Journal of
Sensor and Actuator Networks

Article

Software Defined Networks in Industrial Automation

Khandakar Ahmed [1,*], Jan O. Blech [2], Mark A. Gregory [3] and Heinz W. Schmidt [2]

1 Discipline of IT, College of Engineering & Science, Victoria University, Footscray Park Campus, Melbourne, VIC 3011, Australia
2 School of Science (Computer Science), RMIT University, Melbourne, VIC 3001, Australia; joblech@gmail.com (J.O.B.); heinz.schmidt@rmit.edu.au (H.W.S.)
3 School of Engineering (Electrical and Computer Systems), RMIT University, Melbourne, VIC 3001, Australia; mark.gregory@rmit.edu.au
* Correspondence: khandakar.e.ahmed@ieee.org

Received: 8 June 2018; Accepted: 2 August 2018; Published: 6 August 2018

Abstract: Trends such as the Industrial Internet of Things and Industry 4.0 have increased the need to use new and innovative network technologies in industrial automation. The growth of industrial automation communications is an outcome of the shift to harness the productivity and efficiency of manufacturing and process automation with a minimum of human intervention. Due to the ongoing evolution of industrial networks from Fieldbus technologies to Ethernet, a new opportunity has emerged to harness the benefits of Software Defined Networking (SDN). In this paper, we provide a brief overview of SDN in the industrial automation domain and propose a network architecture called the Software Defined Industrial Automation Network (SDIAN), with the objective of improving network scalability and efficiency. To match the specific considerations and requirements of having a deterministic system in an industrial network, we propose two solutions for flow creation: the Pro-active Flow Installation Scheme and the Hybrid Flow Installation Scheme. We analytically quantify the proposed solutions that alleviate the overhead incurred from the flow setup. The analytical model is verified using Monte Carlo simulations. We also evaluate the SDIAN architecture and analyze the network performance of the modified topology using the Mininet emulator. We further list and motivate SDIAN features and report on an experimental food processing plant demonstration featuring Raspberry Pi as a software-defined controller instead of traditional proprietary Programmable Logic Controllers. Our demonstration exemplifies the characteristics of SDIAN.

Keywords: controller; industry network; Open Flow; Software Defined Networking; Programmable Logic Controller

1. Introduction

Networking large automated machines is a recent focus for industrial automation and one challenge is the connectivity with traditional automation machinery that is not designed to support more than local computer connectivity. Industrial networks can be highly decentralized, rigid and complex to manage due to the tight coupling of the automation data and control plane that is often embedded within the equipment. The computing and communication nodes are often configured individually when the plant is setup and interconnections remain static thereafter. The traditional industrial communications hierarchical structure consists of three network levels with various networking technologies and protocols that limits what can be achieved and adds complexity due to localized configuration. The traditional structure requires offline manual network control and management, which is time-consuming, error-prone and introduces complexity. It hinders the ability to make live changes to the configuration and feature set as the production line is shifted from one task to another. The resolution of medium access control (MAC) address and virtual routing address during data forwarding in an industrial network can lead to challenges, including integrating software and devices from different vendors.

Legacy industrial communications is a challenge to be overcome as part of the fourth-generation industry revolution (FGIR). FGIR is underpinned by the principle of intelligent manufacturing (IM) enabling customized production. To ensure that a smooth transition occurs between production tasks, IM aims to reconstruct the industrial plant by decoupling the manufacturing entities. To attain optimal production, a goal of FGIR is to utilize live monitoring of machine status, environmental values and manufacturing parameters to carry out advanced management, control and fault detection. The outcomes of FGIR will assist with maintenance scheduling to reduce downtime. The future industrial network will connect a varying range of industrial machinery within one or more locations that could change over time. To facilitate FGIR, the current heterogeneous hierarchical localized network structure should be replaced with IP-based networking to provide flexible real-time communications and simplified data mapping. There is also a requirement to change the configuration of the industrial machines and production systems as the production tasks change. It is in this context that future industrial facility networks should embrace Software-Defined Networks (SDNs) to provide flexible programmatic capabilities. The research gap that this paper addresses is the introduction of SDN and IP-based networking into an industrial automation setting to provide flexibility and programmability while maintaining the features and capabilities expected for a real-time communications environment.

1.1. *Software Defined Network*

SDNs [1–3] separate the networks control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane). With the separation of the control and data planes, network switches become simple forwarding devices, and the control logic is implemented in a logically centralized controller, simplifying policy enforcement, and network (re)configuration and evolution [4]. Therefore, the most promising and possibly profitable benefit of SDNs is their potential in making the network directly programmable. SDNs become a hot topic at within cloud and enterprise networks in about 2010. To our knowledge, SDN solutions are new to the industrial automation domain. SDNs permit reusable configurations and designs that improve system performance. SDNs complement and build on technologies such as industrial Ethernet [5–7], wireless technologies [8,9], and network technologies with guaranteed timing behavior for real-time (e.g., [10]) communication. SDNs can be characterized by: (1) decoupling the control plane from the data plane within network devices; (2) providing programmability for network services; (3) taking forwarding decisions based on flow instead of destination; (4) hosting control logic in an external network component called controller or Network Operating System (NOS); and (5) running software applications on top of the NOS to interact with the underlying data plane devices. With the realization of the aforementioned characteristics of SDNs, the current "touch many, configure many" model is being evolved into "touch one, configure many" [11].

1.2. *Brief History of Industrial Networks*

Dedicated industry networks, e.g., Fieldbus System, dominated the early days of industrial automation. The reduction of the communication gap in the lower level of the automation pyramid was the essence of the dedicated communication infrastructure. However, the complexity of coupling different communication technologies and protocols used across different communication layers was one of the fundamental motives to adopt a solution. Table 1 presents the timeline of the progression of industrial automation networks with unsustainable disruptions that come from the evolution of computer networks. In the 2000s, the Internet technologies evolved and became commercially successful raising the possibility of plausible disruption with the inclusion of Ethernet-based networks and IP. However, due to the lack of guaranteed real-time capabilities, the phenomena of having

Ethernet-based industry networks did not occur and the emergence of dedicated industry networks continued. Later some of the Ethernet-based approaches including Powerlink, PROFINET, EtherCAT, to name a few, emerged to meet the low-latency requirements, in particular, for motion control applications. In the early 2000s, network evolution occurred with the integration of wireless networking. The IEEE 802 protocol family was aggressively adopted to realize the flexibility afforded by connecting machines and devices wirelessly. The typical use of wireless networks in the automation industry was limited due to the need for wired networks to provide reliable real-time communications. We have yet to see the full use of Wireless Sensor Networks (WSN) in industry automation though it is now a mature technology.

Until recently, industrial communication was a mixture of Fieldbus, Ethernet and wireless solutions that has become complex, difficult to upgrade or change and remains a challenge to be overcome before industrial automation can take a significant step forward. New networking approaches that have evolved include the Internet of Things (IoT) and Cyber-Physical Systems (CPS), both of which should find a place within future industrial automation solutions. The idea behind CPS being used in industrial automation is to create an industrial ecosystem allowing more comprehensive and more fine-grained interconnections between machines and systems. Moving business logic into the cloud is a promising trend in the application layer of the information processing pyramid. There are two well-known reference architectures for industrial IoT including the Reference Architecture Model for Industry 4.0 (RAMI4.0) [12] and the Industrial Internet Reference Architecture (IIRA) [13]. RAMI 4.0 uses three dimensions including the lifecycle, physical world and the mapping of IT-based business models in describing the space of the fourth industrial revolution. Some of the leading industry sector companies based in Germany initiated and are driving RAMI 4.0. On the other hand, the Industrial Internet Consortium developed IIRA in the U.S. IIRA focuses on four different viewpoints including functional, usage, business, and implementation.

1.3. SDNs in Industrial Automation

In transitioning to a software-defined network, the key challenges involve changing the traditional practices in industrial automation on the factory floor [14,15]. That means providing relevant employees with the tools and knowledge to support new, more intelligent infrastructure and systems.

Cronberger [16] and Kalman et al. [17] first considered and discussed the use of SDN in industrial automation networks. Cronberger investigated the potential of SDN through a conceptual framework whereas Kalman et al. saw SDN as a possible evolution for future industrial Ethernet planning and extensions towards using Layer 3 networks and wireless solutions. In 2015, we first proposed the integration of SDN in industrial automation by reforming the current industry communication pyramid to become a single Ethernet-based solution in a conceptual paper [14]. In [18], the authors proposed an application-aware industrial Ethernet by exploiting the capabilities of SDN in collecting topology information and application requirements. A newly developed routing and scheduling algorithm uses the received information to generate network configuration autonomously. This configuration is later installed in the network through north and southbound communication, and an enhanced TDMA approach is used to facilitate real-time communication. D. Li et al. [15] proposed a single IP-based solution that can respond to the dynamic change of product orders by adaptively reconfiguring the networks. The architecture promised to guarantee real-time data transmission, enable plug-and-play, and support wireless access with seamless handover capability. In [19] the authors reviewed SDN to draw a correlation between the requirement of the industrial network and existing work. In [20] we continue the evaluation of SDN for future industrial automation networks. This work extended the Ryu controller for direct multicast routing of industrial traffic in a cyclic switched Ethernet network setup. The experiment was conducted in an IEC 61499 compliant development environment. The experiment result shows that there is a promising opportunity to have a flexible and reliable network that is also suitable for real-time traffic. Table 2 summarizes the current state of the art of SDN in industrial automation.

Table 1. Industrial communication protocols timeline.

Industrial Communication Protocols							Computer Networks	
Protocol Name		Published by	Place	Com. Tech	Year		Protocol Name	Year
Modbus	Modular Bus	Modicon (now Schneider Electric)	United States	Master/Slave	1979	1970–1980	ARPANET	1970
							Ethernet	1973
							ISO/OSI	1978
PROWAY	Process Data Highway	Working Group 6			1981	1981–1990	MAP	1980
FIP	French Initiatice	factory instrumentation protocol	France	Producer/Consumer	1982			
Bitbus	BIT Fieldbus	Intel Corporation	USA	Master/Slave	1983			
HART	Highway Addressable Remote Transducer	FieldComm Group	USA	Master/Slave	1985		Internet	1981
CAN	Controller Area Network	Robert Bosch GmbH	Detroit, Michigan	Producer/Consumer, Peer to Peer	1985			
P-NET	Process Network	Process-Data Silkeborg ApS	Denmark	Master/Slave	1987		MMS	1985
INTERBUS	INTERBUS	Phoenix Contact	Germany	Master/Slave	1987			
PROFIBUS	The Federal Ministry of Education and Research (BMBF)	process field bus	Germany	Master/Slave, Peer to Peer	1989		Ubiq. Comp	1988
EIB	European Installation Bus (EIB)	EIB Association	Europe	Master/Slave	1991	1991–2000	WWW	1992
Asi	Actuator Sensor Interface	AS-International	Germany	Master/Slave	1992			
SDS	Smart Distributed System	Honeywell	USA	Master/Slave	1993		2G GSM	1996
DeviceNet	Connecting Devices	Allen-Bradley	USA	Producer/Consumer	1993			
FF							WLAN	1997
ControlNet	Real-Time Control Network	Rockwell Automation	USA	Producer/Consumer	1995			
TTP	Time-Triggered-Protocol	Vienna University of Technology	Vienna, Austria	Master/Slave	1998		IoT	1997
Powerlink	Ethernet Powerlink	B&R Industrial Automation GmbH	Austria	Producer/Consumer	2001	2001–2010	Bluetooth	2003
Modbus/TCP	Modbus RTU protocol with a TCP interface that runs on Ethernet	Modicon (now Schneider Electric)	United States	Master/Slave	2001		SOAP	2003
PROFINET	Process Field Net	Profibus & PROFINET International	Germany	Real-Time Ethernet	2001		3G: UMTS	2001
EtherCAT	Ethernet for control automation technology	Beckhoff Automation	Germany	Master/Slave	2003		ZigBee	2003
ISA 100.11a	Wireless Systems for Industrial Automation	International Society of Automation	Worldwide	NIL	2009		3G: HSPA	2005
							UWB	2008
Wire. HART	Wireless HART	HART Communication Foundation	USA	Master/Slave	2007		6loWPAN	2009
							4G: LTE	2010

Table 2. Software-Defined Network Timeline.

Framework/Concept	Brief Description	Year Published
Software Defined Industrial Network [16]	Reflects the possibility of bringing programming capability in industrial network through the use of SDN. A theoretical framework is provided	2014
Outlook on Future Possibilities [17]	Possible evolution of industrial Ethernet using SDN	2014
SDNPROFINET [14]	Proposed to transform the typical communication architecture of PROFINET integrating SDN	2015
SDN-based TDMA in IE [18]	SDN approach is used to formulate an application-aware Industrial Ethernet Based on TDMA	2016
SDIN [15]	Propose a new Software Defined Industry Network (SDIN) architecture to achieve high reliability, low latency, and low energy consumption in Industrial Networks	2016
Challenge and Opportunities [19]	Prospect of future industrial network by means of SDN	2016
Direct Multicast Routing [20]	Evaluates SDN for deterministic communication in distributed industrial automation systems	2017
SDIAN [21]	Software-defined industry automation networks	2017

1.4. Contributions

The contribution of this paper can be summarized as

1. We investigate the research gap that exists for IP-based networking in industrial automation and introduce a novel industrial network framework based on an SDN communication architecture.
2. We propose two solutions for flow creation in relieving the incurred overhead due to the flow setup cost in SDN.
3. We render an optimal latency model based on a meticulous flow analysis using L_1-Norm Optimization to calculate the shortest path. It verifies the quantified model using a Monte Carlo simulation.
4. We validate the proposed scheme by running an experiment in an emulated environment using Mininet [22].
5. We exploit the merits of the proposed framework by presenting an ongoing test bed implementation. The investigation is conducted on a food processing demonstrator.

1.5. Paper Organization

The remainder of this paper is organized as follows. Section 2 presents the architecture, communication framework and flow creation of SDIAN. In Section 3, we examine the flow analysis and present an optimal latency model of the proposed solution. Section 4 exhibits the stochastic analysis of the model formulated in Section 3. In Section 5, the network performance of the target mesh topology is shown using a modelled emulation scenario and a report on the experimental setup in a food processing plant demonstrator is presented. Finally, Section 6 concludes the paper. This manuscript is the extended version of the paper presented in [21].

2. Architecture and Framework

In this section, we first introduce the architecture and three-layer SDIAN framework. Then we describe our proposed flow installation scheme.

2.1. System Model

In this section, we present the conceptual architecture of the proposed SDIAN and the packet dissemination model with the plant components. Figure 1 shows the remolded version of a standard plant hierarchy that incorporates SDN features and builds an intelligent industrial automation network.

In this transformed architecture, within the three hierarchical levels (Control Plane, Plant Level and Field Level), traditional proprietary Programmable Logic Controllers (PLCs) are replaced with the open Raspberry Pi (RPi) systems running the Rasbian operating system, a Linux flavor, and using open-source language for software-defined automation control. Sensors and actuators are interfaced with field level RPis, except the direct I/Os, which are interfaced directly within the plant level hierarchy. A script running on the RPi-based PLCs can receive and send interrupts from the sensors and to the actuators through I/O pins. The scripts written for the RPis replicate the behavior of traditional PLCs. The data layer communication is illustrated using group-1 messages (1A–1E) shown in Figure 1. In this scenario, when an object is detected on the conveyor belt, RPI-PL-1 receives an interrupt and invokes the robotic arm via a reply interrupt. This interrupt is sent through the output pin. In this case, the response of the arm is to deliver the object to another conveyor belt within a limited time constraint. Likewise, group 2 messages (2A–2C) are used to present the control layer communication. In this scenario, a remote SDIAN administrator updates control applications deployed on the controller. After receiving updates, the controller adaptively pushes the information to the associated RPis. Based on the updated instructions received, RPis update the data plane behavior accordingly.

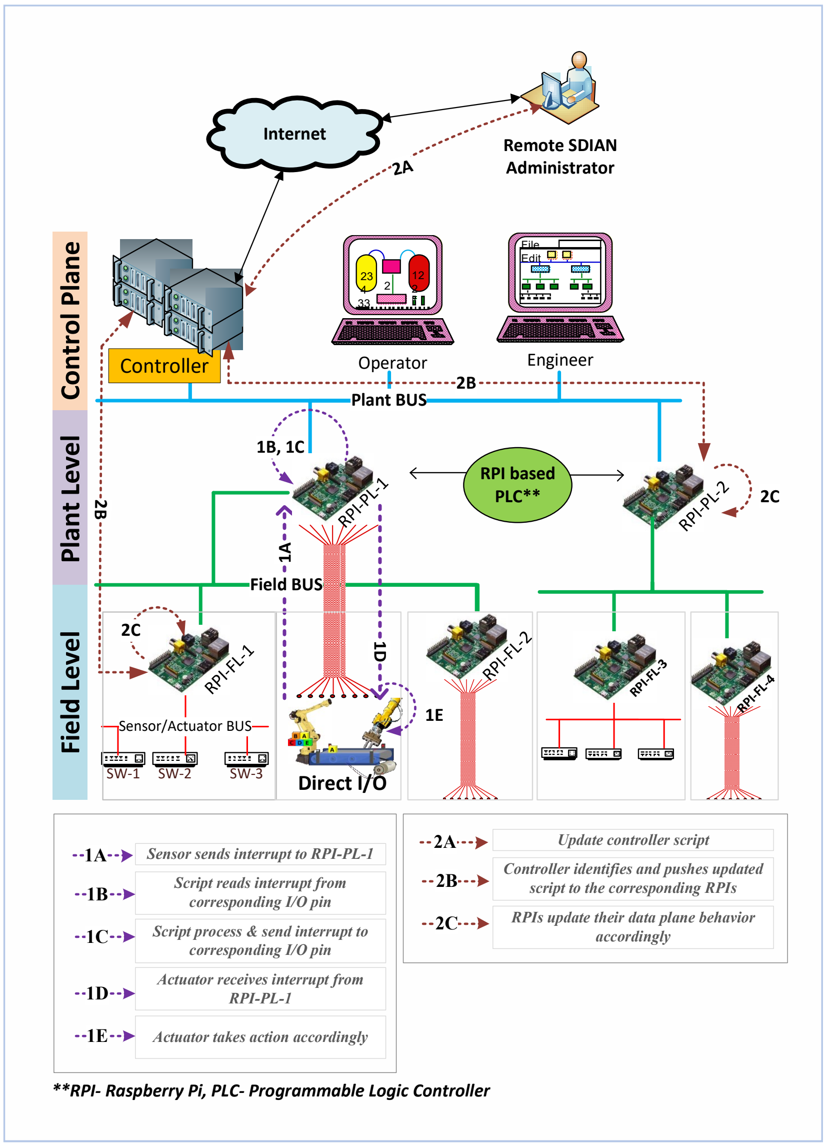

Internet
Remote SDIAN Administrator
2A
Control Plane
Controller
Operator
Engineer
2B
Plant BUS
Plant Level
1B, 1C
RPI-PL-1
RPI based PLC**
RPI-PL-2
2C
2B
1A
Field BUS
1D
Field Level
2C
RPI-FL-1
Sensor/Actuator BUS
SW-1
SW-2
SW-3
Direct I/O
1E
RPI-FL-2
RPI-FL-3
RPI-FL-4
1A Sensor sends interrupt to RPI-PL-1
1B Script reads interrupt from corresponding I/O pin
1C Script process & send interrupt to corresponding I/O pin
1D Actuator receives interrupt from RPI-PL-1
1E Actuator takes action accordingly
2A Update controller script
2B Controller identifies and pushes updated script to the corresponding RPIs
2C RPIs update their data plane behavior accordingly
**RPI- Raspberry Pi, PLC- Programmable Logic Controller

Figure 1. SDIAN (Software Defined Industrial Automation Network) conceptual architecture.

2.2. SDIAN Communication Framework

Figure 2 shows the three-layer SDIAN communication framework. Sensors, actuators, and RPis reside in the data plane, while the logically centralized but physically distributed controllers reside in the control plane. RPis are responsible for receiving packets from sensors and instruct the corresponding actuators to take actions based on the respective flow retrieved from the flow table or corresponding controller. In this framework, RPis are connected through a mesh topology. We deliberately use a mesh topology to map the requirements of the food processing plant, which is presented in Section 5. In the case of a flow table miss [23], an RPi sends a Packet-In message to the controller sitting in the control plane. After getting the Packet-In message, the controller instructs the RPi by sending a Packet-Out/Flow-MOD message. This communication between data plane and control plane happens through the southbound interface (SBI) of the control plane. A task or application is created in the application (also called service management-control) plane that explicitly uses northbound interface (NBI) to translate the business use case, network requirements and, behavior programmatically and logically to the controller. The users are responsible for defining the attributes of a task. Table 3 presents the summary of the different components of the SDIAN architecture.

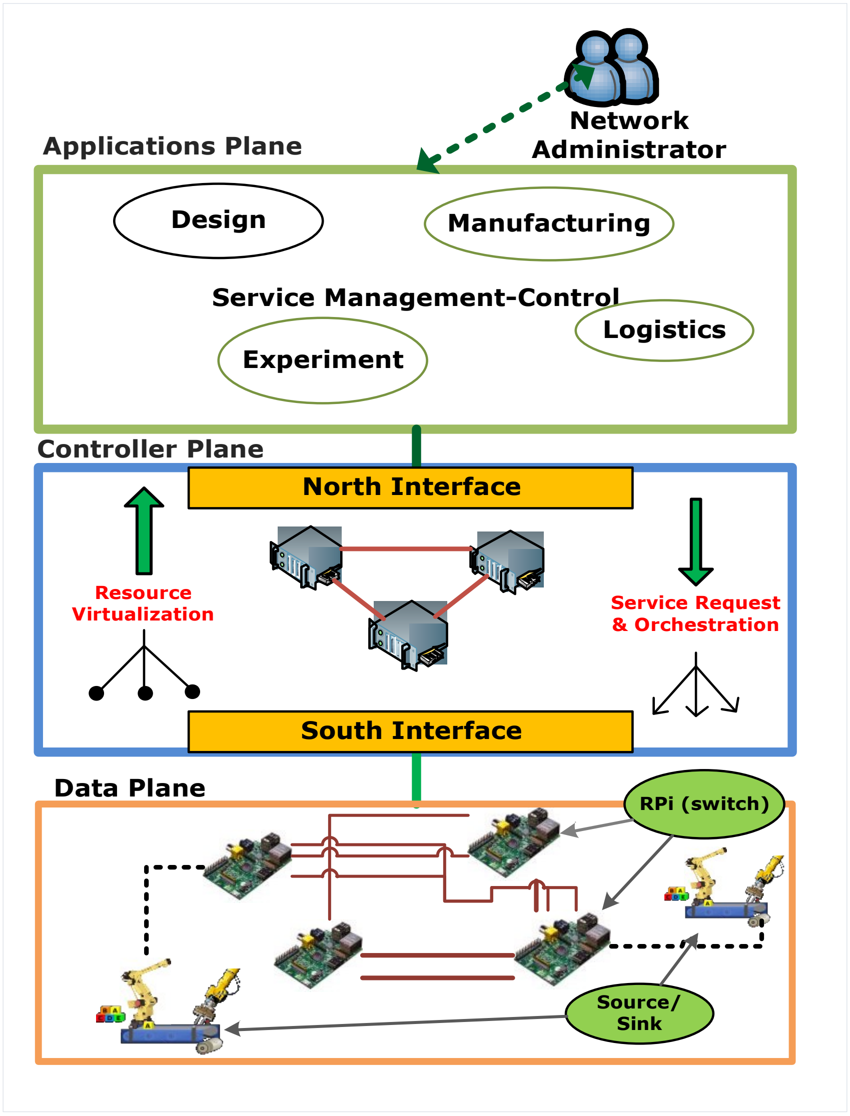


Figure 2. SDIAN communication framework.

Table 3. SDIAN (Software Defined Industrial Automation Network) architectural components.

Component	Task	Layer
RPi	Receive and send interrupt to sensors and actuators	Data Plane
Sensors	Sends an interrupt to an associated RPi immediately after sensing an object	Data Plane
Actuators	Executes the explicitly specified action immediately after receiving an interrupt from RPi	Data Plane
Southbound Interface (SBI)	Interface between data and controller plane. The functions realized through this interface include, but not limited to: (i) programmatic control of all forwarding operations (ii) monitoring (iii) network statistics (iv) advertisement and (v) event notification	Between Control and Data Plane
Controller	*Manage/control* network services. It consists of NBI and SBI agents and control logic. A logically *centralized* but physically *distributed*	Control Pane
Northbound Interface (NBI)	Interface between application and controller plane. It typically provides an abstract view of the network and enables direct expression of network requirements and behavior	Between Application and Control Plane
Applications	Programs in execution that explicitly translate the *business use case*, network *requirements* and, *behavior* programmatically and logically to the controller	Application Plane

2.3. Creating Flows

Unlike other networks, industrial networking environments have specific considerations and requirements to fabricate a deterministic system. These include—real-time network performance, remote access, onsite security, reliability, and ease of use features and manageability. The unique features, when compared to other communication environments, represent significant disparities and pose both challenges and opportunities when implementing SDN-based industrial Ethernet infrastructure. By the inclusion of SDN, there is an inherent opportunity to resolve the reliability, manageability and ease of use issues that are a challenge to achieving real-time performance. Due to the fundamental hardware attributes of switch and software implementation inefficiencies, the latency of flow installations is higher than in traditional network installations. In the case of a flow table miss, there is a higher latency to resolve what should be done with the first packet. From the empirical study provided in [22], it was identified that the root causes of this high latency are as follows: (a) outbound latency, i.e., the latency incurred due to the installation/modification/deletion of forwarding rules, (b) inbound latency, i.e., the latency to send packet events to the controller can be high, in particular, when the switch simultaneously processes forwarding rules received from the controller.

We provide two solutions for flow creation, from which the network administrators can determine the appropriate flow mapping based on their predilection and the application requirements. In the first solution, we use the innovative idea of mixing reactive and pro-active flow installation methods. This is referred to as a Hybrid Flow Installation Scheme (HFIS). With HFIS we cater for non-real-time traffic, in other words, delay tolerant traffic. We use two immediately deployable techniques: Flow Engineering (FE) and Rule Offload (RO). When a switch in the control-level network of a plant receives a packet from control and monitoring devices, it starts by performing a table lookup in the flow table. If a match is found with a flow table entry, it applies the action set associated with the flow as per the Open Flow 1.3 specification [22]. In the case of a table miss, when the controller receives a Packet-In message, it first calculates the shortest route (FE) to reach the destination and then sends the respective Packet-Out/Flow-Mod messages to all switches across this route (RO). Therefore, the packet transmission latency is increased by only one inbound and outbound event irrespective of the number of relay nodes it goes through before it reaches the destination.

The precise synchronization of processes underpins today's manufacturing industry, and therefore, the network must be enhanced to ensure consistent real-time performance in transporting deterministic delay-sensitive traffic. Data must be prioritized based on QoS parameters to ensure that critical information is received first. To tackle this problem, in the second solution, we propose to use a *Pro-active Flow Installation Scheme (PFIS)* catering for delay-sensitive traffic by providing precise synchronization. In this case, we adopted the direct RO method. The controller sends the flow installation packet for all pre-determined critical delay-sensitive traffic to the switches immediately after switch discovery. This pre-installation happens during the convergence of the network. For further clarification, we present the SDIAN packet dissemination model in Figures 3 and 4. In Figure 3, the packet exchange is classified into two categories—Non-Real-Time (NRT) communication and real-time (RT) communication. We apply HFIS for NRT and PFIS for RT. Figure 4 illustrates the working mechanism of PFIS. Please note that in the test bed implementation the data channel and control channel are separate, but for drawing simplification this is not portrayed in Figure 4. As shown in Figure 4a, the switch S1 receives a data packet from a field level device. For this packet, there is a table miss, therefore, the switch sends a control packet (Packet-In) request to the controller. Based on the header information, the controller determines the shortest path for this packet and responds with Packet-Out to all the intermediate switches along this path (Figure 4b). Therefore, as shown in Figure 4c, there is no further table miss as all the intermediate switches along the path pre-install the flow into the flow table before the packet arrives.

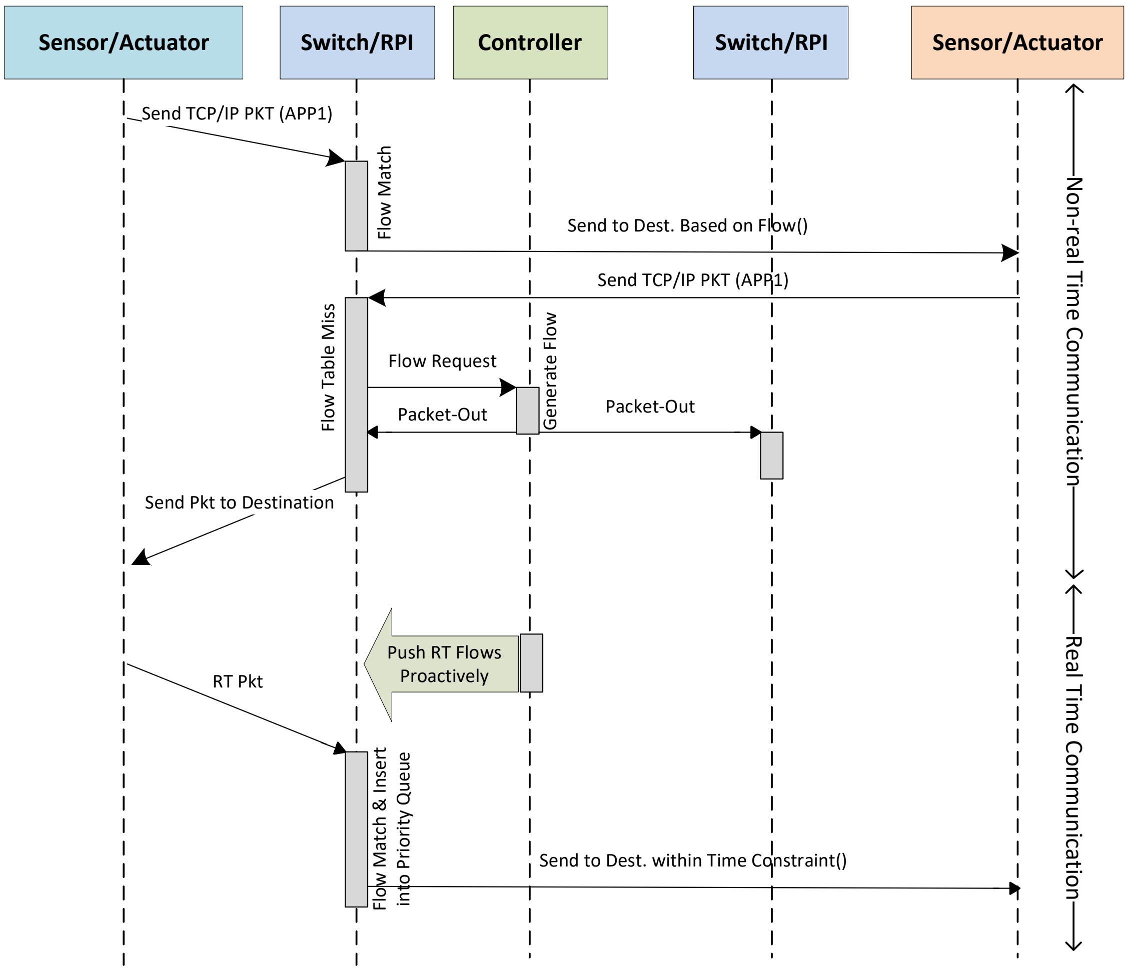
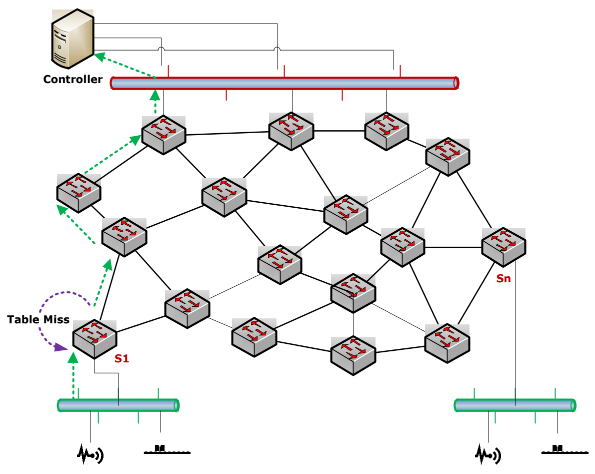
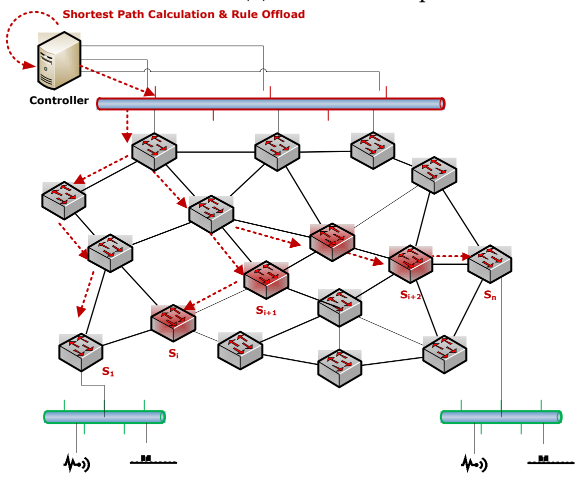


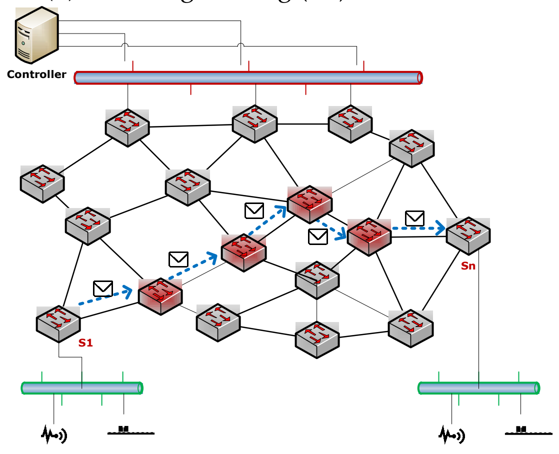
Figure 3. SDIAN packet dissemination model.



(**a**) Table Look Up



(**b**) Flow Engineering (FE) and Rule Offload (RO)



(**c**) Reaches Destination

Figure 4. Working mechanism of HFIS (Hybrid Flow Installation Scheme). (**a**) Table Look Up; (**b**) Flow Engineering (FE) and Rule Offload (RO); (**c**) Reaches Destination.

3. Flow Analysis

In this section, we first illustrate the basic notation used to represent the data layer of the control network of a plant. Since the control channel is separated from the data channel, we kept the graph representation of the control channel out of the scope of this paper and assumed that each switch could reach the controller in single hop fashion using a secured and fast directly connected control channel. Now, we formulate the shortest path routing as the flow optimization problems in a network that is realized by the controller based on the discovered topology. Finally, we compute the model for determining optimal latency to reach the destination.

3.1. Data Layer: Basic Notations

We represent our n-node data plane of the control network of a plant by an undirected graph, $G = (S, L, X)$, where $S = \{s_1, s_{2,\ldots,} s_n\}$ is the set of switches, L is the set of links, and X is an $n \times n$ matrix defined by $\{x_{ij} | (i, j) \in L\}$, where each (i, j)-th entry, denoted by x_{ij}, represents the positive weight of a link $(i, j) \in L$. Due to the undirected nature of the graph, (i, j) and (j, i) designate the same link, i.e., $x_{ij} = x_{ji}$. When $(i, j) \notin L$, delineate $x_{ij} = 0$ fabricating the weight matrix $X == [x_{ij}]$ into symmetric. We also define that X is a 0-1 matrix, i.e., all links have a unit weight, therefore, G refers to a simple graph and, X is the respective adjacency matrix.

Consider $d = [s_1, s_n]$, $s_1, s_n \in S$ denotes the source-destination switch pair in the network G and $F^d : S \times S \to \boldsymbol{R}^+$ function defines the amount of traffic ($f^{(d)}$- unit) that traverse from s_1 (source) to s_n (destination) subject to the following constraints:

(1) along network links:

$$\text{if } (i,\ j) \notin L \text{ then } F^d_{ij} = 0 \tag{1}$$

(2) along one direction:

$$\text{if } F^d_{ij} > 0 \text{ then } F^d_{ji} = 0 \tag{2}$$

(3) at source s_1:

$$f^{(d)} + \sum_{k=1}^{n} F^d_{ks_1} = \sum_{j=1}^{n} F^d_{s_1 j} \tag{3}$$

relay node $i \neq s_1,\ s_n$:

$$\sum_{j=1}^{n} F^d_{ij} = \sum_{k=1}^{n} F^d_{ki} \tag{4}$$

(4) at destination s_n:

$$\sum_{k=1}^{n} F^d_{ks_n} = \sum_{j=1}^{n} F^d_{s_n j} + f^{(d)} \tag{5}$$

The constraint in Equation (1) ensures that for each link $(i,\ j) \notin L$, $F^d_{ij} = 0$ and in particular, for each undirected link $(i,\ j) \in L$, the constraint in Equation (2) says if $F^d_{ij} > 0$ then $F^d_{ji} = 0$ or if $F^d_{ji} > 0$ then $F^d_{ij} = 0$. The traffic constraints defined in Equations (3)–(5) state that the amount of $f^{(d)}$ unit traffic sent by source s_1 is received by destination s_n at the exact number. The amount of traffic entering and leaving a relay switch is same.

Considering a set of intermediate or relay switches $S_{F(d)} \subset S$ and a corresponding subset of links $L_{F(d)} \subset L$ to carry the given $f^{(d)}$ unit traffic from source s_1 to destination s_n, we induce a directed (or oriented) sub-graph of G, $G_{F(d)} = \left(S_{F(d)}, L_{F(d)}\right)$. $G_{F(d)}$ is a directed acyclic graph (DAG, we refer to it as a routing graph) that routes the traffic from source s_1 to destination s_n. The traffic could split or merge across the nodes of $G_{F(d)}$ to travel across multiple paths. We define $F^{d'}$ to refer the collection of flows, in other words, all functions that satisfy the constraints in Equations (1)–(5).

In the following subsection, we derive the shortest path routing strategy by minimizing L_1-norm of traffic between a given source-destination pair. We build this model based on the fabrication of two well-known results [24,25] presented in [26].

Shortest Path Routing (L_1-Norm Optimization)

For simplicity and clarity of notation, we assume that $f^{(d)} = 1$, F_{ij} equivalently specifies the traffic function $F^{(d)}$, $s_1 = 1$, and $s_n = n$. Therefore, we define the following L_1-norm (L_1 Primal) flow optimization problem that can be solved using linear programming (LP).

$$\begin{gathered}\min_{F^d} \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}F_{ij} \\ \text{s.t. (1)–(5)}\end{gathered} \tag{6}$$

To comply with the constraints specified in Equations (1)–(5), (6) can more specifically be stated as

$$\sum_{j:(i,j)\in L} F_{ij} - \sum_{k:(k,i)\in L} F_{ki} = \begin{cases} 1 & \text{if } i = s_1 \\ 0 & \text{if } i,j \neq s_1, s_n \\ -1 & \text{if } i = s_n \end{cases} \tag{7}$$

where, $F_{ij} \geq 0$ and $1 \leq i,j \leq n$.

Hence, the optimization problem presented in (6) minimized the weighted L_1-norm. Based on the flow conservation constraints presented in Equation (7), we consider the dual (L_1 Dual) of (6) in terms of Lagrange multipliers (U_i's) to find the shortest path routing

$$\max_{U} U_1 \tag{8}$$

Subject to,

$$U_n = 0 \text{ and } U_i - U_j \leq x_{ij}, \forall_{ij} \in L \tag{9}$$

Assuming F^* and U^* refer to the optimal traffic solution for the primal and dual problem respectively, we derive the following relations between $F_{ij}^{*\prime}s$ and $U_i^{*\prime}s$

$$\text{if } F_{ij}^* > 0, \text{ then } U_i^* - U_j^* = x_{ij} \tag{10}$$

and

$$\text{if } F_{ij}^* = 0, \text{ then } U_i^* - U_j^* < x_{ij} \tag{11}$$

Based on these relations, we can define the following properties of the optimal solution ($U_i^{*\prime}s$) of the dual problem.

Lemma 1. *Let P_1 and P_2 (alternative to P_1) are two different paths from source (s_1) to destination (s_n) to carry the traffic. If for each link $(i,\ j) \in P_1$, $U_i^* - U_j^* < x_{ij}$ then P_1 is not the shortest path and $U_{s_1}^* < \sum_{(i,j)\in P_1} x_{ij}$. On the other hand, the alternative path P_2 is a shortest path if for each link $(i,\ j) \in P_2$, $U_i^* - U_j^* = x_{ij}$ and $U_{s_1}^* = \sum_{(i,j)\in P_2} x_{ij}$.*

It is evident from the above Lemma that for any switch S_i on a shortest path, $U_{s_i}^*$ is the shortest path distance from the switch S_i to the destination S_n. All intermediate switches including S_i and S_n are the elements of $S_{F^{(d)}}^*$ that form the shortest routing graph $G_{F^{(d)}}^*$.

3.2. Optimal Latency Model: Hybrid

In this subsection, we derive the optimal latency model for HFIS. In HFIS, a packet traverses across the shortest path to reach the destination switch.

Let α denotes the total latency for a packet to reach from source S_1 to destination S_n, α_{in} refers to the inbound latency, α_{ou} is the outbound latency, $\alpha_p^{S_k}$ is the single hop propagation delay of a packet travelling from S_k to S_{k+1}. We consider γ is the average time taken by a controller to process a Packet-In message and β is the control channel latency i.e., time taken by a Packet-In/Packet-Out message to travel between a switch and a controller. To this end, our target is to minimize the value of α, and therefore, the optimization model of latency can be stated as:

$$\min_{\alpha}\left\{\alpha_{in} + \alpha_{ou} + \sum_{k=1}^{m}\left\{\alpha_P^{S_k}\right\} + 2 \times \beta + \gamma\right\} \tag{12}$$

where, m is the number of hops i.e., the total number of switches in the shortest routing graph $G^*_{F(d)}$ and $S_k \in S^*_{F(d)}$, where $S^*_{F(d)} = \{S_i, S_{i+1}, \ldots\ldots, S_{i+m}\}$.

According to HFIS, only the first switch generates the packet event to the controller; then all switches, including the first switch, along the path receive and install the flow instruction. Therefore, there is only one inbound, one outbound, two control channels and one Packet-In resolution latency. Considering a consistent and deterministic link state and performance among all switches, we assume $\alpha_P^{S_k} \cong \alpha_P^{S_{k+1}} \cong \alpha_P^{S_{k+2}} \cdots \cong \alpha_P^{S_{k+m}} \cong \alpha_p$, where α_p is the average propagation delay, therefore, we can rewrite Equation (12) as follows

$$\min_{\alpha}\left\{\alpha_{in} + \alpha_{ou} + m \times \alpha_p + 2 \times \beta + \gamma\right\} \tag{13}$$

Lemma 2. *During the lifetime of a packet, if it traverses across the shortest path, then latency $\alpha \propto \alpha_p$, i.e., $\alpha = m \times \alpha_p + K$, where $K = \alpha_{in} + \alpha_{ou} + 2 \times \beta + \gamma$.*

Lemma 2 asserts that in the entire journey of a packet, there is no more than one table miss regardless of the number of hops across the path. Therefore, one table miss generates only one Packet-In event incurring single inbound (α_{in}) and outbound (α_{ou}) latency with the associated control channel (β) and Packet-In processing time by controller (γ).

3.3. Optimal Latency Model: Pro-Active

According to the second solution, the controller will pro-actively offload the rule to all switches immediately after the deployment of an application. A network administrator deploys an application through the application plane. The application plane creates a particular flow and sends it to the controller in the control plane through the northbound interface. The controller then floods the flow across all the switches within the respective domain. The value of the associated *Idle timeout* and *Hard timeout* [23], in this case, are set to zero i.e., Flow entry is considered permanent, and it does not timeout unless it is removed with a flow table modification message of type OFPFC_DELETE [23]. When a switch receives a packet of this kind, the switch gets an obvious *table match* and therefore, apply the action accordingly. This pre-offloading of flows eventually eradicates the control channel communication entirely during the lifetime of a packet in the data plane.

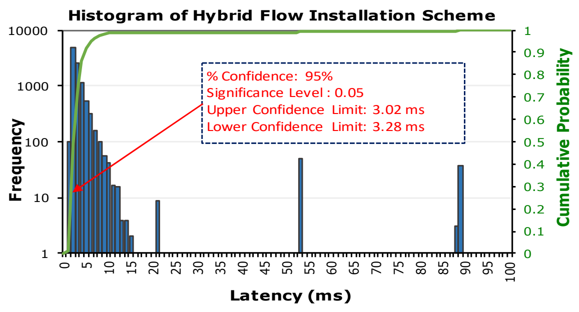
Lemma 3. *With the PFIS, if a packet travels across the shortest path, then the latency is calculated as $\alpha \propto \alpha_p$ i.e., $\alpha = m \times \alpha_p + K$, where $K \cong \alpha_{in} + \alpha_{ou} + 2 \times \beta + \gamma \cong 0$.*

Lemma 3 asserts that in the entire journey of a packet, there is no *table miss* regardless of the number of hops across the path. Therefore, there is no Packet-In event, i.e., inbound (α_{in}) and outbound (α_{ou}) latency with the associated control channel (β) and Packet-In processing time by the controller (γ) are equivalent to zero.

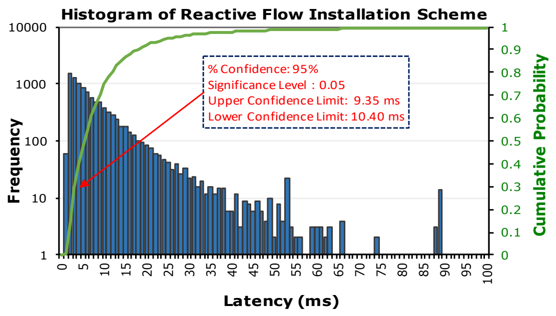
4. Stochastic Analysis of SDIAN

To validate the analytical approach presented in Section 3, we perform an extensive Monte Carlo simulation with 10,000 runs. In each run, we use a randomized distribution of inbound (α_{in}) and outbound (α_{ou}) flow latency, control channel latency (β), data channel latency (α_p) and packet processing time (γ) by a controller. The distribution of α_{in} and α_{ou} is fabricated from the outcome of a comprehensive measurement study [27,28] conducted using four types of production SDN switches. The distribution of inbound latency is a Chi-squared distribution attributed by a mean of 1.853 ms, a median of 0.71 ms and a standard deviation of 6.71 ms. The outbound delay is less variable and skewed with the same mean and median of 1.09 ms and a standard deviation of 0.18 ms. Assuming the simulation is running with a ten (10) switch control network for a single small-scale plant, the number of hops (m) in the shortest path calculation is varied between 1 and 10 and the distribution is a normal distribution with a mean of six (6) and standard deviation of three (3). The Round Trip Time (RTT) between two switches (α_p) and between controller and switch (β) is negligible $(\approx 0.1 \text{ ms})$. The influx rate of *Packet-In* messages from switches to the controller determines the time (β) taken by a controller to process a packet and therefore the distribution of (β) is a normal distribution with a mean of 5.49 µs and standard deviation of 2.86 µs.

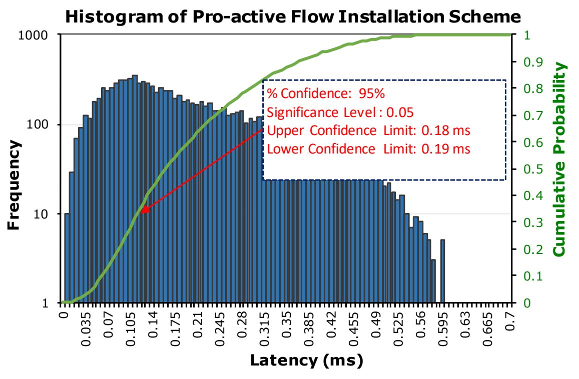
Figure 5a–c respectively shows the histogram of the Monte Carlo simulation results of three flow installation schemes: hybrid, reactive and pro-active. The bin size in Figure 5a,b is 5 ms whereas in it is 0.035 ms. The ascendancy acquired by using HFIS and PFIS over the Reactive Flow Installation Scheme (RFIS) is discernible. 95% of packets are resolved within 3.28 ms using the HFIS and within 0.19 ms using the PFIS. Table 4 presents the summary simulation result statistics as shown in Figure 5.



(**a**)



(**b**)



(**c**)

Figure 5. Histogram of monte carlo simulation results of (**a**) Hybrid Flow Installation Scheme (HFIS) (**b**) Reactive Flow Installation Scheme (RFIS) and (**c**) Pro-active Flow Installation Scheme (PFIS).

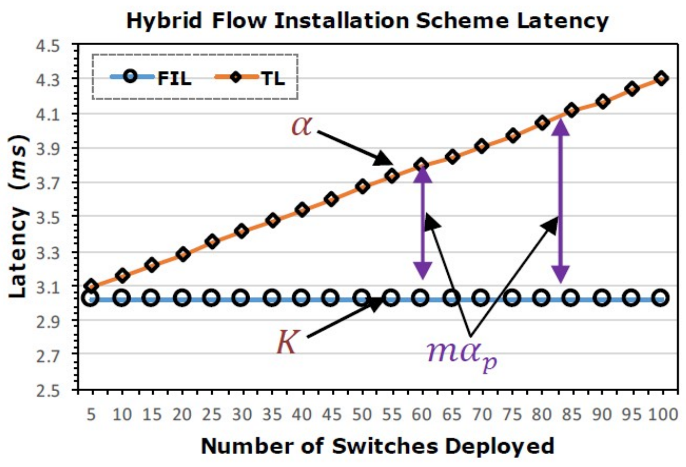
Table 4. Summary statistics of stochastic analysis.

Sample Size		**HFIS**	**RFIS**	**PFIS**
		10,000	**10,000**	**10,000**
Central Tendency	**Mean**	3.15533	9.84786	0.19091
	Median	2.03766	5.40382	0.163
	StErr	0.06706	0.27245	0.00119
Spread	**StDev**	6.7095	26.7239	0.1189
	Max	88.7062	883.67201	0.598
	Min	0.6773	0.6773	0.003
	Range	88.0288	882.9946	0.595
	Q(0.75)	3.0157	10.2242	0.264
	Q(0.25)	1.5196	2.7774	0.098
	Q Range	1.4960	7.4467	0.166
Shape	Skewness	10.3932	15.9369	0.8368
	Kurtosis	118.0304	330.8579	0.0093
Quantiles, Percentiles, Intervals	90% Interval	Q(0.05) = 1.17 Q(0.95) = 6.01	Q(0.05) = 1.34 Q(0.95) = 24.9	Q(0.05) = 0.04 Q(0.95) = 0.42
	95% Interval	Q(0.025) = 1.08 Q(0.975) = 7.84	Q(0.025) = 1.22 Q(0.975) = 35.61	Q(0.025) = 0.03 Q(0.975) = 0.47
95% CI for the Mean	Upper Limit	3.0210	9.5795	0.1883
	Lower Limit	3.2839	10.7175	0.1930

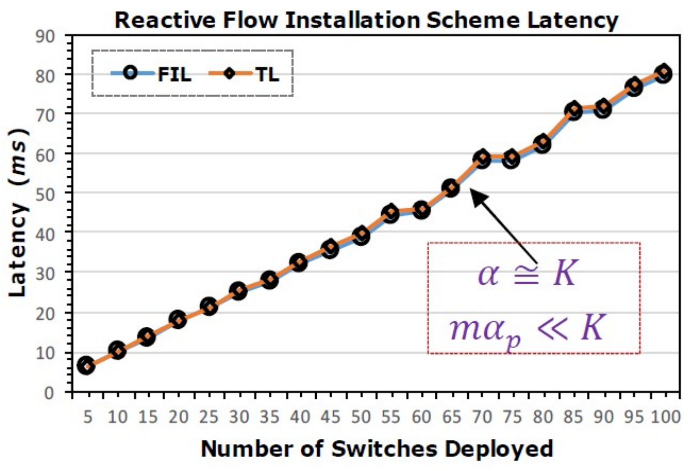
To see the implication of Lemma 2, we repeat the simulation with the number of switches varying between ($5 \leq S \leq 100$). After performing all the Monte Carlo simulation runs, we average the results obtained for each value of S as shown in Figure 6. Figure 6a shows that with HFIS, the total flow installation latency ($K = \alpha_{in} + \alpha_{ou} + 2 \times \beta + \gamma$) is constant irrespective of the network size; therefore, the total latency (α) is directly proportional to $m \times \alpha_p$. Figure 6b presents the latency for RFIS. In the worst-case scenario for RFIS, each switch in a route could have packet flow table miss with the associated flow setup cost. Therefore, the total latency is dominated by the flow installation overhead ($\alpha \cong K$, $m\alpha_p \ll K$). The PFIS latency results are shown in Figure 6c. Since the respective flow is installed across all switches before the arrival of any data packet, there is no table miss. As in the HFIS case α is directly proportional to $m\alpha_p$ with $\alpha \cong m\alpha_p$ and $K \cong 0$.

Discussion

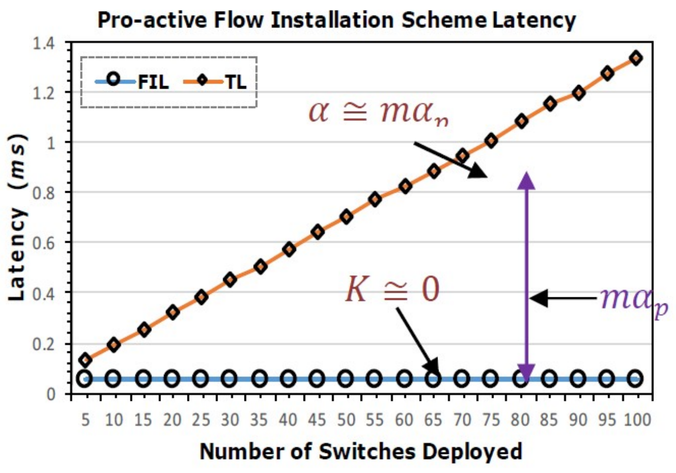
The results presented in Figures 5 and 6 and Table 4, highlight that the PFIS confers the lowest latency as the overhead from flow establishment is, in fact, close to zero. Regarding HFIS, the cost for flow setup is constant regardless of the network size. The upper and lower limits of the 95% Confidence Interval (CI) for HFIS in a network of ten (10) switches are 3.02 ms and 3.28 ms respectively, indicating a stable deterministic condition. For consistent RT performance in transporting deterministic delay-sensitive traffic, we can apply PFIS, while we can use HFIS to provision the rest of the traffic sustaining the dynamic behavior of the SDN network. In a nutshell, the latency bound for RFIS, HFIS and PFIS are 0.025–0.975, 1.08–7.84 and 1.22–35.61 ms respectively with 95% confidence.



(a)



(b)



(c)

Figure 6. Mean Latency for varied number of switches (**a**) Hybrid (**b**) Reactive (**c**) Pro-active. FIL—Flow Installation Latency, TL—Total Latency.

5. Experiments

In this section, we first present the network performance of the target mesh topology using a modelled emulation scenario and then report on an experimental setup with the adaptive configuration in a food processing plant demonstrator.

5.1. Emulation Environment

For further validation of our proposed scheme, we run another experiment in an emulated environment using Mininet. Although the accuracy of Mininet cannot be taken for granted particularly for large scale topologies, the SDN community adopts it widely. In our case, we are essentially interested in looking at the expediency of our proposed solution before we investigate it with limited functionality in a real testbed. Therefore, we deploy a small mesh network of five (5) switches and a Ryu controller [29] as shown in Figure 7. The Ryu controller is tailored to incorporate the three flow installation schemes, and Spanning Tree Protocol (STP) is implemented to discard any possibility of creating a loop. We generate the plant level network packets from openflow switch#2 (source) to openflow switch#5 (sink) and vice versa. We varied the rate of packets generated from source to sink and measure the latency and success rate for the three flow installation schemes. We present the results in Figures 8 and 9. In Figure 8, we present the latency for each flow installation scheme against a varying number of packets generated per second. The latency of RFIS increases linearly with the increase in the number of packets while PFIS and HFIS show a similar pattern. The latency bound of PFIS and HFIS are 1–3 ms and 3–7 ms respectively, therefore for this setup, the guaranteed delay is <7 ms. Figure 9 shows the success rate of the three flow installation schemes against a varying packet rate. The success rate for PFIS and HFIS varies from 98–99% and 97–99%.

From the results it was found that the HFIS retains a consistent low latency and high success rate as well as maintaining the flexibility and dynamic behavior of SDN.

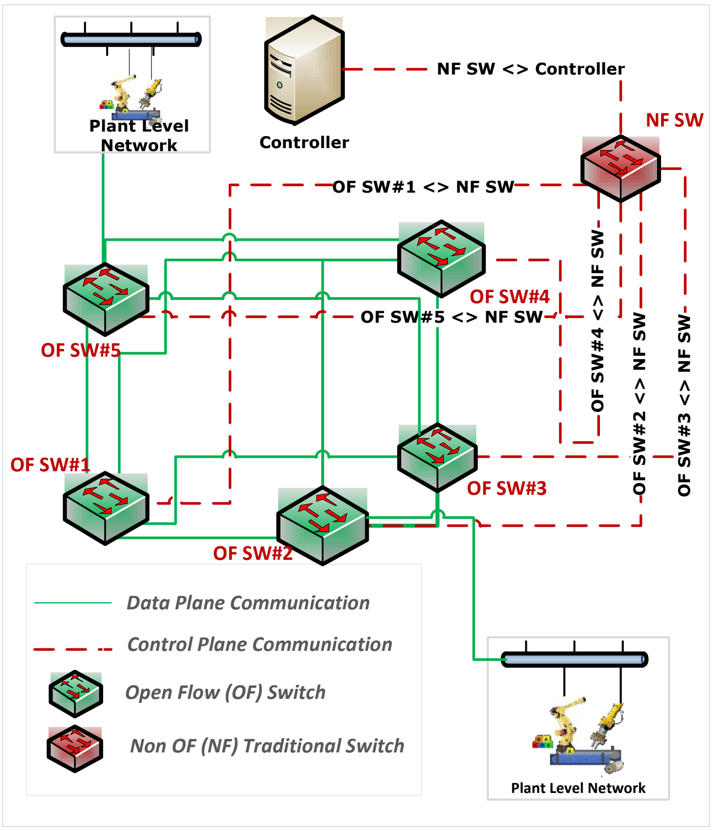


Figure 7. Network setup in Mininet.

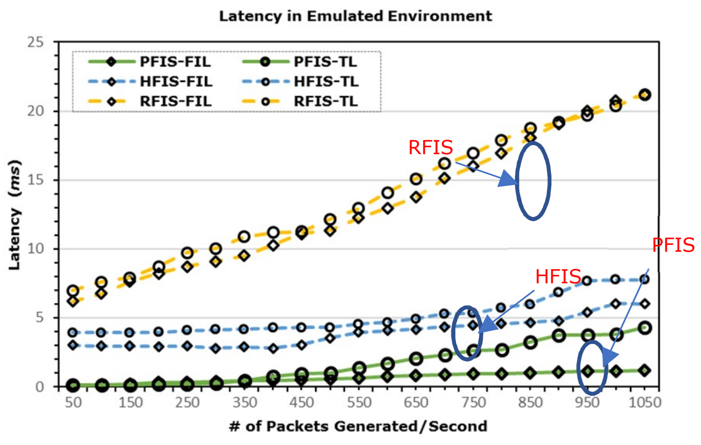


Figure 8. Latency in emulated environment.

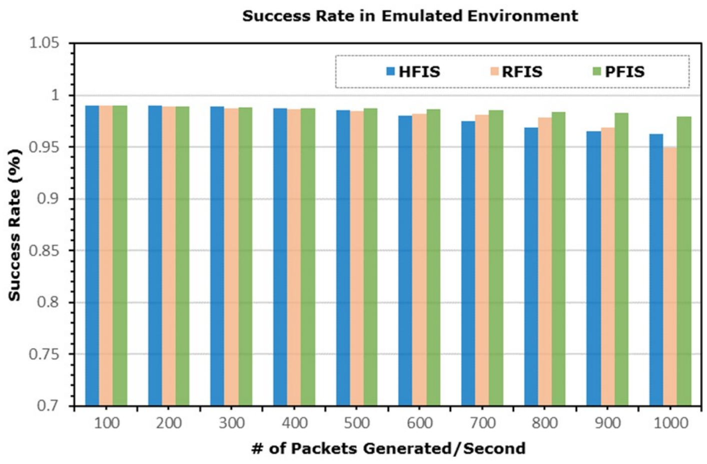


Figure 9. Success rate in emulated environment.

5.2. Test Bed Implementation

The demonstrator bottling plant comprises sensors and actuators such as conveyor belts, physical and vacuum grippers, robots and a turning table. We designed and implemented the test bed experiment to study the performance of the proposed SDIAN model. To do so, we transformed some parts of the demonstrator plant to be controlled by the RPis, while other parts rely on classical PLC solutions and vendor specific robot controllers. The portion that was controlled by the RPis includes a conveyor belt carrying bottle caps, sensors to detect when a cap arrives and a robotic arm as an actuator that will pick the cap and restore it into the designated location. The behavior of the sensors and actuators are determined by the controller and accordingly the script is pushed into the RPi. In this experiment, we have replaced two of the traditional PLCs with RPI-based PLCs to control a small set of sensors/actuators mounted on the Festo plant demonstrator. As shown in Figure 10, we interface two RPi-based PLCs (RPI-1 and RPI-2) with one of the gear boxes from the food demonstrator plant to get connectivity with a set of sensors and actuators. The two RPi-based PLCs are connected to a controller through a control channel. We use a python script to read, write and process signals from/to the I/O pin of the RPi. The python script replicates the standard behavior of traditional PLCs. We deploy a controller application in the controller to facilitate flow control communication between controller and RPi-based PLCs.

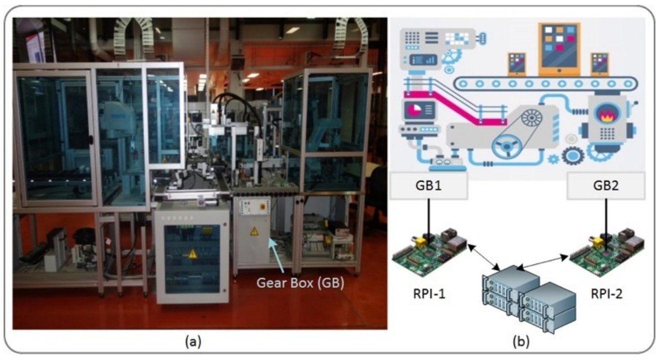
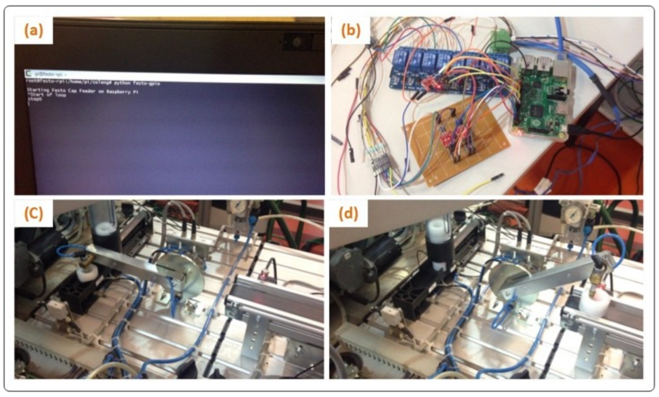


Figure 10. (**a**) Festo-based food processing plant demonstrator; (**b**) deployment diagram of the test bed.

Figure 11 presents the collage of a few snapshots of our test bed setup. It briefly demonstrates the different stages of the experiment. Clockwise from top left: a python script running on an RPi-based PLC replicates a traditional PLC, interfacing of RPis with sensors/actuators through the gear box, a robotic arm picking the desired object based on the instruction received from the corresponding RPi, and placing the object into a designated conveyor belt.



```
root@festo-rpi:/home/pi/coleng# python festo-gpio

Starting Festo Cap Feeder on Raspberry Pi
*Start of loop
step0
```

Figure 11. Clockwise from top left: (**a**) python script (**b**) interfacing with gear box (**c**,**d**) task execution by robotic arm.

The supplementary video clip demonstrates that the transformed architecture is working in a small-scale testbed experiment.

6. Conclusions

In this paper, we have explained the characteristics of SDN in the context of industrial automation. We highlighted the design of two flow installation schemes to precisely synchronize the industrial automation processes as well as presenting the potential benefits and opportunities of SDN. Furthermore, we have presented our architectural model that utilizes SDN and brought this into the context of an ongoing demonstrator project. Future work comprises the use of our demonstrator in current industry and academic projects. We are addressing both the challenges of the industrial automation hardware as well as integrating SDN into the communications utilizing software configurable devices.

Limitations of the demonstrator constrain evaluation of the proposed framework at this stage; however, the results obtained provide support for the approach and future work. For simplification, we limit our work to wired network technologies although it is evident that the approach could be extended to the integration of wireless (e.g., sensor network) with the wired network to achieve a unified architecture. The inclusion of wireless networks will introduce challenges including the seamless integration between the controllers across the wired and wireless domains. We also have limited our scope to one plant; therefore, the validation of using multiple controllers across multiple plants and the east-west communication are left unexplored and identified as future work. In the proposed framework, each RPi-based PLC is also used as an SDN switch, in future we may consider the use of lightweight SDN switches such as Zodiac FX, which could reduce the chance of bottlenecks across the RPis and clearly separate the forwarding devices from underlying field level sensors and actuators.

Supplementary Materials: The following are available online at http://www.mdpi.com/2224-2708/7/3/33/s1. Video S1: The supplementary video clip of the demonstrator.

Author Contributions: K.A. was responsible for conceptualization, data curation, investigation, methodology, validation, visualization, and writing the original draft. M.G. reviewed and edited the manuscript. J.O.B. and H.S. participated in conceptualization and reviewed the final manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors wish to thank the Virtual Experiences Laboratory (VXLab), RMIT University. VxLab ensures smooth access to the Festo-based food processing plant demonstrator in conducting test bed implementation. This work is also supported in part by the Australia-India Research Centre for Automation Software Engineering (AICAUSE), RMIT University.

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

References

1. Xia, W.; Wen, Y.; Foh, C.H.; Niyato, D.; Xie, H. A Survey on Software-Defined Networking. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 27–51. [CrossRef]
2. Huang, T.; Yu, F.R.; Zhang, C.; Liu, J.; Zhang, J.; Liu, J. A Survey on Large-scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges. *IEEE Commun. Surv. Tutor.* **2016**, *19*. [CrossRef]
3. Hu, F.; Hao, Q.; Bao, K. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 2181–2206. [CrossRef]
4. Raghavan, B.; Casado, M.; Koponen, T.; Ratnasamy, S.; Ghodsi, A.; Shenker, S. Software-defined internet architecture: Decoupling architecture from infrastructure. In Proceedings of the 11th ACM Workshop on Hot Topics in Networks, Redmond, WA, USA, 29–30 October 2012.
5. Skeie, T.; Johannessen, S.; Holmeide, O. Timeliness of real-time IP communication in switched industrial Ethernet networks. *IEEE Trans. Ind. Inform.* **2006**, *2*, 25–39. [CrossRef]
6. Decotignie, J.D. Ethernet-Based Real-Time and Industrial Communications. *Proc. IEEE* **2005**, *93*, 1102–1117. [CrossRef]
7. Rojas, C.; Morell, P.; Sales, D.E. Guidelines for Industrial Ethernet infrastructure implementation: A control engineer's guide. In Proceedings of the 2010 IEEE-IAS/PCA 52nd Cement Industry Technical Conference, Colorado Springs, CO, USA, 28 March–1 April 2010; pp. 1–18.
8. Gungor, V.C.; Hancke, G.P. Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches. *IEEE Trans. Ind. Electron.* **2009**, *56*, 4258–4265. [CrossRef]
9. Hou, L.; Bergmann, N.W. Novel Industrial Wireless Sensor Networks for Machine Condition Monitoring and Fault Diagnosis. *IEEE Trans. Instrum. Meas.* **2012**, *61*, 2787–2798. [CrossRef]
10. Kopetz, H.; Ademaj, A.; Grillinger, P.; Steinhammer, K. The time-triggered Ethernet (TTE) design. In Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05), Seattle, WA, USA, 18–20 May 2005; pp. 22–33.
11. Cronberger, D. Software Defined Networks. 2015. Available online: http://www.industrial-ip.org/en/industrial-ip/convergence/software-defined-networks (accessed on 20 July 2018).
12. RAMI 4.0. Retrieved in August 2017. Available online: https://www.zvei.org/en/subjects/industry-4-0/thereference-architectural-model-rami-40-and-the-industrie-40-component/ (accessed on 20 July 2018).
13. IIRA. Retrieved in August 2017. Available online: http://www.iiconsortium.org/ (accessed on 20 July 2018).

14. Ahmed, K.; Blech, J.O.; Gregory, M.A.; Schmidt, H. Software Defined Networking for Communication and Control of Cyber-Physical Systems. In Proceedings of the 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), Melbourne, VIC, Australia, 14–17 December 2015; pp. 803–808.
15. Li, D.; Zhou, M.T.; Zeng, P.; Yang, M.; Zhang, Y.; Yu, H. Green and reliable software-defined industrial networks. *IEEE Commun. Mag.* **2016**, *54*, 30–37. [CrossRef]
16. Cronberger, D. The software defined industrial network. *Ind. Ethernet Book* **2014**, *84*, 8–13.
17. Kalman, G.; Orfanus, D.; Hussain, R. Overview and future of switching solutions for industrial Ethernet. *Int. J. Adv. Netw. Serv.* **2014**, *7*, 206–215.
18. Schweissguth, E.; Danielis, P.; Niemann, C.; Timmermann, D. Application-aware Industrial Ethernet Based on an SDN-supported TDMA Approach. In Proceedings of the 2016 IEEE World Conference on Factory Communication Systems (WFCS), Aveiro, Portugal, 3–6 May 2016.
19. Henneke, D.; Wisniewski, L.; Jasperneite, J. Analysis of realizing a future industrial network by means of Software-Defined Networking (SDN). In Proceedings of the 2016 IEEE World Conference on Factory Communication Systems (WFCS), Aveiro, Portugal, 3–6 May 2016.
20. Schneider, B.; Zoitl, A.; Wenger, M.; Blech, J.O. Evaluating Software-Defined Networking for Deterministic Communication in Distributed Industrial Automation Systems. In Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Limassol, Cyprus, 12–15 September 2017.
21. Ahmed, K.; Nafi, N.S.; Blech, J.O.; Gregory, M.A.; Schmidt, H. Software defined industry automation networks. In Proceedings of the 2017 27th International Telecommunication Networks and Applications Conference (ITNAC), Melbourne, VIC, Australia, 22–24 November 2017; pp. 1–3.
22. Open Networking Fundation. Software-Defined Networking: The New Norm for Networks. Available online: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf (accessed on 5 March 2018).
23. Opennetworking.org. OpenFlow—Open Networking Foundation. 2016. Available online: https://www.opennetworking.org/sdn-resources/openflow (accessed on 10 April 2018).
24. Kelly, F.P. Network routing. *Philos. Trans. R. Soc. Lond. A Math. Phys. Eng. Sci.* **1991**, *337*, 343–367. [CrossRef]
25. Yufei, W.; Zheng, W.; Leah, Z. Internet traffic engineering without full mesh overlaying. In Proceedings of the INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, Anchorage, AK, USA, 22–26 April 2001; Volume 1, pp. 565–571.
26. Li, Y.; Zhang, Z.L.; Boley, D. From Shortest-Path to All-Path: The Routing Continuum Theory and Its Applications. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 1745–1755. [CrossRef]
27. He, K.; Khalid, J.; Gember-Jacobson, A.; Das, S.; Prakash, C.; Akella, A.; Li, L.E.; Thottan, M. Measuring control plane latency in SDN-enabled switches. In Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, Santa Clara, CA, USA, 17–18 June 2015.
28. Blenk, A.; Basta, A.; Zerwas, J.; Reisslein, M.; Kellerer, W. Control Plane Latency With SDN Network Hypervisors: The Cost of Virtualization. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 366–380. [CrossRef]
29. Community, R.S.F. Ryu SDN Framework. 2014. Available online: https://osrg.github.io/ryu/ (accessed on 10 April 2018).