scheme is described. First, the system architecture and system components are described. Then, the proposed algorithm is presented.

4.1. System Components

In this section, the MPResiSDN system components are described. As shown in Figure 5, the proposed system comprises six main components: Topology Discovery, Challenge Detector, Link Status, Nodes Manager, *k*-diverse Paths, and Rules Generator. The description of each component is outlined as follows:

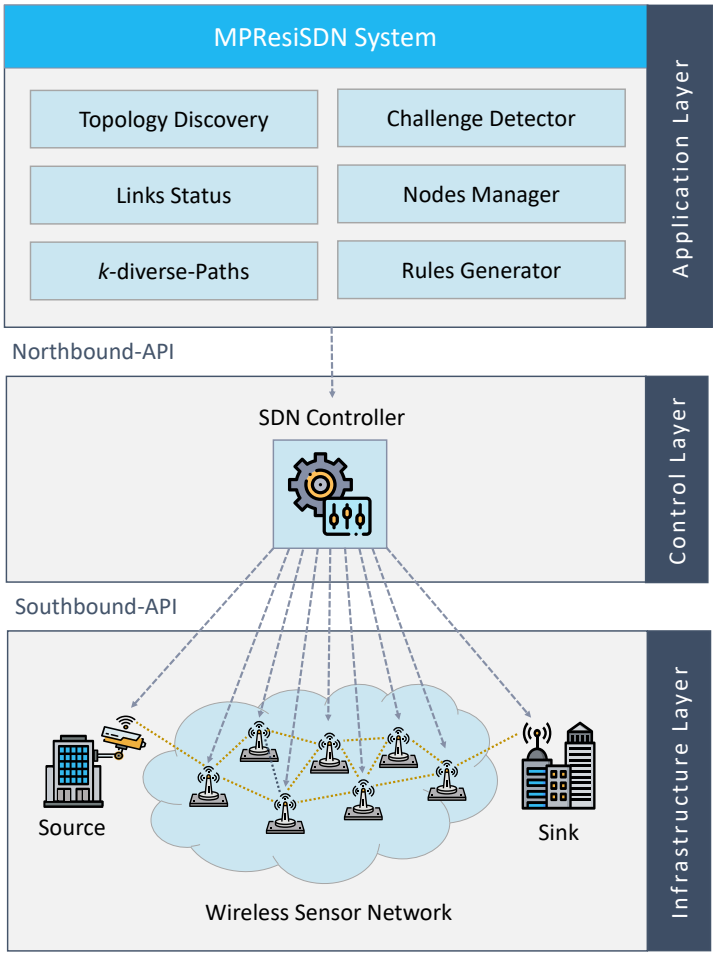


Figure 5. MPResiSDN system architecture.

- Topology Discovery: This component is responsible for identifying the network topology and providing information about the available sensor nodes and links. It initiates as soon as the network connects to the SDN controller. It provides an ID list of all connected nodes and links. In addition, it determines the interfaces to which each link and node are connected. The component achieves these tasks by exploiting the built-in topology discovery algorithm provided in the RYU controller.
- Challenge Detector: This component detects challenges inside the sensing area. It also specifies the type of challenge. It uses sensors, such as temperature sensors, to detect the possibility of fire challenges. It also uses weather sensors, such as lightning and humidity sensors, to detect storms. Furthermore, this component is responsible for setting the k parameter, which represents the number of paths to be used by the system. The k parameter value depends on the results of the challenge detecting process. The k value increases relative to the severity of the challenge. The severity level is proposed to be determined based on the obtained sensory information. The implementation of this component is outside the scope of this research.
- Link Status: This component stores link parameters, such as link bandwidth and bit error ratio. It tracks link availability and reports disconnected links.
- Nodes Manager: The Nodes Manager stores node attributes and configurations. It maintains node types, such as a sensor or sink. It also gathers information about its power source, i.e., whether it is battery- or solar-powered. In addition, it keeps track of the status of nodes and reports node failures.
- **k**-diverse-Paths: This is the key component of the system. It takes the output parameter k from the Challenge Detector component and then applies the Floyd-Warshall algorithm [57] to compute k diverse paths between nodes and available sinks. The

computed paths are then used by the Rules Generator component to set the SDN rules for the system. Note, this component receives the reports about link and node failures sent by the Link Status and Nodes Manager components. It acts on the failure reports by eliminating any failed node or link, and then updates the network topology to avoid utilizing failed nodes and links when computing k diverse paths.

- Rules Generator: This component takes the list of computed k paths and generates the SDN rules associated with each path. Then, it uses the rules to fill the routing tables of all nodes in the used paths. Note, any SDN protocol can be used here. In our MPResiSDN system, we utilized the OpenFlow protocol. The rule is defined as an OpenFlow flow entry. Its abstract format is shown in Figure 6.

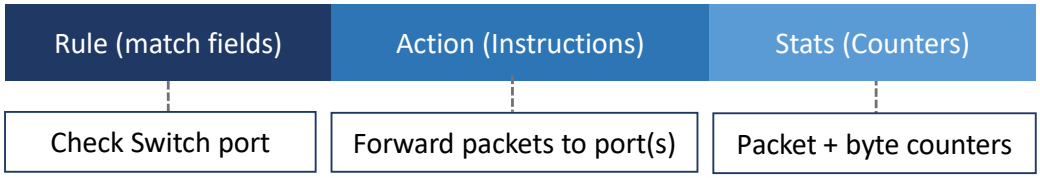


Figure 6. SDN OpenFlow flow entry format.

4.2. System Algorithm

In this section, the MPResiSDN algorithm is described. the algorithm has seven functions: TopologyDiscovery(), ChallengeDetector(), DiversePaths(k, src, dst, G), FindSwitches($path$), DetermineInOutPorts(sw, $path$), ReversePath($path$), and GenerateRule(sw, in, out). The pseudocode of the MPResiSDN algorithm is illustrated in Algorithm 1.

Initially, TopologyDiscovery() determines the network topology. The function results in a set of nodes N, links L, and a graph $G = (N, L)$. The ChallengeDetector() function reads the sensory information from the lightning, humidity, and temperature sensors. It analyses the results to detect the existence of natural challenges and specifies the type of challenge. Most importantly, this function specifies a suitable value for the number of alternative paths to be used, (k).

The DiversePaths(k, src, dst, G) function takes the specified value k and the determined topology graph G to compute k diverse paths between a source node src and a destination sink dst. The FindSwitches($path$) function takes a path as input and determines all connected switches. The DetermineInOutPorts(sw, $path$) function takes a switch ID and the path it belongs to as inputs. Then, it determines the connected input and output ports for each switch in the path.

The ReversePath($path$) function takes a $path$ as input and determines the reverse path associated with it. Finally, the GenerateRule(sw, in, out) function generates the SDN rules to be pushed to the OpenFlow switches. It takes a switch ID along with its associated input and output ports as inputs. Then it produces an SDN rule to be used later to fill the switch's routing table.

The code flow for the MPResiSDN algorithm is described here. At the initialization level, the network topology G is obtained by the TopologyDiscovery() function and passed to the DiversePaths(k, src, dst, G) function. The parameter k is determined by the ChallengeDetector() function and passed to the DiversePaths(k, src, dst, G) function. Then, the DiversePaths(k, src, dst, G) function immediately generates the k diverse paths to be used. Subsequently, the SDN rules list is initialized, and the process begins of generating the SDN rules to enable the system to utilize the determined k paths. At first, the algorithm takes each path in turn and applies several processing steps. For each path, the algorithm determines all switches on the path using the FindSwitches($path$) function. Then, for each found switch, the input and output ports are determined using the DetermineInOutPorts(sw, $path$) function and the associated SDN rule is generated using the GenerateRule(sw, in, out) function. Eventually, the SDN rule is appended to the SDN rules list.

SDN rules are created to handle the acknowledgment packets. The ReversePath($path$) function is used to generate the reversed path of the original forward path. Then, the same

processing steps that were applied to the forward path are applied to the reversed path. The generated rules list is utilized to fill the routing tables of all switches belonging to the path. Note that all switches use the resulting routing tables until an event happens, e.g., a new challenge is detected, or a link or a node is damaged. When any of these events happens, the SDN controller reruns the algorithm and updates the routing tables of all switches. To consider the algorithmic complexity of our algorithm, we focus on expensive functions and loops. Hence, to compute k shortest paths, our algorithm uses the Floyd-Warshall algorithm to find diverse paths, which incur $O(n^2)$ with a network of size n nodes [57]. In addition, the algorithm uses two nested four loops to construct rules, which incur a total algorithmic complexity of $O(ps)$, where p represents the number of paths and s represents the number of switches inside a path. Therefore, our algorithm's complexity is $O(n^2 + ps)$, which can be simplified to $O(n^2)$ since p and s cannot be larger than v.

Algorithm 1: MPResiSDN algorithm.

Functions:
TopologyDiscovery(): determines the network topology as a graph G
ChallengeDetector(): decides the value of k based on environmental sensors.
DiversePaths(k, src, dst, G): determines k paths between source src and destination dst in a graph G.
FindSwitches($path$): generates a list of all switches in a given path.
DetermineInOutPorts(sw, $path$): determines input and output ports for a switch sw in a given $path$.
ReversePath($Path$): determines the opposite direction of a path.
GenerateRule(sw, in , out): generates an SDN rule for a switch sw, given ports in and out.
Input:
src: source node.
dst: destination node.
Output:
$RulesList$: SDN rules.

```
begin
    G = TopologyDiscovery()
    k = ChallengeDetector()
    Paths = DiversePaths(src,dst,G,k)
    SDNRules = []
    for path in Paths do
        for sw in FindSwitches(path) do
            in , out = DetermineInOutPorts(sw, path)
            SDNRule = GenerateRule(sw, in , out)
            RulesList.append(sw,SDNRule)
        end
        reversedPath = ReversePath(path)
        for sw in FindSwitches(reversedPath) do
            in , out = DetermineInOutPorts(sw, path)
            SDNRule = GenerateRule(sw, in , out)
            RulesList.append(sw,SDNRule)
        end
    end
    return RulesList
end
```

5. Evaluation Methodology

In this section, the evaluation environment is described, i.e., the network topology, applied challenges, and experimental setup. The performance metrics used to evaluate the system are defined.

5.1. Network Topology

The topology of the network used to evaluate the system is shown in Figure 7. The network is composed of four elements, a surveillance camera, intermediate nodes, the Integrated Command and Control Center (ICCC), and wireless links.

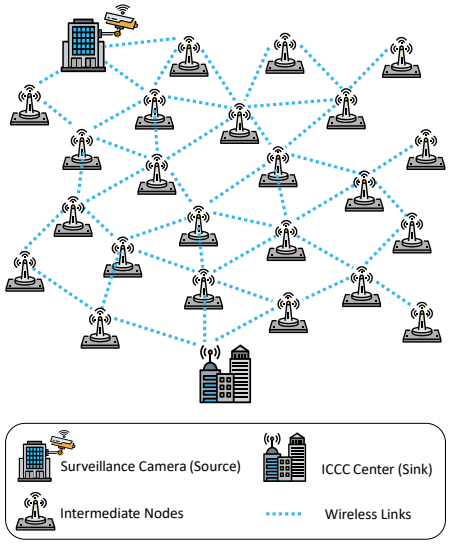


Figure 7. Evaluation network topology.

The surveillance camera generates information. The intermediate nodes deliver information to the network sink, i.e., the ICCC. The ICCC is responsible for collecting sensed information. The wireless links transmit information between nodes. All nodes in the network topology are connected to OpenFlow switches. Each OpenFlow switch is connected to a remote controller that is running the MPResiSDN algorithm. Figure 8 illustrates the three diverse paths that are used in the evaluation scenario.

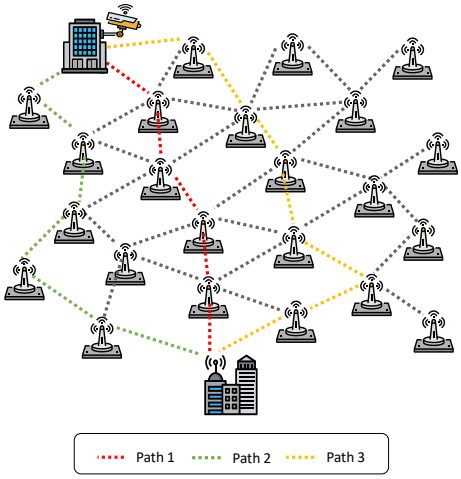


Figure 8. Paths used for evaluation.

5.2. Natural Challenges

To evaluate the system, two different types of natural challenges were modeled and applied; a fire challenge and a storm challenge. Figures 9 and 10 illustrate the scenarios in which each challenge was applied.

The fire challenge was modeled to create permanent damage to all nodes it affects. The details of how it acted during the experiment are shown in Figure 9. Initially, the fire

challenge was small; however, it expanded until it started to hit some intermediate nodes at time $t = 30$. Over time, the fire continued to expand and affect additional nodes.

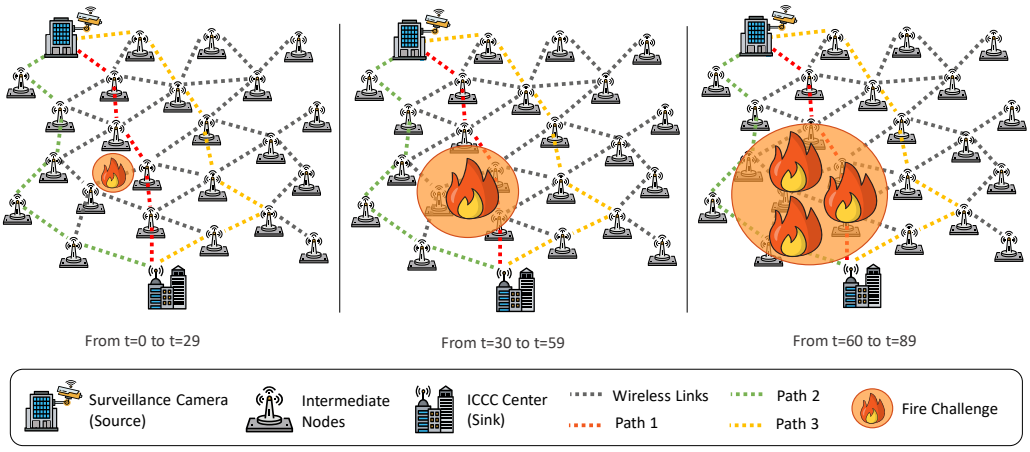


Figure 9. Fire challenge scenario.

The storm challenge was modeled to create a temporary effect on the nodes; specifically, to cause noise that affects the communications to and from the nodes it covers. Once the storm moves out, the nodes resume normal operation. Figure 10 illustrates how the modeled storm challenge acted during the experiment. At first, the storm only affected a single node on path 2. Then, it moved in a northeasterly direction. As it was moving, more nodes on path 1 and path 3 were affected, and the nodes on path 2 gradually resumed their normal operation. At $t = 50$, the storm was about to leave the sensing area, allowing nodes on all three paths to resume their normal operation.

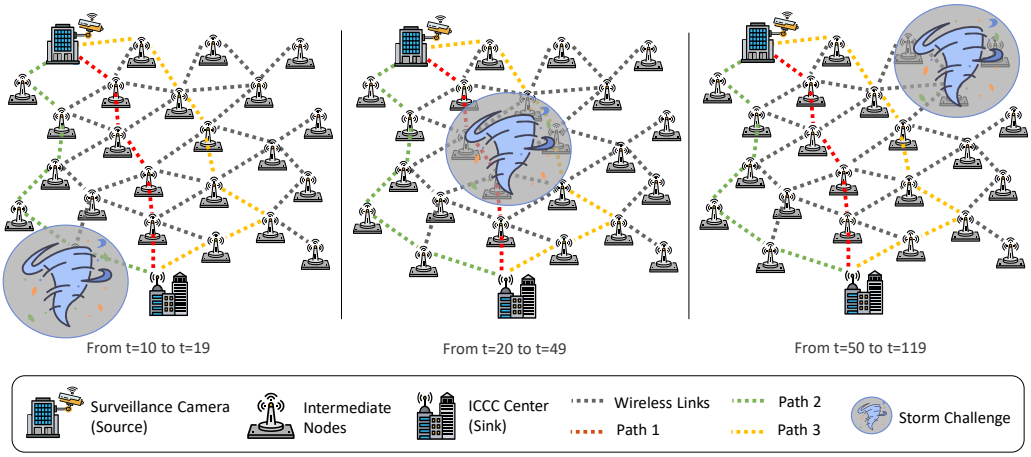


Figure 10. Storm challenge scenario.

5.3. Performance Metrics

The system was evaluated under three main cases: normal operation, a storm challenge, and a fire challenge. For each case, four performance metrics were used, i.e., actual throughput, goodput, overhead, and end-to-end delay. Actual throughput represents all network traffic that is received at the destination. Goodput represents only actual meaningful data, excluding any redundancy. Overhead represents the percentage of the goodput over the actual throughput, and end-to-end delay is the time the system takes to deliver information from the source to the sink.

5.4. Experimental Setup

All experiments were carried out using Ubuntu MATE 18.04, with 2 GB of RAM and a 1.90 GHz processor. The networks were emulated using Mininet 2.2.2 interfaces with Python 2.7.17. The challenges were applied by configuring links losses. The RYU controller was used to implement the multipath algorithm. Data traffic was generated using iPerf3 and captured using tcpdump. The results were analyzed using Wireshark and Python Pandas. The emulation parameters are listed in Table 1.

Table 1. Emulation Parameter Values.

Parameter	Value
Operating System	Ubuntu 18.4
Memory	2 GB of RAM
CPU	1.90 GHZ
Emulator	Mininet 2.2.2
SDN Framework	RYU Controller
Programming Language	Python 2.7.17
Traffic Generator	iPerf3
Link Bandwidth	10 Mbps

6. Results and Discussion

In this section, the results of applying the evaluation methodology described in Section 5 to our proposed MPResiSDN scheme are presented and discussed. The MPResiSDN scheme was evaluated using different k values; ($k = 1$), ($k = 2$), and ($k = 3$). We refer to the results as MPResiSDN($k = 1$), MPResiSDN($k = 2$), and MPResiSDN($k = 3$). For execution time, the algorithm performance is evaluated with different network sizes and numbers of paths. Our execution time evaluation is shown in Figure 11 and the results show that execution time is not significantly high, which is approximately 0.003 ms for a network with 100 nodes with MPResiSDN($k = 1$) and 0.008 ms for a network with 100 nodes with MPResiSDN($k = 2$) and MPResiSDN($k = 3$). For comparison, the same evaluation methodology was applied to the same evaluation network topology, using the traditional Spanning Tree scheme that is integrated in the RYU controller. For all four schemes, the evaluation was made under three cases: normal operation, fire challenges, and storm challenges. The results for each case are presented and discussed individually in the following subsections.

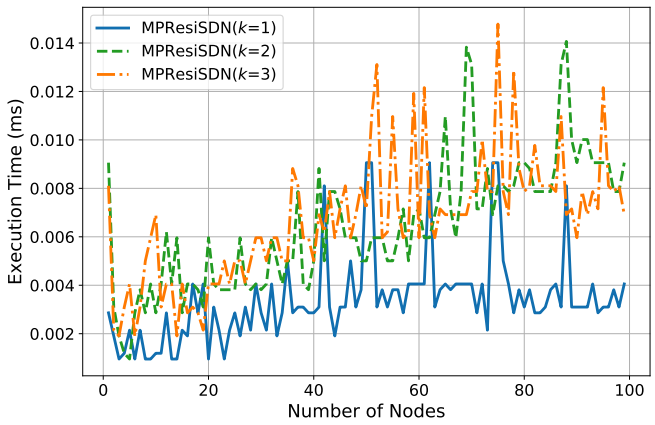
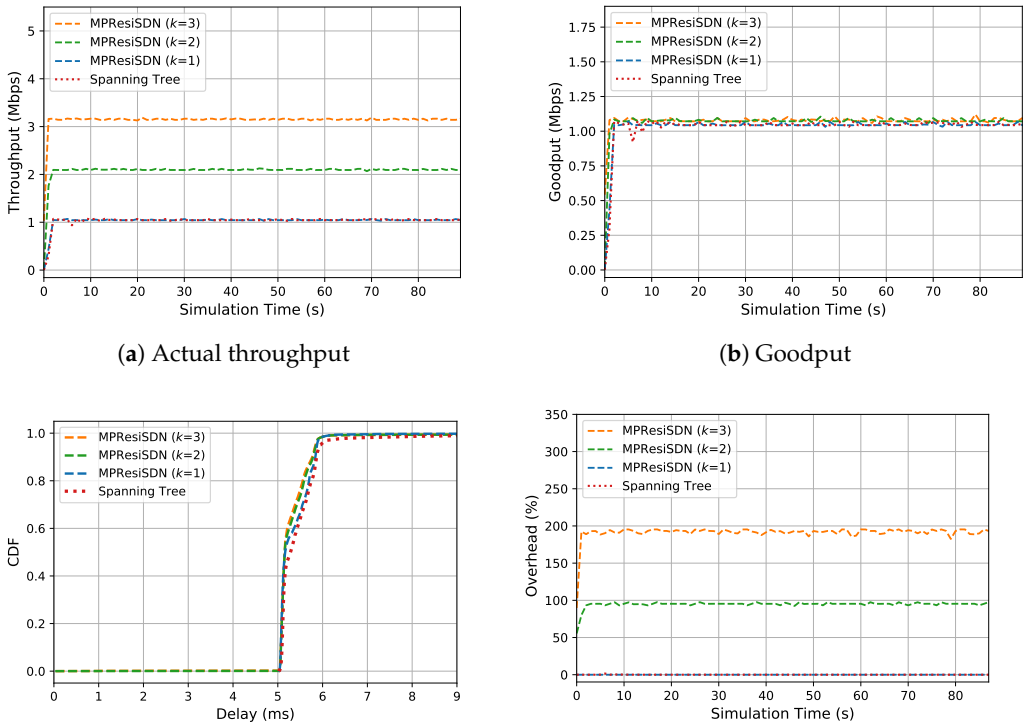


Figure 11. Evaluation of execution time using different k values and network sizes.

6.1. Normal Operation

First, we look at the result of measuring the actual throughput produced by each of the four schemes when tested under normal operation, where no challenge is presented. The results are shown in Figure 12a. As can be seen, MPResiSDN($k = 3$) produces triple the throughput produced by MPResiSDN($k = 1$), while MPResiSDN($k = 2$) produces twice the throughput. This is because MPResiSDN($k = 2$) and MPResiSDN($k = 3$) use additional paths to deliver data to the sink, which obviously increases the actual throughput.MPResiSDN($k = 1$) and Spanning Tree produce the same amount of data; however, Spanning Tree takes a much longer time to discover the topology before it starts to produce throughput.



(**a**) Actual throughput (**b**) Goodput

(**c**) End-to-end delay (**d**) Overhead

Figure 12. Comparison of the four strategies under normal operation.

Now, we look at the results of measuring the goodput produced by the four schemes evaluated in the case of no challenge (Figure 12b). Differing from the actual throughput, the goodput of all schemes is approximately the same. These results are expected given that the goodput filters out all redundant data and only keeps one copy of all required packets.

Figure 12c shows the cumulative distribution functions (CDF) of the delay of each scheme under normal operation. For the most part, the delay values range between 5 and 6 ms, regardless of the schemes used. It is evident that increasing the number of alternative paths used in MPResiSDN does not affect the delivery time of the information when the environment does not contain a challenge.

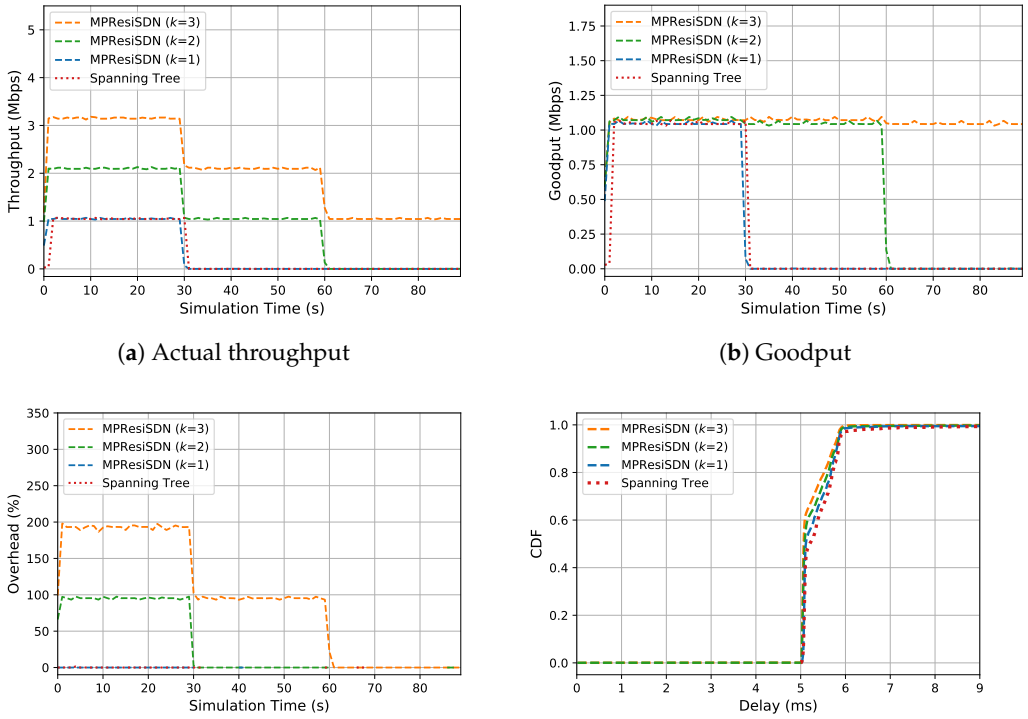
The difference in the actual throughput and goodput results when tested under normal operation leads us to look at the actual overhead caused by the four schemes (Figure 12d). The overhead is calculated as a percentage of the actual throughput divided by the goodput, which represents the redundancy in the data. MPResiSDN($k = 3$) results in an increase in overhead of up to 200%, while using MPResiSDN($k = 2$) results in an increase of nearly 100%.

As can be seen, using additional paths to deliver data under normal operational conditions when no challenge is presented creates massive overhead without providing any

benefits. Thus, it is crucial to consistently use only one path to deliver data under normal operation. This result reinforces the importance of the Challenge Detector component. When the detector reports no challenge, the MPResiSDN should use $k = 1$ to avoid unnecessary overhead on the sensor network.

6.2. Fire Challenges

The results of evaluating the MPResiSDN scheme under the modeled fire challenge illustrated in Figure 9, using the actual throughput performance metric, are shown in Figure 13a. According to the fire challenge scenario, in the time interval between $t = 0$ and $t = 30$ the fire was too small to affect any sensor node. Consequently, the actual throughput in these time intervals was the same as the actual throughput under normal operating conditions.



(**a**) Actual throughput (**b**) Goodput

(**c**) End-to-end delay (**d**) Overhead

Figure 13. Comparison of the four strategies under the fire challenge.

However, at $t = 30$, the fire expanded and began to affect the sensor nodes. In particular, some of the sensor nodes on path 1 were damaged by the fire. Note that Path 1 is the only path used in the MPResiSDN($k = 1$) scheme, whereas it is one of the selected paths in MPResiSDN($k = 2$)) and MPResiSDN($k = 3$) schemes. Accordingly, the actual throughput produced by MPResiSDN($k = 1$) is zero at $t = 30$, while the throughput produced by the MPResiSDN($k = 2$) and MPResiSDN($k = 3$) schemes is reduced.

At $t = 60$, the fire had expanded and affected both paths 1 and 2. Consequently, the throughput produced by the MPResiSDN($k = 2$) scheme under normal conditions was completely obstructed, because all paths used by that scheme were blocked. The MPResiSDN($k = 3$) scheme continued to function because it had more alternative paths, including path 3, which was not affected by the fire challenge at this time. In this case, the importance of implementing an algorithm with more alternative paths, such as MPResiSDN($k = 3$) is evident.

While the actual throughput shows how each scheme behaves in the face of a certain challenge, the goodput shows whether that behavior ultimately succeeded in delivering the required data. Figure 13b shows the detailed results of measuring the goodput over time

under the modeled fire challenge. As expected, all schemes worked well at the beginning because none of the paths were affected by the fire challenge. However, when the fire challenge started to hit path 1, which is the only path used in the MPResiSDN($k = 1$) and Spanning Tree schemes, both algorithms failed to deliver data. MPResiSDN($k = 2$) and MPResiSDN($k = 3$) performed well until the fire started to hit path 2 at $t = 60$. At this time, MPResiSDN($k = 2$) failed because both of the paths it uses failed. MPResiSDN($k = 3$) continued to work well as it had an additional alternative path to use. The Spanning Tree algorithm struggled after the fire started to impact any path because it takes a very long time to figure out a new path to use.

Figure 13c shows the details of the overhead generated by the four schemes under the fire challenge over time. Given that MPResiSDN(k = 3) uses two additional alternative paths to deliver data, it takes approximately twice the amount of throughput compared to MPResiSDN($k = 1$) when no challenge is presented at the beginning. In other words, MPResiSDN($k = 3$), under no challenge, results in an overhead of 200% of the actual amount of data produced by MPResiSDN($k = 1$). Similarly, MPResiSDN($k = 2$) generates double the actual throughput produced by MPResiSDN($k = 1$). MPResiSDN($k = 1$) and Spanning Tree deliver the data without any additional overhead. These results confirm the finding mentioned previously when discussing the normal operation case, i.e., alternative paths are only worthwhile under challenges.

However, when the fire starts to affect the path used by MPResiSDN($k = 1$) and Spanning Tree schemes at $t = 30$, they fail completely. Consequently, no traffic is generated; thus, there is no overhead. At this point, MPResiSDN($k = 2$) works the same as MPResiSDN($k = 1$) under normal operation, while MPResiSDN($k = 3$) works the same as MPResiSDN($k = 2$) under normal operation.

At $t = 60$, when the fire starts to affect path 2, the only scheme that could handle this challenge was MPResiSDN($k = 3$) because it has a third path to use to deliver data. No overhead is generated by MPResiSDN($k = 3$) at this point because it works with only one path. MPResiSDN($k = 2$) and MPResiSDN($k = 1$) fail completely, and Spanning Tree continues to fail because it takes an exceptionally long time to find the new shortest path. Finding a new path becomes increasingly difficult for Spanning Tree due to rapid changes in network topology. In this case, obviously no overhead is generated as no data is being delivered.

Figure 13d shows the CDFs of the delays incurred by each of the four schemes when evaluated under the fire challenge. As can be seen, the delay results here are like the delay results of the normal operation case. The delay values that are produced by all schemes during the entire time interval are almost constant. They primarily vary between 5 and 6 ms. As has been noted, regardless of the type of scheme or the challenge status, the delay values are not affected.

In conclusion, using alternative paths definitely helped handle the fire challenge. Moreover, it is evident that increasing the number of alternative paths played a major role in increasing the resilience of routing schemes against the fire challenge. Differing from the case under no challenge, the generated overhead is not worthless.

6.3. Storm Challenges

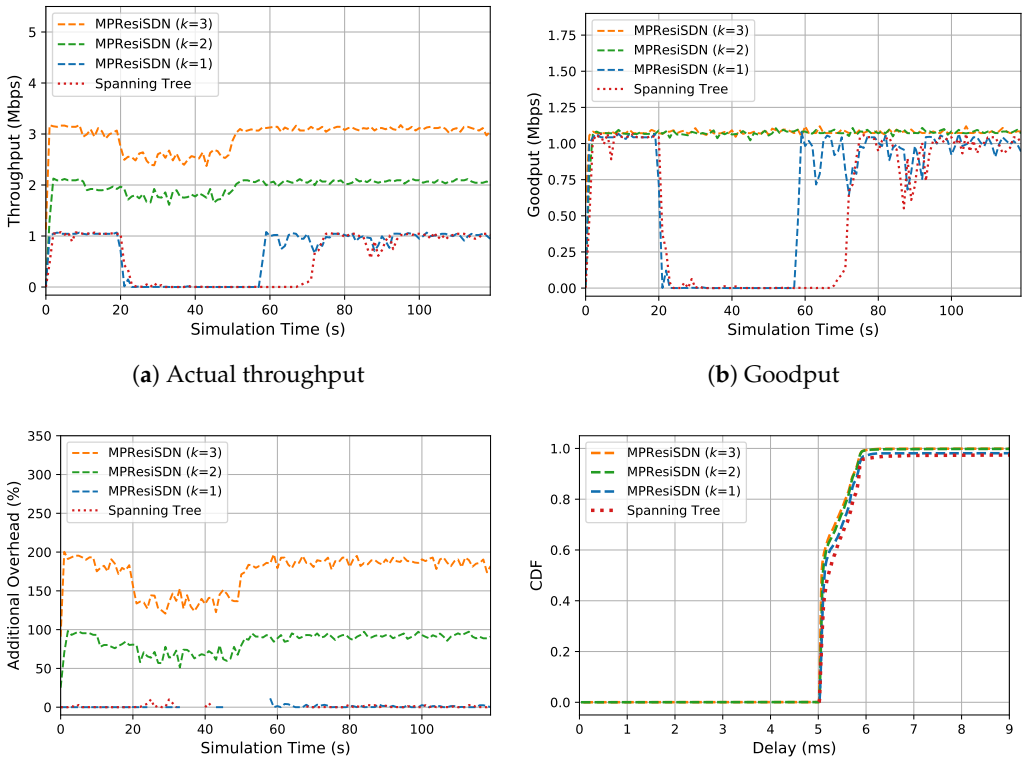
Here, we look at the results of evaluating the four schemes under the storm challenge model, using the four performance metrics. Figure 14a shows the actual throughput produced by the schemes under the storm challenge, which is modeled in Figure 10. The throughputs for all four schemes are at their highest values in the first 10 s because the storm has not yet started.

From $t = 10$ to $t = 20$, the storm began to affect path 2, which is used by MPResiSDN($k = 2$) and MPResiSDN($k = 3$). The storm challenge increases noise, which affects the sensor nodes impacted by the storm. This is reflected in a slight effect on the throughput of both MPResiSDN($k = 2$) and MPResiSDN($k = 3$). At this time, the Spanning

Tree and MPResiSDN($k = 1$) are not affected because they use path 1, which is not yet covered by the storm challenge.

The worst damage caused by the storm challenge occurs from $t = 20$ to $t = 49$. Although path 2 is freed as the storm moves away, it begins to affect both paths 1 and 3. As path 1 is the only path used by MPResiSDN($k = 1$) and Spanning Tree, the throughput dramatically decreases at that time. However, MPResiSDN($k = 2$) and MPResiSDN($k = 3$) can successfully handle the challenge because they can use an alternative path. As shown in Figure 14a, the throughput produced by the MPResiSDN($k = 2$) and MPResiSDN($k = 3$) schemes is reduced; however, they still maintain an acceptable level of performance.

At $t = 50$, the storm begins to leave the affected area. As the storm leaves, no paths are affected. This is immediately reflected by an increase in the actual throughput of the MPResiSDN($k = 2$) and MPResiSDN($k = 3$) schemes. MPResiSDN($k = 1$) takes longer to recover and the Spanning Tree scheme takes even more time.



(**a**) Actual throughput (**b**) Goodput

(**c**) Overhead (**d**) End-to-end delay

Figure 14. Comparison of the four strategies under the storm challenge.

The evaluation results for the four schemes under the storm challenge using the goodput performance metric are shown in Figure 14b. Initially, goodput is not affected by the storm challenge as the challenge has not affected a wide area. When the challenge affects a larger area and covers more paths, the goodput of the MPResiSDN($k = 1$) and Spanning Tree schemes drops dramatically because these schemes do not use alternative paths. MPResiSDN($k = 2$) and MPResiSDN($k = 3$) schemes can handle the challenge and maintain approximately the same level of goodput. They are not affected by the storm challenge because they always have alternative paths to use. When the storm starts to leave the area, the single path used by MPResiSDN($k = 1$) and Spanning Tree becomes available. Thus, these schemes start to work again. Note that the Spanning Tree scheme requires more time to recover from the challenge.

The results of evaluating the four schemes under the storm challenge using the overhead metric are shown in Figure 14c. In the first 10 s, the overhead results under the storm challenge are similar to those under normal operation because the storm has not

started yet. From $t = 10$ to $t = 19$, the storm starts to cover path 2, resulting in noise that affects communication from and to the sensor nodes on the path. The noise slightly reduces the overhead of MPResiSDN($k = 2$) and MPResiSDN($k = 3$) schemes because it causes dropped packets on path 2. However, packets on path 2 are all duplicates; therefore, dropping some of them reduces overhead rather than goodput. The goodput is not affected for paths 1 and 3 because they are both completely out of the challenge. Spanning Tree and MPResiSDN($k = 1$) schemes are not affected at this time because path 1 is free.

From $t = 20$ to $t = 49$, the storm challenge moves away. Path 2 is no longer affected by the storm. However, path 1 and path 3 are affected. This is reflected in a decrease in the overhead of MPResiSDN($k = 2$) and MPResiSDN($k = 3$) in the same manner that has been described previously. At the same time, the Spanning Tree and MPResiSDN($k = 1$) schemes only use path 1, which is currently under the storm challenge. The sensor nodes on path 1 are affected by random noise caused by the storm challenge. The noise flips some of the bits in the packets, causing them to be dropped by the link layer. This causes a severe problem for the Spanning Tree and MPResiSDN($k = 1$) schemes because they do not have alternative paths to use. As a result, the overhead sometimes goes to zero because no throughput or goodput are generated. At other times, overhead increases because duplicates are generated due to re-transmissions.

The results of the delay under the storm challenge are similar to the previous results for delays under the fire challenge and normal operation. Figure 14d shows the CDFs of the delays for each of the four schemes under the storm challenge. The delay varies between 5 and 6 ms. This indicates that there is no relationship between using alternative paths and the delay in delivering the data. In other words, using alternative paths does not affect the delay.

6.4. Evaluation Summary

In this section, we summarize our findings after studying the performance results under normal operating conditions and under the challenges. We found that, under the challenges, the proposed MPResiSDN scheme improved data delivery in terms of the obtained goodput by up to 100% compared to Spanning Tree when a suitable value for k diverse paths was selected. However, to maintain this enhancement, the value of k should be increased as the severity of the challenge increases in terms of the number of nodes it affects and whether the affected nodes belong to the used paths. However, selecting a value larger than the required value of k results in massive unnecessary overhead, i.e., overhead can increase by up to 200% or more. Thus, under normal operation without any challenge, selecting a value that is greater than 1 for k diverse paths is completely inefficient because overhead increases without providing any observable benefit.

Considering the effect of the proposed MPResiSDN scheme on end-to-end delay, it is evident that the end-to-end delay in the running phase is not affected. However, the proposed system outperformed Spanning Tree in terms of response time initially when the topology was identified; topology identification is faster with the proposed scheme. Moreover, the proposed system outperforms Spanning Tree when it comes to response time to changes in network topology.

7. Conclusions

In smart cities, maintaining a resilient network is crucial. The proposed MPResiSDN system improved network resilience by exploiting SDN capabilities to implement multipath routing in case of challenges. The system was evaluated under normal operation and under two different types of natural challenges. The evaluation results demonstrate that MPResiSDN routing improved data delivery under challenges by up to 100% compared to Spanning Tree when a suitable value for k diverse paths was selected. In addition, the results show that fewer paths are required to confront a relatively trivial challenge, such as a storm challenge, while more paths are needed to confront a more severe challenge, such as a fire challenge. However, increasing the number of paths increases overhead;

thus, to avoid unnecessary overhead, the number of paths needed to achieve the required resilience should be decided thoughtfully when implementing the system. To clarify, the value of k should be determined based on the required level of resilience and the severity level of the challenge. The more severe the challenge, the more alternative paths should be used, so larger k values are recommended. This emphasizes the importance of the challenge detecting phase. In future, further studies could be conducted to investigate the best ways to determine the optimal value of k based on the type of natural challenge. In addition, other types of natural challenge, such as floods, earthquakes, and volcanoes, could be considered.

Author Contributions: S.L.A. performed the experiments, analyzed the data, and wrote the paper. M.J.F.A. supervised the research and critically revised the paper. Both authors have read and agreed to the published version of the manuscript.

Funding: The authors extend their appreciation to the Deanship of Scientific Research at King Saud University for funding this work through research group No (RG-1441-512).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors thank the Deanship of Scientific Research and RSSU at King Saud University for their technical support.

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

References

1. *World Urbanization Prospects: The 2018 Revision*; United Nations: San Francisco, CA, USA, 2018; pp. 101–103.
2. *World Population Prospects 2019*; Demographic Profiles; United Nations: San Francisco, CA, USA, 2019; Volume 2, p. 1214. Available online: https://population.un.org/wpp/ (accessed on 19 February 2020).
3. Yin, C.; Xiong, Z.; Chen, H.; Wang, J.; Cooper, D.; David, B. A literature survey on smart cities. *Sci. China Inf. Sci.* **2015**, *58*. [CrossRef]
4. Berrone, P.; Ricart, J.E.; Carrasco, C.; Duch, A. *IESE Cities in Motion Index 2018*; Technical Report, ST-509-E; IESE: Barcelona, Spain, 2019. Available online: https://media.iese.edu/research/pdfs/ST-0509-E (accessed on 11 June 2020).
5. Silva, B.N.; Khan, M.; Han, K. Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities. *Sustain. Cit. Soc.* **2018**, *38*, 697–713. [CrossRef]
6. Talari, S.; Shafie-Khah, M.; Siano, P.; Loia, V.; Tommasetti, A.; Catalão, J.P. A review of smart cities based on the internet of things concept. *Energies* **2017**, *10*, 421. [CrossRef]
7. Nandury, S.V.; Begum, B.A. Smart WSN-based ubiquitous architecture for smart cities. In Proceedings of the 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Kochi, India, 10–13 August 2015; pp. 2366–2373.
8. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
9. Du, R.; Santi, P.; Xiao, M.; Vasilakos, A.V.; Fischione, C. The sensable city: A survey on the deployment and management for smart city monitoring. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 1533–1560. [CrossRef]
10. Gharaibeh, A.; Salahuddin, M.A.; Hussini, S.J.; Khreishah, A.; Khalil, I.; Guizani, M.; Al-Fuqaha, A. Smart cities: A survey on data management, security, and enabling technologies. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2456–2501. [CrossRef]
11. Arasteh, H.; Hosseinnezhad, V.; Loia, V.; Tommasetti, A.; Troisi, O.; Shafie-khah, M.; Siano, P. Iot-based smart cities: A survey. In Proceedings of the 2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC), Florence, Italy, 7–10 June 2016; pp. 1–6.
12. Rashid, B.; Rehmani, M.H. Applications of wireless sensor networks for urban areas: A survey. *J. Netw. Comput. Appl.* **2016**, *60*, 192–219. [CrossRef]
13. Aljohani, S.L.; Alenazi, M.J. Evaluation of WSN's Resilience to Challenges in Smart Cities. *Int. J. Comput. Commun. Eng.* **2020**, *9*, 193–206. [CrossRef]
14. Çetinkaya, E.K.; Broyles, D.; Dandekar, A.; Srinivasan, S.; Sterbenz, J.P. Modelling communication network challenges for future internet resilience, survivability, and disruption tolerance: A simulation-based approach. *Telecommun. Syst.* **2013**, *52*, 751–766. [CrossRef]
15. Yaghoubi, F.; Furdek, M.; Rostami, A.; Öhlén, P.; Wosinska, L. Resilient SDN-Based Routing Against Rain Disruptions for Wireless Networks. In *Guide to Disaster-Resilient Communication Networks*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 507–522.

16. Alablani, I.; Alenazi, M. EDTD-SC: An IoT Sensor Deployment Strategy for Smart Cities. *Sensors* **2020**, *20*, 7191. [CrossRef] [PubMed]
17. Gao, J.; Barzel, B.; Barabási, A.L. Universal resilience patterns in complex networks. *Nature* **2016**, *530*, 307–312. [CrossRef] [PubMed]
18. AlZoman, R.; Alenazi, M.J. Exploiting SDN to Improve QoS of Smart City Networks Against Link Failures. In Proceedings of the 2020 Seventh International Conference on Software Defined Systems (SDS), Paris, France, 20–23 April 2020; pp. 100–106.
19. Bawany, N.Z.; Shamsi, J.A. Smart city architecture: Vision and challenges. *Int. J. Adv. Comput. Sci. Appl.* **2015**, *6*. [CrossRef]
20. Obaidat, M.S.; Misra, S. *Principles of Wireless Sensor Networks*; Cambridge University Press: Cambridge, UK, 2014.
21. Alablani, I.; Alenazi, M. Performance Evaluation of Sensor Deployment Strategies in WSNs Towards IoT. In Proceedings of the 2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates, 3–7 November 2019; pp. 1–8.
22. Fundation, O.N. Software-defined networking: The new norm for networks. *ONF White Pap.* **2012**, *2*, 11.
23. Jarschel, M.; Zinner, T.; Hoßfeld, T.; Tran-Gia, P.; Kellerer, W. Interfaces, attributes, and use cases: A compass for SDN. *IEEE Commun. Mag.* **2014**, *52*, 210–217. [CrossRef]
24. Kreutz, D.; Ramos, F.M.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proc. IEEE* **2014**, *103*, 14–76. [CrossRef]
25. Alenazi, M.J.; Almutairi, A.; Almowuena, S.; Wadood, A.; Çetinkaya, E.K. NFV Provisioning in Large-Scale Distributed Networks With Minimum Delay. *IEEE Access* **2020**, *8*, 151753–151763. [CrossRef]
26. Sahoo, K.S.; Mohanty, S.; Tiwary, M.; Mishra, B.K.; Sahoo, B. A comprehensive tutorial on software defined network: The driving force for the future internet technology. In *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*; ACM: New York, NY, USA, 2016; pp. 1–6.
27. Sahoo, K.S.; Sahoo, B.; Panda, A. A secured SDN framework for IoT. In Proceedings of the 2015 International Conference on Man and Machine Interfacing (MAMI), Odisha, India, 17–19 December 2015; pp. 1–4.
28. Sterbenz, J.P. Smart city and IoT resilience, survivability, and disruption tolerance: Challenges, modelling, and a survey of research opportunities. In Proceedings of the 2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM), Alghero, Italy, 4–6 September 2017; pp. 1–6.
29. Smith, P.; Hutchison, D.; Sterbenz, J.P.; Schöller, M.; Fessi, A.; Karaliopoulos, M.; Lac, C.; Plattner, B. Network resilience: A systematic approach. *IEEE Commun. Mag.* **2011**, *49*, 88–97. [CrossRef]
30. Sterbenz, J.P.; Hutchison, D.; Çetinkaya, E.K.; Jabbar, A.; Rohrer, J.P.; Schöller, M.; Smith, P. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Comput. Netw.* **2010**, *54*, 1245–1265. [CrossRef]
31. Bawany, N.Z.; Shamsi, J.A. Application layer DDoS attack defense framework for smart city using SDN. In Proceedings of the The Third International Conference on Computer Science, Computer Engineering, and Social Media (CSCESM2016), Thessaloniki, Greece, 13–15 May 2016; p. 1.
32. Bawany, N.Z.; Shamsi, J.A. SEAL: SDN based secure and agile framework for protecting smart city applications from DDoS attacks. *J. Netw. Comput. Appl.* **2019**, *145*, 102381. [CrossRef]
33. Chen, W.; Xiao, S.; Liu, L.; Jiang, X.; Tang, Z. A DDoS attacks traceback scheme for SDN-based smart city. *Comput. Electr. Eng.* **2020**, *81*, 106503. [CrossRef]
34. Xu, C.; Lin, H.; Wu, Y.; Guo, X.; Lin, W. An SDNFV-based DDoS defense technology for smart cities. *IEEE Access* **2019**, *7*, 137856–137874. [CrossRef]
35. Han, T.; Ge, X.; Wang, L.; Kwak, K.S.; Han, Y.; Liu, X. 5G converged cell-less communications in smart cities. *IEEE Commun. Mag.* **2017**, *55*, 44–50. [CrossRef]
36. Usman, M.; Gebremariam, A.A.; Raza, U.; Granelli, F. A software-defined device-to-device communication architecture for public safety applications in 5G networks. *IEEE Access* **2015**, *3*, 1649–1654. [CrossRef]
37. Bi, Y.; Lin, C.; Zhou, H.; Yang, P.; Shen, X.; Zhao, H. Time-constrained big data transfer for SDN-enabled smart city. *IEEE Commun. Mag.* **2017**, *55*, 44–50. [CrossRef]
38. Khan, M.; Iqbal, J.; Talha, M.; Arshad, M.; Diyan, M.; Han, K. Big data processing using internet of software defined things in smart cities. *Int. J. Parallel Program.* **2020**, *48*, 178–191. [CrossRef]
39. Cui, X.; Huang, X.; Ma, Y.; Meng, Q. A load balancing routing mechanism based on SDWSN in smart city. *Electronics* **2019**, *8*, 273. [CrossRef]
40. Rametta, C.; Baldoni, G.; Lombardo, A.; Micalizzi, S.; Vassallo, A. S6: A Smart, Social and SDN-based Surveillance System for Smart-cities. *Procedia Comput. Sci.* **2017**, *110*, 361–368. [CrossRef]
41. Kunst, R.; Avila, L.; Pignaton, E.; Bampi, S.; Rochol, J. Improving network resources allocation in smart cities video surveillance. *Comput. Netw.* **2018**, *134*, 228–244. [CrossRef]
42. Raja, G.; Dhanasekaran, P.; Anbalagan, S.; Ganapathisubramaniyan, A.; Bashir, A.K. SDN-enabled Traffic Alert System for IoV in Smart Cities. In Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Online, 6–9 July 2020; pp. 1093–1098.
43. Abbas, M.T.; Muhammad, A.; Song, W.C. SD-IoV: SDN enabled routing for internet of vehicles in road-aware approach. *J. Ambient. Intell. Humaniz. Comput.* **2020**, *11*, 1265–1280. [CrossRef]

44. Bhatia, J.; Dave, R.; Bhayani, H.; Tanwar, S.; Nayyar, A. Sdn-based real-time urban traffic analysis in vanet environment. *Comput. Commun.* **2020**, *149*, 162–175. [CrossRef]
45. Rego, A.; Garcia, L.; Sendra, S.; Lloret, J. Software defined networks for traffic management in emergency situations. In Proceedings of the 2018 Fifth International Conference on Software Defined Systems (SDS), Barcelona, Spain, 23–26 April 2018; pp. 45–51.
46. Rego, A.; Garcia, L.; Sendra, S.; Lloret, J. Software Defined Network-based control system for an efficient traffic management for emergency situations in smart cities. *Future Gener. Comput. Syst.* **2018**, *88*, 243–253. [CrossRef]
47. García, L.; Parra, L.; Taha, M.; Lloret, J. System for detection of emergency situations in smart city environments employing smartphones. In Proceedings of the 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Bangalore, India, 19–22 September 2018; pp. 266–272.
48. Fragkos, G.; Tsiropoulou, E.E.; Papavassiliou, S. Disaster management and information transmission decision-making in public safety systems. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019; pp. 1–6.
49. Huang, X.L.; Ma, X.; Hu, F. Machine learning and intelligent communications. *Mob. Netw. Appl.* **2018**, *23*, 68–70. [CrossRef]
50. Ahmed, A.A. A Lightweight Software Defined Network for Resilient Real-time Internet of Things. *IJCSNS* **2019**, *19*, 1.
51. Alenazi, M.J. Evaluating Multipath TCP Resilience against Link Failures. *ISeCure* **2019**, *11* . [CrossRef]
52. Singh, P.K.; Sharma, S.; Nandi, S.K.; Nandi, S. Multipath TCP for V2I communication in SDN controlled small cell deployment of smart city. *Veh. Commun.* **2019**, *15*, 1–15. [CrossRef]
53. Ai, J.; Chen, H.; Guo, Z.; Cheng, G.; Baker, T. Improving Resiliency of Software-Defined Networks with Network Coding-based Multipath Routing. In Proceedings of the 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 29 June–3 July 2019; pp. 1–6.
54. Cheng, Y.; Nguyen, T.A.N.; Rahman, M.M.; Gangadhar, S.; Sterbenz, J.P. Cross-layer geodiverse protocol stack for resilient multipath transport and routing using OpenFlow. In Proceedings of the 2016 12th International Conference on the Design of Reliable Communication Networks (DRCN), Paris, France, 15–17 March 2016; pp. 103–105.
55. Rezende, P.; Kianpisheh, S.; Glitho, R.; Madeira, E. An SDN-based framework for routing multi-streams transport traffic over multipath networks. In Proceedings of the ICC 2019-2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
56. Baldoni, G.; Melita, M.; Micalizzi, S.; Rametta, C.; Schembra, G.; Vassallo, A. A dynamic, plug-and-play and efficient video surveillance platform for smart cities. In Proceedings of the 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8-11 January 2017; pp. 611–612.
57. Ojo, A.; Ma, N.W.; Woungang, I. Modified floyd-warshall algorithm for equal cost multipath in software-defined data center. In Proceedings of the 2015 IEEE International Conference on Communication Workshop (ICCW), London UK, 8–12 June 2015; pp. 346–351.