Software Defined Networks in Industrial Automation

Khandakar Ahmed[1], Jan O. Blech[2], Mark A. Gregory[2], Heinrich Schmidt[2]
[1]Victoria University, [2]RMIT University
khandakar.e.ahmed@ieee.org, {janolaf.blech, mark.gregory, heinrich.schmidt}@rmit.edu.au

Abstract—Trends such as Industrial Internet of Things (IIoT) and Industry 4.0 have increased the need to use powerfull network technologies in industrial automation. The growing communication in industrial automation is harnessing the productivity and efficiency of manufacturing and process automation with minimum human intervention. Due to the ongoing evolution of industrial networks from Fieldbus technologies to Ethernet, the new opportunity has emerged to integrate the Software Defined Networking (SDN) technique. In this paper, we provide a brief overview of SDN in the domain of industrial automation. We propose a network architecture called 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: Pro-active Flow Installation Scheme (PFIS) and Hybrid Flow-Installation Scheme (HFIS). We analytically quantify the proposed solutions in alleviating the overhead incurred from the flow setup cost. The analytical model is verified through monte carlo simulations. We also evaluate the SDIAN architecture and analyze the network performance of the modified topology using an emulator called Mininet. We further list and motivate SDIAN features and in particular report on an experimental food processing plant demonstration featuring Raspberry PIs (RPIs) instead of traditional Programmable Logic Controllers (PLCs). Our demonstration exemplifies the characteristics of SDIAN.

Index Terms— Controller, Industry Network, Open Flow, Software Defined Networking, Programmable Logic Controller

1 Introduction

TRAINED network experts have typically not been working on large automated machineries. Also, traditional automation machinery is not designed to support extensive network connections. The existing industrial networks are highly decentralized, rigid and complex to manage due to the tight coupling of the data and control plane that are embedded on the same device. Nodes are configured individually and interconnections remain static. The current hierarchical structure consists of three network levels with various networking technologies and protocols. Communication among these levels through data mapping incurs significant challenge and latency. The traditional structure requires offline manual network control and management, which is time-consuming and error-prone. It hinders in bringing live changes as per the updated requirement of the production line. The resolution of medium access control (MAC) address and routing address during data forwarding leads to possible incompatibility added by different vendors and products needs extra effort to resolve.

These immanent intricacies are posing significant challenges in achieving the fourth-generation industry revolution (FGIR). FGIR is underpinned by the intelligent manufacturing (IM) enabling customized production. To ensure a smooth adaptation toward the changes in product orders, IM aims to reconstruct the industrial plant by decoupling the manufacturing entities. To attain optimal production, FGIR plans to ensure live monitoring of machine status, environmental factors and manufacturing parameters that can be fed into the upper layer application enabling advanced and accurate periodic faults detection. These features will help to arrange timely maintenance with near zero downtime. The future industrial network also requires flexible adaptation toward the changes in the manufacturing entities and location. Therefore, it is necessary to replace the current heterogeneous hierarchical network structure with the flat IP-based networking to ensure real-time transmission of critical messages and simple data mapping. Furthermore, one of the timeliest requirements is the online dynamic configuration of the manufacturing systems and networking devices to meet the real-time change of the product order.

To this context, Software Defined Networks (SDNs) could bring some unique solutions by simplifying the network architecture on factory floors. 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 are becoming a hot topic at the enterprise level since 2010. To our knowledge, SDN solutions are new to the industrial automation domain. SDNs allow the creation of 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 aforementioned characteristics of SDNs, the current "touch many, configure many" model is being evolved into "touch one, configure many" [11]. In transitioning to a software-defined network, the key challenges involve changing the traditional practices in industrial automation on the factory floor [12, 13]. That means providing relevant employees with the tools and knowledge to move toward a new, more intelligent infrastructure. In the end, the skilled trades are a critical part of the repair and replacement of equipment in the factory.

The contribution of this paper can be summarized as

1. We introduce a novel industrial network framework based on SDN describing the 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 Monte-Carlo simulation.
4. We validate the proposed scheme by running an experiment in an emulated environment using Mininet [14].
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.

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 exmine 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 well modelled emulation scenario and a report on the experimental setup in food processing plant demonstrator is presented. Finally Section 6 concludes the paper. This manuscript is the extended version of the paper presented in [15].

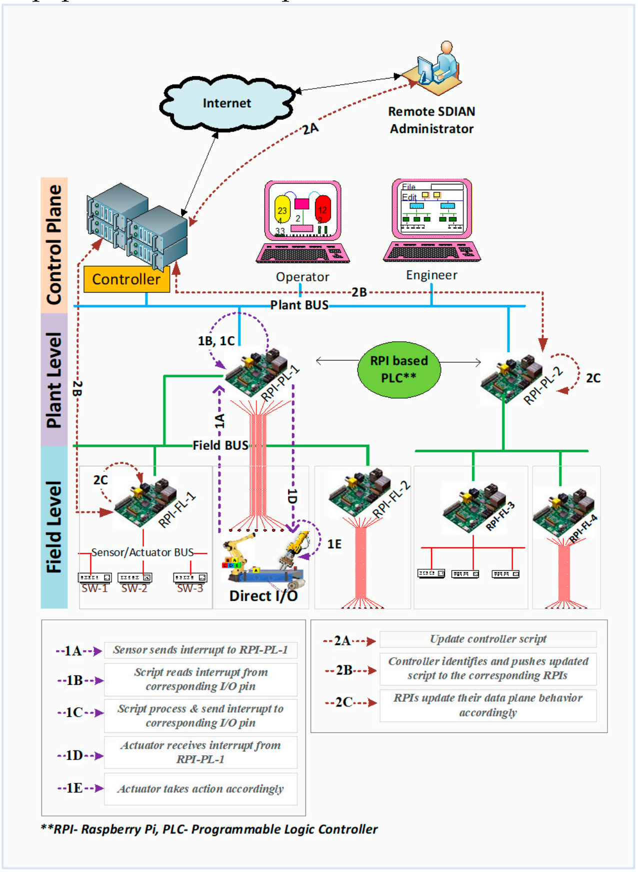

Internet
Remote SDIAN Administrator
2A
Control Plane
Controller
Operator
Engineer
2B
Plant BUS
Plant Level
1B, 1C
RPI-PL-1
RPI based PLC**
RPI-PL-2
2C
2B
1A
Field BUS
1D
Field Level
2C
RPI-FL-1
Sensor/Actuator BUS
SW-1
SW-2
SW-3
Direct I/O
1E
RPI-FL-2
RPI-FL-3
RPI-FL-4
1A Sensor sends interrupt to RPI-PL-1
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

Fig. 1. Conceptual Architecture of SDIAN

2 Architecture & 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 our proposed SDIAN and the packet dissemination model within 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 PLCs are replaced with the RPi based PLCs. Sensors and actuators are interfaced with field level RPi based PLCs, except the direct I/Os. These 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.

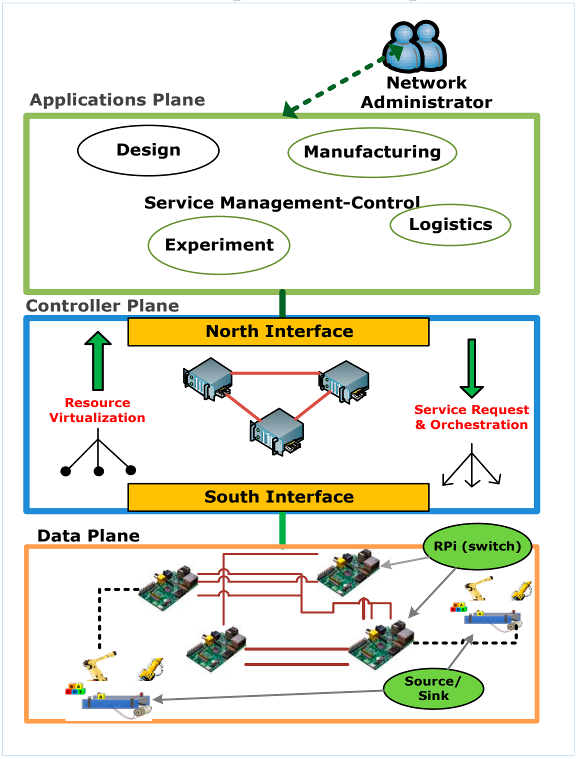


Fig. 2. SDIAN Communication Framework

2.2 SDIAN Communication Framework

Figure 2 shows the three-layer communication framework of SDIAN. 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 installed in the flow table. In this framework, RPis are connected through a mesh topology. We deliberately use 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 [16], 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 1 presents the brief summary of different compo-

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 June 2018 doi:10.20944/preprints201806.0138.v1

Peer-reviewed version available at J. Sens. Actuator Netw. 2018, 7, 33; doi:10.3390/jsan7030033

nents of SDIAN architecture.

TABLE 1
SDIAN 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 explicitely specified action immediately after receving an interrupt from RPi	Data Plane
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 behaviour.	Between Application and Control 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 notificaiton	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
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 others, industrial networking environments have some 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. These unique features, compared to other communication fields, represent significant disparities and poses both challenges and opportunities in implementing SDN based industrial Ethernet infrastructure. By the inclusion of SDN, it inherently brings the opportunity to resolve the reliability, manageability and ease of use issues which makes it challenging to ensure 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, this becomes worse incurring higher latency to resolve the first packet. From the different empirical study [14], it is realised 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 generate packet events to the controller can be high, in particular, when the switch simultaneously processes forwarding rules received from the controller.

To this end, we provide two solutions for flow creation, from which the network administrators can determine the appropriate flow mapping based on their predilection and the requirements of application. In the first solution, we use the innovative idea of mixing reactive and pro-active flow installation methods. This is referred to as *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 [14]. 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 faces only one inbound and outbound latency irrespective of the number of relay nodes it goes through before it reaches the destination.

Peer-reviewed version available at J. Sens. Actuator Netw. 2018, 7, 33; doi:10.3390/jsan7030033

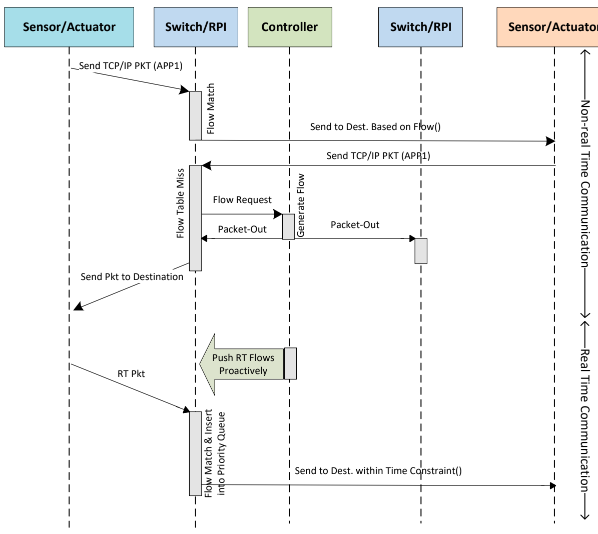


Fig. 3. Packet Dissemination Model of SDIAN

The precise synchronization of processes is underpinning 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 pro-active flow installation scheme (PFIS) catering for these delay sensitive traffics providing smooth and precise synchronization. In this case, we adopted the direct rule offload method. The controller sends the flow installation packet for all predetermined critical delay-sensitive traffic to the switches instantaneously after the discovery of a switch. This pre-installation happens during the convergence of the network. For further clarification, we present the packet dissemination model of SDIAN 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 although in the test bed implementation, the data channel and control channel are separate, for simplification of drawing this is not portrayed in Figure 4. As shown in Figure 4(a), the switch S_1 receives a data packet from a field level device. For this particular 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 particular packet and responds with packet-out to all the intermediate switches along this path (Figure 4(b)). Therefore, as shown in Figure 4(c), 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.

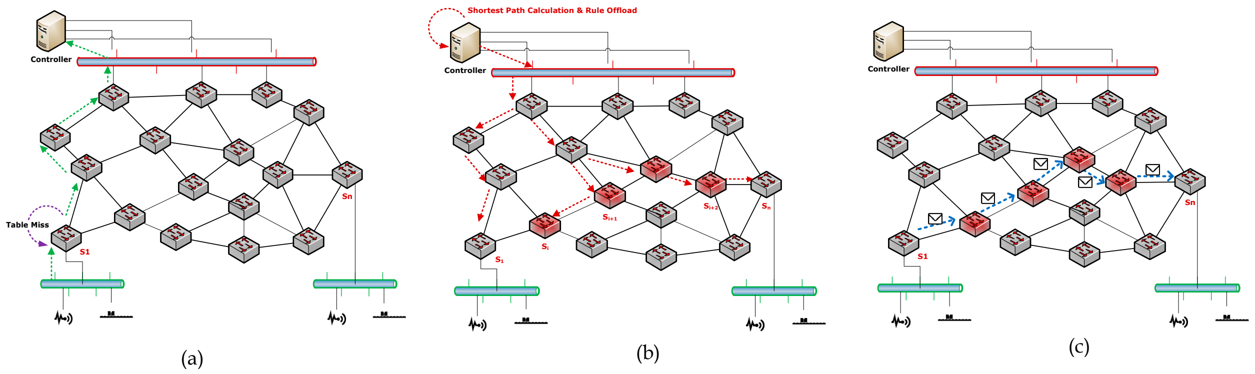


Fig. 4. Working Mechanism of HFIS a) Table Look Up b) FE & RO c) Reaches destination

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. Then, 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 $F^d\colon S \times S \rightarrow \boldsymbol{R}^+$ 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:

$$\text{if } (i, j) \notin L \text{ then } F_{ij}^d = 0 \qquad (1)$$

2) along one direction:

$$\text{if } F_{ij}^d > 0 \text{ then } F_{ji}^d = 0 \qquad (2)$$

3) at source s_1:

$$f^{(d)} + \sum_{k=1}^{n} F_{ks_1}^d = \sum_{j=1}^{n} F_{s_1 j}^d \qquad (3)$$

relay node $i \neq s_1, s_n$:

$$\sum_{j=1}^{n} F_{ij}^d = \sum_{k=1}^{n} F_{ki}^d \qquad (4)$$

4) at destination s_n:

$$\sum_{k=1}^{n} F_{ks_n}^d = \sum_{j=1}^{n} F_{s_n j}^d + f^{(d)} \qquad (5)$$

The constraint in (1) ensures that for each link $(i, j) \notin L$, $F_{ij}^d = 0$ and in particular, for each undirected link $(i, j) \in L$, the constraint in (2) says if $F_{ij}^d > 0$ then $F_{ji}^d = 0$ or if $F_{ji}^d > 0$ then $F_{ij}^d = 0$. The traffic constraints defined in eq. (3), (4) and (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, $G_{F^{(d)}} = (S_{F^{(d)}}, L_{F^{(d)}})$. $G_{F^{(d)}}$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 (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 [17, 18] presented in [19].

3.1.1 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).

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

In order to comply with the constraints specified in (1) ~ (5), (6) can more specifically be stated as –

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

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 eq. (7), we consider the dual (L_1 Dual) of (6) in terms of Lagrange multipliers (U_i's) to find the shortest path routing –

$$\max_{U} U_1 \qquad (8)$$

Subject to,

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

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

$$\text{if } F_{ij}^* > 0, \text{then } U_i^* - U_j^* = x_{ij} \qquad (9)$$

and

$$\text{if } F_{ij}^* = 0, \text{then } U_i^* - U_j^* < x_{ij} \qquad (10)$$

Based on these relations, we can define the following properties of the optimal solution ($U_i^*{}'s$) 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, $\alpha_{ou}$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:

$$\min_{\alpha} \left\{ \alpha_{in} + \alpha_{ou} + \sum_{k=1}^{m} \{\alpha_p^{S_k}\} + 2 \times \beta + \gamma \right\} \qquad (11)$$

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 $S_{F(d)}^* = \{S_i, S_{i+1}, \dots \dots, S_{i+m}\}$.

According to HFIS, only the first switch generates the packet event to the controller and installs the flow accordingly. 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 the eq. (11) as follows-

$$\min_{\alpha} \{\alpha_{in} + \alpha_{ou} + m \times \alpha_p + 2 \times \beta + \gamma\} \qquad (12)$$

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* [16], 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 [16]. 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 pro-active flow installation scheme, 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 over 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 $\alpha_{ou}$is fabricated from the outcome of a comprehensive measurement study [20, 21] 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 variant and skewed with a same mean and median of 1.09 (ms) and a standard deviation of 0.18 (ms). Assuming the simulation is running in a ten (10) switches control network of a single small scale plant, the number of hops (m) in 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 ($\approx 0.1\ ms$). 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 5(a) – 5(c) respectively show the histogram of Monte Carlo simulation results of three flow installation schemes: hybrid, reactive and pro-active. The bin size in Figure 5(a) and Figure 5(b) is 5 (ms) whereas in Figure 5(c) it is 0.035 (ms). The ascendancy acquired by using HFIS and PFIS over the Reactive Flow Installation Scheme (RFIS) is discernible. 95% packets are resolved within 3.28 (ms) using the HFIS and within 0.19 (ms) using the PFIS. Table II presents the summary statistics of the simulation results shown in Figure 5

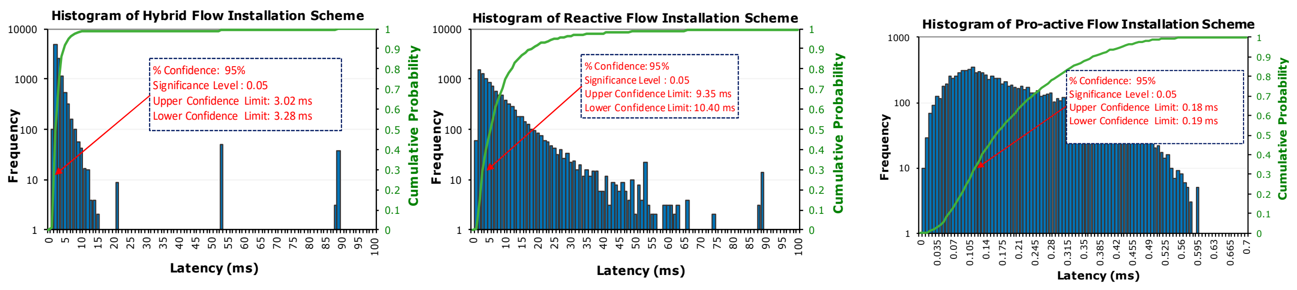
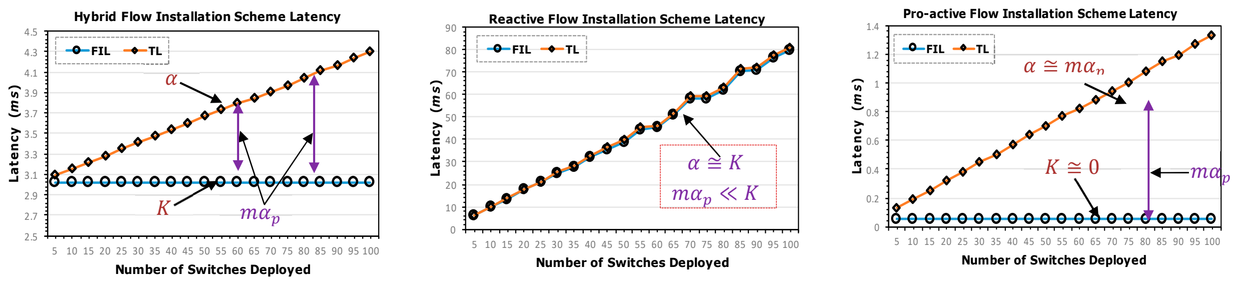


Fig. 5 Histogram of Monte Carlo Simulation Results of a) Hybrid Flow Insallation Scheme (HFIS) b) Reactive Flow Installation Scheme (RFIS) and c) Pro-active Flow Installation Scheme (PFIS)



*FIL – Flow Installation Latency *TL – Total Latency

Fig. 6 Mean Latency for varied number of switches a) Hybrid b) Reactive c) Pro-active

To see the implication of *Lemma 2*, we repeat the simulation for varying number of switches ($5 \leq S \leq 100$) to this end. After performing all the runs of Monte-Carlo simulation, we average the results obtained for each value of S as indicated in Figure 6. Figure 6(a) depicts that with HFIS, the total flow installation latency ($K = \alpha_{in} + \alpha_{ou} + 2 \times \beta + \gamma$) is constant irrespective of the size of the network; therefore, the total latency (α) is directly proportional to $m \times \alpha_p$. Figure 6(b) presents the latency for RFIS. In the worst case scenario of this method, each switch in a route could face table miss with flow setup cost. Therefore, the total latency is fully dominated by the flow installation overhead ($\alpha \cong K,\ m\alpha_p \ll K$). The latency of PFIS is investigated in Figure 6(c). 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 caseα is directly proportional to $m\alpha_p$ with $\alpha \cong m\alpha_p$ and $K \cong 0$.

TABLE 2
SUMMARY STATISTICS OF STOCHASTIC ANALYSIS

		HFIS	RFIS	PFIS
Sample Size		10000	10000	10000
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, Percen-tiles, Intervals	90% Interval	Q(.05)=1.17 Q(.95)=6.01	Q(.05)=1.34 Q(.95)=24.9	Q(.05)=0.04 Q(.95)=0.42
	95% Interval	Q(.025)=1.08 Q(.975)=7.84	Q(.025)=1.22 Q(.975)=35.61	Q(.025)=.03 Q(.975)=.47
95% CI for the Mean	Upper Limit	3.0210	9.5795	0.1883
	Lower Limit	3.2839	10.7175	0.1930

4.1 Discussion

By following the result presented in Figure 5, 6 and Table 2, we can deduce 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 size of the network. The upper and lower limit of the 95% Confidence Interval (CI) of HFIS in a network of ten (10) switches are 3.02 ms and 3.28 ms respectively, indicating a stable deterministic condition. For consistent real-time performance in transporting deterministic delay sensitive traffic, we can apply PFIS, while we can use HFIS to provision rest of the traffic sustaining the dynamic behavior of the SDN network.

5 Experiments

In this section, we first present the network performance of the target mesh topology using a well modelled emulation scenario and then we 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 [22] 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 open-flow switch#2 (source) to open-flow 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 three different flow installation schemes. We present the result in Figure 8 and 9. From the result it is obvious that hybrid flow installation scheme retains a consistent low latency and high success rate as well as the flexibility and dynamic behavior of SDN.

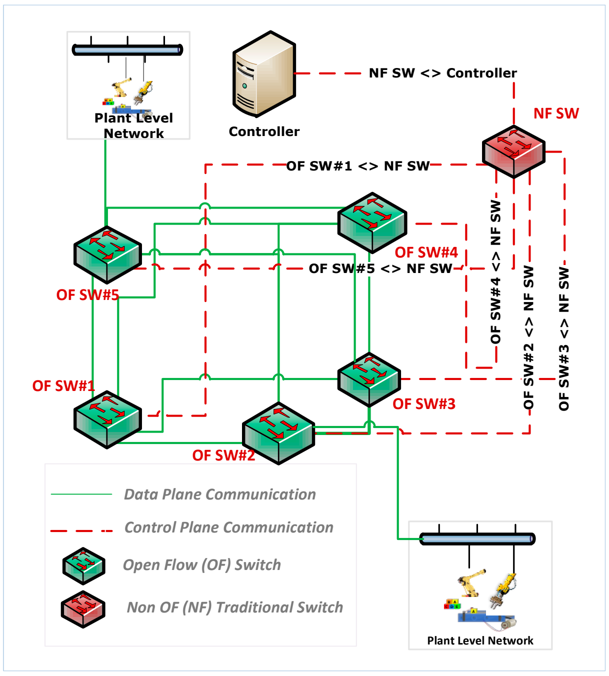


Fig. 7 Network Setup in Mininet

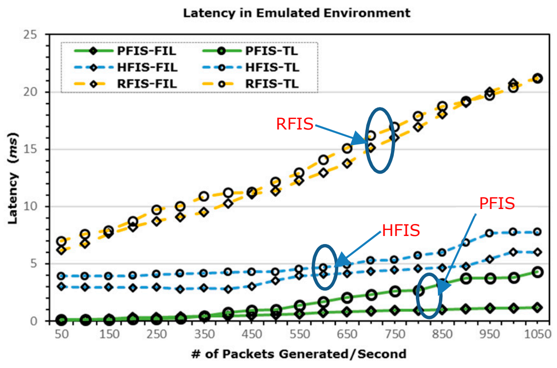


Fig. 8 Latency in Emulated Environment

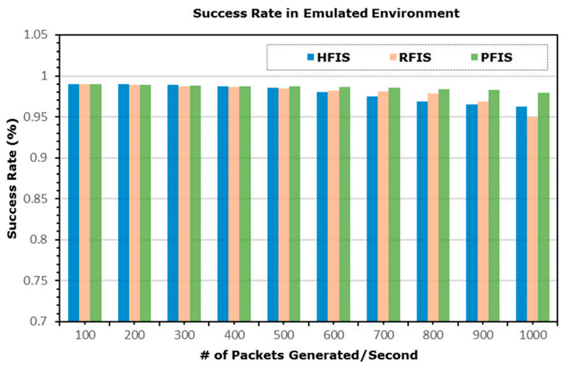


Fig. 9 Success Rate in Emulated Environment

5.2 TEST BED IMPLEMENTATION

We design and implement the test bed experiment to study the performance of the proposed SDIAN model. We conduct the test on a food processing plant demonstrator. 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 further 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 11 presents the collage of a few snapshots of our test bed setup. It briefly demonstrates 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.

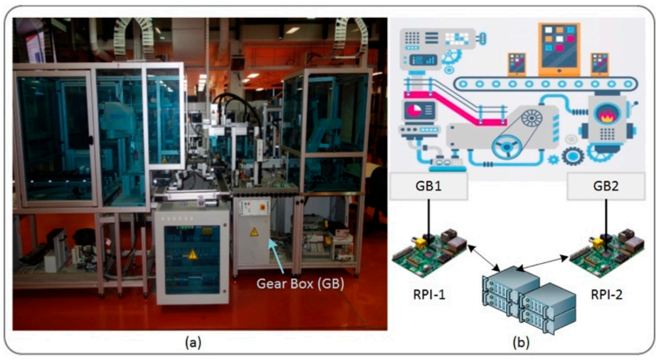
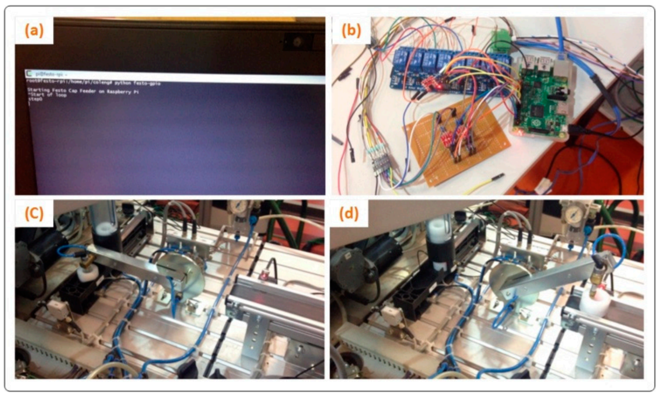


Fig. 10 a) Festo-based Food Processing Plant Demonstrator b) Deployment Diagram of the Test Bed



```
root@festo-rpi:/home/pi/coleng# python festo-gpio
Starting Festo Cap Feeder on Raspberry Pi
*Start of loop
step0
```

Fig. 11 Clockwise from top left: a) Python Script b) Interfacing with Gear Box c, d) Task Execution by Robotic Arm

6 Conclusion

In this paper, we have explained the characteristics of SDN in the context of industrial automation. We highlighted on carefully designing two flow installation schemes to ensure the precise synchronization of the processes in industrial automation as well as presenting the potential benefits and opportunities of SDN. Furthermore, we have presented our architectural model involving 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 side as well as integrating SDN support with existing software platforms.

Some of its limitations constrain the proposed framework at this stage. For simplification, we limit our work to wired network technology although it is evident to have an integration of wireless (e.g. sensor network) and wired network to achieve a unified architecture. The inclusion of wireless networks will open many challenges including the seamless integration between the controllers across two domains. We also have limited our scope to one single plant; therefore the validation of using multiple controllers across multiple plants and the east-west communication are left unexplored. In the proposed framework, each Raspberry Pi based PLC is also used as an SDN switch, in future we may consider the use of lightweight SDN switches like Zodiac FX, which could overrule the chances of getting bottleneck across each Raspberry Pi and clearly separate the forwarding devices from underlying field level sensors and actuators.

ACKNOWLEDGMENT

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.

REFERENCES

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Communications Surveys & Tutorials,* vol. 17, no. 1, pp. 27-51, 2015.
[2] T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang, and J. Liu, "A Survey on Large-scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges," *IEEE Communications Surveys & Tutorials,* vol. PP, no. 99, pp. 1-1, 2016.
[3] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials,* vol. 16, no. 4, pp. 2181-2206, 2014.
[4] B. Raghavan *et al.*, "Software-defined internet architecture: decoupling architecture from infrastructure," presented at the Proceedings of the 11th ACM Workshop on Hot Topics in Networks, Redmond, Washington, 2012.
[5] T. Skeie, S. Johannessen, and O. Holmeide, "Timeliness of real-time IP communication in switched industrial Ethernet networks," *IEEE Transactions on Industrial Informatics,* vol. 2, no. 1, pp. 25-39, 2006.
[6] J. D. Decotignie, "Ethernet-Based Real-Time and Industrial Communications," *Proceedings of the IEEE,* vol. 93, no. 6, pp. 1102-1117, 2005.
[7] C. Rojas, P. Morell, and D. E. Sales, "Guidelines for Industrial Ethernet infrastructure implementation: A control engineer's guide," in *Cement Industry Technical Conference, IEEE-IAS/PCA 52nd*, 2010, pp. 1-18.
[8] V. C. Gungor and G. P. Hancke, "Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches," *IEEE Transactions on Industrial Electronics,* vol. 56, no. 10, pp. 4258-4265, 2009.
[9] L. Hou and N. W. Bergmann, "Novel Industrial Wireless Sensor Networks for Machine Condition Monitoring and Fault Diagnosis," *IEEE Transactions on Instrumentation and Measurement,* vol. 61, no. 10, pp. 2787-2798, 2012.
[10] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered Ethernet (TTE) design," in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, 2005, pp. 22-33.
[11] D. Cronberger. (2015). *Software Defined Networks*. Available: http://www.industrial-ip.org/en/industrial-ip/convergence/software-defined-networks
[12] K. Ahmed, J. O. Blech, M. A. Gregory, and H. Schmidt, "Software Defined Networking for Communication and Control of Cyber-Physical Systems," in *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*, 2015, pp. 803-808.
[13] D. Li, M. T. Zhou, P. Zeng, M. Yang, Y. Zhang, and H. Yu. (2016, 10). *Green and reliable software-defined industrial networks*.
[14] O. N. Fundation. Software-defined networking: The new norm for networks [Online].
[15] K. Ahmed, N. S. Nafi, J. O. Blech, M. A. Gregory, and H. Schmidt, "Software defined industry automation networks," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, 2017, pp. 1-3.
[16] Opennetworking.org. (2016). *OpenFlow - Open Networking Foundation* [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow.
[17] F. P. Kelly, "Network routing," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences,* vol. 337, no. 1647, pp. 343-367, 1991.
[18] W. Yufei, W. Zheng, and Z. Leah, "Internet traffic engineering without full mesh overlaying," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2001, vol. 1, pp. 565-571 vol.1.
[19] Y. Li, Z. L. Zhang, and D. Boley, "From Shortest-Path to All-Path: The Routing Continuum Theory and Its Applications," *IEEE Transactions on Parallel and Distributed Systems,* vol. 25, no. 7, pp. 1745-1755, 2014.
[20] K. He *et al.*, "Measuring control plane latency in SDN-enabled switches," presented at the Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, Santa Clara, California, 2015.
[21] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control Plane Latency With SDN Network Hypervisors: The Cost of Virtualization," *IEEE Transactions on Network and Service Management,* vol. 13, no. 3, pp. 366-380, 2016.
[22] R. S. F. Community. (2014). *Ryu SDN Framework* [Online]. Available: https://osrg.github.io/ryu/

Khandakar Ahmed (M'2014, S'2011) is a Lecturer in Discipline of IT, School of Engineering and Science, Victoria University, Melbourne, Australia. He completed his PhD Degree from the RMIT University, Australia in December 2014. During his PhD, he was an active scholar in the network research group of School of Engineering, where he explored Data-Centric Storage (DCS) in Wireless Sensor Network (WSN). Before joining Victoria University, Khandakar was a full-time Lecturer with the School of IT and Engineering (SITE), Melbourne Institute of Technology (MIT), Melbourne and a researcher in Australia-India Research Centre for Automation Software Engineering (AICAUSE), RMIT University (2016-2017). He has been with AICAUSE as a Post-Doctoral researcher before he takes the full-time position at MIT during 2015-2016. He received his M.Sc. in Networking and e-Business Centred Computing (NeBCC) under the joint consortia of University of Reading, UK; Aristotle University of Thessaloniki, Greece and Charles III University of Madrid (UC3M), Spain in 2011. During his M.Sc. dissertation, he worked as a research assistance in Advance Computing and Emerging Technology (ACET) Centre, University of Reading (2010-2011). He has published more than 30 peer-reviewed journals and conference papers. His research interests include Software Defined Networking (SDN), Wireless Sensor Network (WSN), Internet of Things (IoT) and Cyber Security. He is serving as the program and publication chair of 2017 International Telecommunication Networks and Applications Conference (ITNAC'2017). He is also serving as the member of the editorial board of Australian Journal of Telecommunications and the Digital Economy (AJTDE). He has also been serving as a reviewer for several international A* journals and conferences.

Jan Olaf Blech is a research fellow in the Australia-India Research Centre for Automation Software Engineering at RMIT University in Australia. His previous affiliations include fortiss GmbH in Germany, Verimag in France, TU Kaiserslautern, TU Berlin and the University of Karlsruhe in Germany. Dr. Blech has worked on several publicly and industry funded projects and collaborated with major industrial automation and automotive companies in Europe and Australia. His experience comprises research and technical project leadership roles. His main areas of research are software engineering, formal methods and embedded systems with applications in industrial automation, automotive and aviation. He has (co-)authored over 60 scientific publications. Dr. Blech is a member of IEEE and ACM.

Mark A Gregory (SM'99) is a Fellow of the Institute of Engineers Australia and a Senior Member of the IEEE. Mark A Gregory received a Ph.D. from RMIT University, Melbourne, Australia in 2008, where he is a Senior Lecturer in the School of Engineering. In 2009, he received an Australian Learning and Teaching Council Citation for an outstanding contribution to teaching and learning. He is the Managing Editor of two international journals (AJTDE and IJICTA) and the General Co-Chair of ITNAC. Research interests include telecommunications, network design and technical risk.

Heinrich Schmidt is a director of Australia India Research Centre for Automation Software Engineering (AICAUSE) and RMIT eResearch Office. He has experience over 35 years in software R&D and R&D management, over 150 publications, a focus on distributed software architectures for real-time, embedded, connected and/or high-performance parallel or cloud systems. He developed formal foundations and new algorithms where they help in practice to improve methods and tools for designing, testing or managing systems. His leadership in industry-university collaborations combined with change as a motor of innovative thinking and patience that gets things done and builds long-term capability and impact. He was a Visiting Professor or Fellow in several institutions across the world incl. USA, DE, SE, FR, AUS. His specialties include Component-based software engineering, architecture and change management. Special expertise in virtual enterprise architecture, concurrency, reliability & performance prediction, testing, eResearch, grid and cloud computing.