# A Hierarchical Control Plane for Software-Defined Networks-based Industrial Control Systems

Béla Genge and Piroska Haller
Department of Informatics
"Petru Maior" University of Tîrgu Mureş
N. Iorga, No. 1, Tîrgu Mureş, Mureş, Romania, 540088
Email: bela.genge@ing.upm.ro, phaller@upm.ro

***Abstract*—Modern Industrial Control Systems (ICS) integrate advanced solutions from the field of traditional IP networks, i.e., Software-Defined Networks (SDN), in order to increase the security and resilience of communication infrastructures. Despite their clear advantages, such solutions also expose ICS to common cyber threats that may have a dramatic impact on the functioning of critical infrastructures, e.g., the power grid. As a response to these issues, this work develops a novel hierarchical SDN control plane for ICS. The approach builds on the features of a novel SDN controller named OptimalFlow that redesigns the network according to the solutions delivered by an integer linear programming (ILP) optimization problem. The developed ILP problem encapsulates a shortest path routing objective and harmonizes ICS flow requirements including quality of service, security of communications, and reliability. OptimalFlow exposes two communication interfaces to enable a hierarchical control plane. Its northbound interface reduces a complete switch infrastructure to an emulated (software) switch, while its southbound interface connects to an OpenFlow controller to enable the monitoring and control of real/emulated switches. Extensive experimental and numerical results demonstrate the effectiveness of the developed scheme.**

***Index Terms*—Industrial Control Systems, Software-Defined Networks, Resilience, Security, Reliability.**

## I. Introduction

THE massive proliferation of traditional Information and Communication Technologies (ICT) into the architecture of Industrial Control Systems (ICS) will constitute a turning point in the operation and functioning of modern ICS. This will provide the building blocks for novel infrastructural paradigms, and will facilitate innovative applications such as robust voltage control, renewable energy programs, and electric vehicles. Despite these clear advantages, however, the pervasive integration of commodity off the shelf ICT hardware and software will also expose ICS to new threats [1], [2], [3]. These may have a significant impact on the functioning of critical infrastructures, e.g., the power grid, and may lead to the failure of services, to economic and, possibly, to human losses. As a response, several recently developed techniques address the security of ICS. Furthermore, the NIST Guide to Industrial Control Systems Security [4] recommends integrating different solutions into a defense-in-depth security strategy. On the other hand, ICS require communication resilience solutions that ensure their normal functioning even in the presence of disturbances such as failure and disruptive cyber attacks.

ISBN 978-3-901882-83-8 © 2016 IFIP

To alleviate the aforementioned issues we develop a novel hierarchical SDN control plane for ICS. The proposed scheme provides a scalable solution for ICS distributed across large geographical areas. The approach builds on the features of a novel SDN controller named *OptimalFlow*, which monitors a single SDN domain, and redesigns the network according to the solutions delivered by an integer linear programming (ILP) optimization problem. The developed ILP problem encapsulates a shortest path routing objective and harmonizes ICS flow requirements including quality of service, security, and reliability. *OptimalFlow* exposes two communication interfaces to enable a hierarchical control plane. Its northbound interface reduces a complete switch infrastructure to an emulated (software) SDN switch, which exposes the domain's edge ports to the upper tiers and can be monitored and controlled through the *OpenFlow* protocol. We believe that this is a salient feature of the developed scheme, since it facilitates *OptimalFlow*'s adoption in any installation supporting the *OpenFlow* protocol. *OptimalFlow*'s southbound interface connects to an *OpenFlow* controller, to monitor and control a network of emulated or real SDN switches. In order to minimize the impact of network updates on ICS flows we further propose two algorithms. Algorithm 1 dynamically reduces the set of variables in the optimization problem such that only the flows that are affected by a disturbance are optimally redistributed. Algorithm 2 constructs a dependency graph for network updates in order to avoid link congestion. *OptimalFlow* is implemented in the `Python` language and its effectiveness is verified through experiments conducted with Mininet [5] and with the AIMMS optimization software [6].

The rest of this paper is organized as follows. Related Work is briefly discussed in Section II, while the proposed scheme is presented in Section III. Experimental results are detailed in Section IV and the paper concludes in Section V.

## II. Related Work

Several recent studies demonstrated the benefits of SDN-enabled communication infrastructures and identified key challenges in adopting this emerging technology. Yonghong Fu *et al.* [7] developed Orion, a hybrid hierarchical control plane for large-scale networks. Orion defines three planes: the domain physical network, the tier 0 control plane consisting of area controllers, and the tier 1 control plane consisting of a distributed set of domain controllers. While Orion addresses the routing problem in large-scale multi-domain SDN infrastructures, *OptimalFlow* focuses on the requirements of ICS communications including security, reliability and resilience to disturbances. Therefore, similarly to Orion, *OptimalFlow* proposes a hierarchical control plane architecture, but expands the routing criteria from traditional ICT with those specific to ICS. Tuncer *et al.* [8] developed an SDN-based management and control framework for backbone networks. The approach followed a hierarchical and modular structure to support large-scale topologies and the simple integration of various management applications. The work of Tuncer also proposed a network planning algorithm based on the uncapacitated facility location problem. In [9] the authors developed Dionysus, a system for consistent network updates in SDN. Dionysus builds the graph of network update dependencies and schedules these updates by taking into account the performances of network switches. To eliminate packet losses [10] proposed zUpdate, a solution that uses packet labeling for zero packet losses during network updates. In comparison to these works, we believe that *OptimalFlow*, on one hand, and Dionysus and zUpdate on the other hand expose complementary features. Particularly, the hierarchical control plane and the network optimization problem proposed in this work could be extended with Dionysus and zUpdate and their ability to provision network updates with minimum (zero) packet losses.

In the industrial sector, Goodney, *et al.* [11] showed the high degree of network flexibility that can be achieved by adopting SDN for phasor measurement unit (PMU) communications. The benefits of industrial SDN were further demonstrated in a test infrastructure comprising IEC61850-based electrical system [12]. Finally, the work of Dorsch, *et al.* [13] analyzed the advantages and the possible disadvantages of adopting SDN in industrial networks. The authors of [13] acknowledge the benefits of network management applications, quality of service optimization and the enhancement in the system's resilience, but raise serious concerns pertaining to the increased risks of cyber attacks against SDN's centralized controllers.

## III. Proposed Approach

### A. Architectural Overview

Nowadays, industrial operators are moving towards the adoption of advanced networking solutions from the field of traditional IP networking in order to increase the security and resilience of communication infrastructures. Solutions including Multi Protocol Label Switching (MPLS) [14] and Software-Defined Networks (SDN) [15] have recently been integrated into ICS and have replaced older implementations based on Frame Relay and Asynchronous Transfer Mode (ATM). Nevertheless, communications in large-scale ICS usually cross the boundaries of one administrative domain. In fact, in order to deliver fault-tolerant communications, several lines may be leased from different Internet Service Providers (ISP). Traffic crossing an ISP's networking infrastructure will therefore be subject to the constraints and routing decision specific to each domain. Based on these assumptions we propose a hierarchical SDN controller scheme that embraces the multi-domain characteristic of ICS networks by means of an n-Tier controller architecture, as shown in Fig. 1. The bottom tier represents the physical infrastructure and consists of network switches and links. This represents the data forwarding plane and can be structured in several domains. Each SDN domain includes a *FlowControl* unit that: (i) monitors the underlying domain for changes in network parameter values, e.g., the status of switch ports; (ii) changes the set of installed flows according to the solutions delivered by an optimization problem aimed to preserve critical communication parameters; and (iii) transparently exposes the edge ports of an entire SDN domain to the upper tiers by means of an emulated SDN switch accessible through the *OpenFlow* protocol.

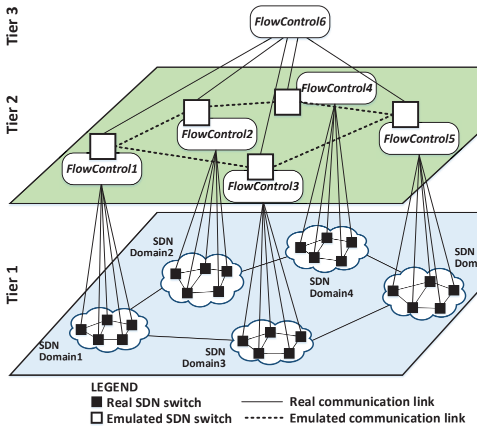


Fig. 1. Architectural overview of the proposed scheme.

### B. The FlowControl Unit

The *FlowControl* unit (depicted in Fig. 2) includes two software controllers: *OpenFlow* and *OptimalFlow*. The *OpenFlow* controller is a traditional SDN controller that monitors and controls an underlying SDN network using the *OpenFlow* protocol. The *OpenFlow* controller configures the forwarding plane of SDN switches and exposes a communication interface that may be used to implement specially-tailored network traffic control strategies. The main contribution of this work, however, lies in the architecture and in the features exposed by the *OptimalFlow* controller. *OptimalFlow* implements a novel network traffic optimization problem that, as a response to disturbances, computes a new optimal distribution of the affected flows, while preserving the requirements of ICS flows, e.g., security, reliability. Architecturally, it implements four main modules: *SDNStateHandler*, *OFControllerCommunication*, *OptimalSolver*, and *OpenFlowSwitchEmulator*. Its main module is the *SDNStateHandler*, which maintains an in-memory representation of the underlying SDN network and repeatedly issues calls to the *OFControllerCommunication* module to update its internal state. In the case a change is detected, it issues a call to the *OptimalSolver* module

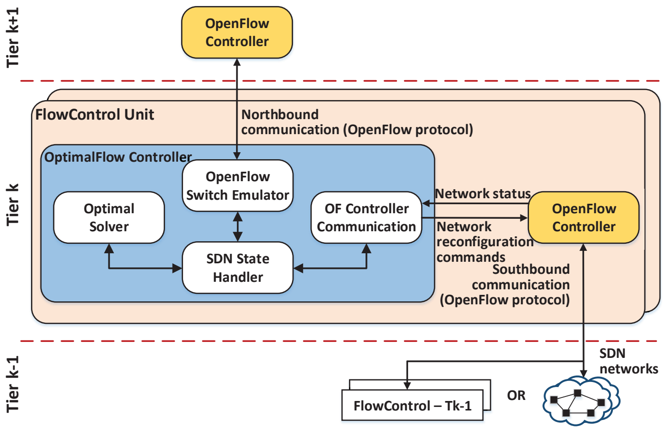


Fig. 2. Detailed architecture of the FlowControl unit.

TABLE I
KEY NOTATIONS

| | Symbol | Description |
|---|---|---|
| *Sets* | $I, J, B$ | Flows, SDN switches, and links. |
| | $S$ | Security features. |
| *Parameters* | $d_i$ | Demand of flow $i$. |
| | $u^b_{jl}$ | Capacity of link $(j, l, b)$. |
| | $x^A_{ij}, x^E_{ji}$ | Access and egress flow connectivity. |
| | $p^s_i$ | Security property requirements for flow $i$. |
| | $y^{bs}_{jl}$ | Security properties of link $(j, l, b)$. |
| | $q_i$ | Minimum reliability requirement for flow $i$. |
| | $r^b_{jl}$ | Link failure probability. |
| | $\alpha_i$ | Penalty value for disconnected flow $i$. |
| *Variables* | $t^{bi}_{jl}$ | Selection of flow $i$ for routing on link $(j, l, b)$. |
| | $w^A_{ij}, w^E_{ji}$ | Selection of flow $i$ for routing betw. acc./egr. switch $j$. |
| | $o_i$ | Selection of flow $i$ for disconnection. |

to compute the optimal distribution of the flows affected by the disturbance. The new network configuration is then transmitted by the *OFControllerCommunication* module to the *OpenFlow* controller via a set of static flows that are installed in the SDN switches. The *OptimalFlow* controller exposes an *OpenFlow* northbound communication interface via its *OpenFlowSwitchEmulator* module. By doing so, the *OptimalFlow* controller is connected to upper tiers as a regular SDN switch that can be monitored and controlled via the *OpenFlow* protocol. This represents an effective strategy to build a hierarchical SDN network, where each tier adopts the same *FlowControl* software units. Furthermore, we believe that this is a salient feature of the proposed scheme, since it facilitates the provisioning of *FlowControl* without the need to change the *OpenFlow* protocol and the implementation of SDN switches/controllers.

## C. Network Model and Optimization Problem

We assume a demand matrix of flows routed between access and egress switches. The routing needs to be performed in such a way to reduce communication delays by means of selecting the shortest paths and the largest capacity links. Flows are assumed to be non-bifurcated multicommodity flows such that each flow can only be routed on one path. This is a fundamental requirement to ensure that flows may cross the boundaries of one administrative domain. A summary of key notations is tabulated in Table I.

We define $I$ to be the set of flows and $J$ to be the set of switches. Network devices, especially in the industrial sector, are usually connected by more than one link, i.e., by primary and back-up links. Therefore, we define $B$ to be the set of possible links that may connect two switches, and we use $(j, l, b)$ to denote link $b$ between switches $j$ and $l$, where $b \in B$ and $j, l \in J$. Links between switches may implement various security features such as basic packet filtering (traditional firewalls), encrypted communication links, signature/anomaly detection systems. At the same time, flows may require packets to be forwarded on links with specific security features in place. Therefore, we define $S$ as the set of security features and we use $s \in S$ to denote a specific security feature.

Next, we define the ILP problem's parameters. Let $d_i$ denote the demand of flow $i$ and $u^b_{jl}$ the capacity of link $(j, l, b)$. We assume that if switches $j$ and $l$ are not connected, then $u^b_{jl} = 0, \forall b \in B$. Then, let $x^A_{ij}$ be a binary parameter with value 1 if the access end-point of flow $i$ is connected to switch $j$, and $x^E_{ji}$ a binary parameter with value 1 if the egress end-point of flow $i$ is connected to switch $j$.

The security requirements of flow $i$ are configured with the help of the binary parameter $p^s_i$. This is 1 if flow $i$ may be routed on a link with security property $s$, and is 0, otherwise. On the other hand, the security properties that are actually installed on a particular link are defined with the binary parameter $y^{bs}_{jl}$, which is 1 if link $(j, l, b)$ implements the security property $s$, and is 0, otherwise. In the problem at hand we assume that flow $i$ can be routed on link $(j, l, b)$ only if at least one of the security properties configured for flow $i$ is implemented on link $(j, l, b)$. This means that, for instance, a flow that requires only the integrity security property $(s_1)$, may also be routed on a link that implements other properties as well ($s_2$, where $s_1$ is included in $s_2$), such as integrity, confidentiality. The minimum required reliability of a forwarding path for flow $i$ is defined as parameter $q_i$, which is a real number bounded between 0 and 1. The probability of link failure is defined as parameter $r^b_{jl}$, which is a real number bounded between 0 and 1. Finally, we define the $\alpha_i$ parameter as a penalty associated with disconnecting flows. In the problem at hand flows may be disconnected and not routed in the case of significant network failures, which effectively reduce the network's resources and its ability to route flows. The choice of $\alpha_i$ values will lead to a ranking in the significance of flows and it should be chosen significantly larger than the maximum possible value of the objective function's first part, as discussed later in this section.

Next, we define the problem's variables. Let $t^{bi}_{jl}$ be a binary variable with value 1 if flow $i$ is routed on link $(j, l, b)$. Let $w^A_{ij}$ be a binary variable with value 1 if the access end-point of flow $i$ is routed by switch $j$, and 0 otherwise, and the binary variable $w^E_{ji}$ with value 1 if the egress end-point of flow $i$

is routed by switch $j$, and 0 otherwise. We further define the binary variable $o_i$ with value 1 if flow $i$ is not routed due to the unavailability of communication resources, e.g., unavailable bandwidth, and 0, otherwise.

The objective of the optimization is to select the shortest routing path for each flow, while selecting the links with the largest capacities. On one hand, this is accomplished by adopting a *minimization* objective that selects the minimum number of communication links needed to route flows across an SDN domain, i.e., minimize $\sum d_i(t^{bi}_{jl} + t^{bi}_{lj})$. On the other hand, we assume the relative load on a particular link given by the formula $\frac{\sum d_i(t^{bi}_{jl}+t^{bi}_{lj})}{u^b_{jl}}$, which will ensure that the solution includes the links with the maximum capacity. The objective function's second part controls the activation of penalty values in the case of disconnected flows. Therefore, if $o_i = 1$, it activates the penalty value $\alpha_i$ within the objective function. Since $\alpha_i$ is configured as a large integer, the minimization objective will set $o_i = 1$ only if flow $i$ can no longer be routed due to insufficient resources. The objective function is thus defined as follows:

$$\min \sum_{j,l \in J, b \in B} \left( F(j,l,b) \sum_{i \in I} d_i(t^{bi}_{jl} + t^{bi}_{lj}) \right) + \sum_{i \in I} \alpha_i o_i, \quad (1)$$

where $F(j,l,b)$ is 0 if $u^b_{jl} = 0$, and is $\frac{1}{u^b_{jl}}$, otherwise. For the LP at hand the following constraints are defined:

$$w^A_{ij} \le x^A_{ij}, w^E_{ji} \le x^E_{ji}, \quad \forall i \in I, j \in J \quad (2)$$

$$\sum_{j \in J} w^A_{ij} \le 1, \sum_{j \in J} w^E_{ji} \le 1, \quad \forall i \in I \quad (3)$$

$$\sum_{j \in J} w^A_{ij} = 1 - o_i, \forall i \in I \quad (4)$$

$$w^A_{ij} - w^E_{ji} - \sum_{l \in J, b \in B} \left( t^{bi}_{jl} - t^{bi}_{lj} \right) = 0, \quad \forall j \in J, i \in I \quad (5)$$

$$\sum_{i \in I} d_i(t^{bi}_{jl} + t^{bi}_{lj}) \le u^b_{jl}, \quad \forall j,l \in J, b \in B \quad (6)$$

$$\sum_{s \in S} p^s_i y^{bs}_{jl} \ge t^{bi}_{jl}, \quad \forall i \in I, j,l \in J, b \in B \quad (7)$$

$$\prod_{j,l \in J, b \in B} (1 - r^b_{jl} t^{bi}_{jl}) \ge q_i, \quad \forall i \in I \quad (8)$$

Constraints (2) enforce that access and egress end-points are only routed by the possible switches, while constraints (3) enforce that each flow end-point is routed by only one switch. Constraints (4) ensure that in the case of insufficient resources flows are disconnected, i.e., $w^A_{ij} = 0$, $o_i = 1$. Constraints (5) denote classical multicommodity flow conservation constraints [16], which impose the selection of a continuous path between access and egress connection endpoints. Constraints (6) impose that the bandwidth required to route flows on link $(j,l,b)$ does not exceed the link capacity. Constraints (7) ensure that flow $i$ is only routed on link $(j,l,b)$ if there exists at least one security property $s$ configured in $p^s_i$ that is implemented on link $(j,l,b)$ and is configured in parameter $y^{bs}_{jl}$.

Constraints (8) are classical (serial) reliability conditions imposing that the reliability of communication links on a particular path satisfy the minimum reliability requirement $q_i$. However, we note that the multiplication of several $t$ variables in the reliability constraint (8) will yield a non-linear problem. Therefore, we apply the transformations proposed in [17] to derive a linear set of equations. We observe that since $t^{bi}_{jl}$ is a binary variable, then $(1 - r^b_{jl} t^{bi}_{jl})$ can be rewritten as $(1 - r^b_{jl})^{t^{bi}_{jl}}$. As a result, constraint (8) is redefined as:

$$\prod_{j,l \in J, b \in B} (1 - r^b_{jl})^{t^{bi}_{jl}} \ge q_i, \quad \forall i \in I \quad (9)$$

By applying the natural logarithm function on both sides of inequality (9) we obtain the following linear reliability constraints, which are adopted in the proposed ILP problem:

$$\sum_{j,l \in J, b \in B} \left[ \mathtt{ln}(1 - r^b_{jl}) t^{bi}_{jl} \right] \ge \mathtt{ln}(q_i), \quad \forall i \in I \quad (10)$$

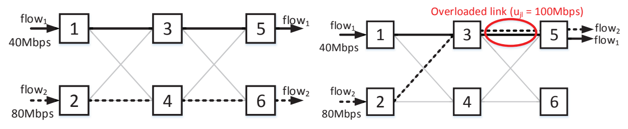
## *D. Flow Migration Reduction Algorithm*

When first launched, *OptimalFlow* computes the optimal mapping of flows on the complete network topology and it configures these flows through an *OpenFlow* controller. However, subsequent runs and most importantly, responses to disturbances might yield a complete network reconfiguration. In fact, such solutions would introduce significant communication disturbances and delays that might disrupt time-critical services and could be easily exploited by attackers. To avoid such scenarios, *OptimalFlow* solves the same optimization problem with a reduced set of variables pertaining to the affected flows. As such, *OptimalFlow* implements Algorithm 1 to reduce the set of variables to those that are affected by disturbances. Initially, the algorithm assumes that all variables are parameters with the value computed in the previous run. Then, for a specific link $(j', l', b')$ on which a disturbance is detected, *OptimalFlow* computes the set of flows $AF$ that are routed on $(j', l', b')$. For each flow $i \in AF$ *OptimalFlow* adds to the new subset $(SV)$ the variables for the flow's access and egress switches ($w^A_{ij}$, $w^E_{ji}$), and for the flow's disconnection ($o_i$). Then, the algorithm identifies all the communication paths between the flow's access and egress switches. The switches from all the paths $SW_i$ will identify the $t^{bi}_{jl}$ variables, which are added to $SV$.

While following the above procedure will significantly reduce the impact on communications, after a number of executions the network might require a complete re-organization in order to accommodate all flows and to satisfy the problem's constraints. Therefore, Algorithm 1 is extended with the function CompleteOptimNeeded($sol$) that verifies if a complete network reconfiguration is needed. If so, then *OptimalFlow* changes the status of all $w^A_{ij}$, $w^E_{ji}$, $o_i$, and $t^{bi}_{jl}$ parameters to variables and solves the complete network reconfiguration optimization problem. The CompleteOptimNeeded($sol$) may be implemented in various ways. For instance, the function might check if $\sum_i o_i > 0$, or if $\sum_i o_i \alpha_i > \beta$, where $\beta$ is a predefined limit above which a complete network reconfiguration is executed.

**Algorithm 1** Flow Migration Reduction

```
Let $(j', l', b')$ be a failed link.
ChangeOptimizationAllParameters();
$AF$ = GetAffectedFlows($j', l', b'$);
for each $i \in AF$ do
    AddAccessVariable($i$);
    AddEgressVariable($i$);
    AddDisconnectVariable($i$);
    $SW_i$ = GetAccessEgressPaths($i$);
    AddSwitchVariables($SW_i$);
end for
$sol$ = SolveOptimizationProblem();
if CompleteOptimNeeded($sol$) then
    ChangeOptimizationAllVariables();
    $sol$ = SolveOptimizationProblem();
end if
```



(a) Initial topology. (b) Intermediate topology.

Fig. 3. SDN topology reconfiguration with an overloaded link.

**Algorithm 2** Dependency Graph Construction

```
for each $j, l \in J, b \in B$ do
    $rescap^b_{jl}$ = GetLinkResidualCapacity($j, l, b$);
    $remf^{bi}_{jl}$ = GetLinkRemoveFlows($j, l, b$);
    $addf^{bi}_{jl}$ = GetLinkAddFlows($j, l, b$);
    if $rescap^b_{jl} \leq \sum_{i \in I} addf^{bi}_{jl} d_i$ then
        $dep_{ii'} = 1, \forall i \in I : addf^{bi}_{jl} = 1, \forall i' \in I : remf^{bi'}_{jl} = 1$
    end if
end for
$DEPG$ = InitializeGraph();
$OI$ = GetOrderedFlows();
for each $i \in OI$ do
    if IsNotMigrated($i$) then
        @StepToNextFlow
    end if
    for each $i' \in I$ do
        if $dep_{ii'} = 1$ and ($i'$,"REMOVE") $\notin DEPG$ then
            AddToGraph($DEPG$,($i'$,"REMOVE"));
        end if
    end for
    if ($i$,"REMOVE") $\notin DEPG$ then
        AddToGraph($DEPG$,($i$,"REMOVE"));
    end if
    AddToGraph($DEPG$,($i$,"ADD"));
end for
```

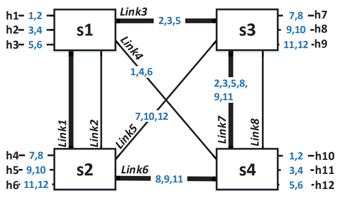
## *E. Network Update Dependency Graph Construction*

The proposed optimization problem generates a new flow configuration that must be provisioned across an SDN installation by the *OptimalFlow* controller. However, the migration of flows from one path to another needs to be carefully planned to ensure that the procedure does not introduce additional disturbances. For example, let us assume the migration of two flows, *flow*$_1$ and *flow*$_2$, with demands of $d_{flow_1}$ = 40Mbps and $d_{flow_2}$ = 80Mbps, as depicted in Fig. 3. Let us further assume that the capacity of links between switches is of $u^b_{jl}$ =100Mbps, where the number of possible links between switches is equal to one. Initially, the optimization problem routes *flow*$_1$ on links (1,3,1), (3,5,1), and *flow*$_2$ on links (2,4,1), (4,6,1). In the case of a disturbance, however, the migration of any of the two flows before the other is removed may overload links (see Fig. 3 (b)). As a solution, this work adopts a flow migration strategy that relies on the assumption of prioritized flows, which is particularly specific to ICS. For instance, communications in the core of ICS encompassing control hardware connected to critical physical processes may have a higher priority than the communication between human machine interfaces and data servers. The proposed link overload avoidance algorithm, therefore, adopts a priority-based flow migration strategy and builds a network update dependency graph according to the priority of flows. For each flow the algorithm identifies the sequence of flows that need to be deleted before this update can be performed. For instance, by further expanding the previous illustrative scenario, we assume that the priority of *flow*$_1$ is higher than the priority of *flow*$_2$. As a result, before migrating *flow*$_1$, *flow*$_2$ needs to be removed, since *flow*$_1$ has a higher priority. Then, the new paths of *flow*$_1$ and finally of *flow*$_2$'s can be configured in the network switches.

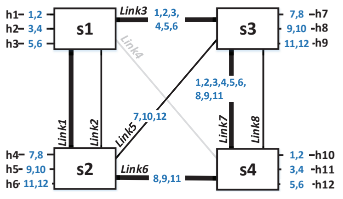
A formal description of the above steps is given in Algorithm 2. At first, the algorithm computes the residual link capacity $rescap^b_{jl}$, it stores in the binary parameter $remf^{bi}_{jl}$ the flows that will be removed from link $(j, l, b)$, and it stores in the binary parameter $addf^{bi}_{jl}$ the new flows that will be routed on link $(j, l, b)$. For each link $(j, l, b)$, if the residual capacity $rescap^b_{jl}$ is lower than the sum of flow demands $d_i$ to be routed on this link, the algorithm adds to the dependency matrix $dep_{ii'}$ of each newly added flow $i$ the flows $i'$ that will be removed from this link. Based on the dependency matrix $dep_{ii'}$, the algorithm then proceeds with the construction of the dependency graph $DEPG$ by first ordering the flows descendingly according to their priorities (set $OI$). Then, for each $i \in OI$ if $i$ needs to be migrated and a dependency is found with flow $i'$, a `REMOVE` network update is added for flow $i'$ to $DEPG$. Finally, a `REMOVE` update is added for flow $i$ and an `ADD` network update is inserted into $DEPG$.

## *F. Implementation Details*

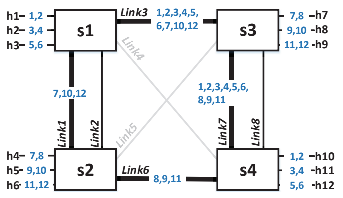
A prototype of the *OptimalFlow* controller was implemented in the `Python` language. *OptimalFlow* integrates part of the POX *OpenFlow* controller's code [18] for emulating an SDN switch. *OptimalFlow* extends three functions in POX's code: `_rx_feature_request()` sends back to the *OpenFlow* controller the status of edge switch ports from an underlying SDN domain; `_flow_mod_add()` calls



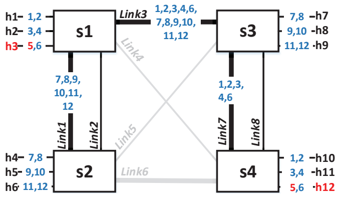
(a) Initial routing of flows.



(b) Link4 down.

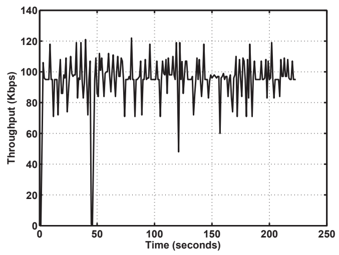


(c) Link4 and link5 down.

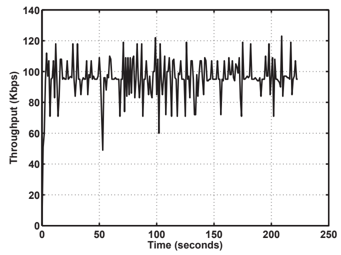


(d) Link4, link5, and link6 down.

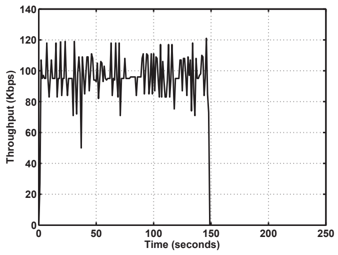
Fig. 4. Network topology and flow routing in the single domain experimental scenario. Switches are connected to the same Floodlight controller and are controlled by one *OptimalFlow* controller (not shown in figure).



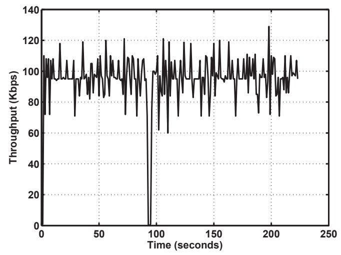
(a) Flow1 (high priority).



(b) Flow3 (low priority).



(c) Flow5 (low priority).



(d) Flow7 (high priority).

Fig. 5. A selection of flows in the single domain experimental scenario. Link4 is disabled at 46s, Link5 is disabled at 92s, and Link6 is disabled at 148s.

*OptimalFlow*'s internal functions to add a new flow; and `_flow_mod_delete()` calls *OptimalFlow*'s internal functions to delete a flow. The *OptimalSolver* module generates an ILP description of the optimization problem and calls the external SCIP (Solving Constraint Integer Programs) solver [19]. Finally, the $OFControllerCommunication$ module uses `pycurl` to communicate with the *Floodlight* [20] controller via its REST API. *OptimalFlow*'s source code is available at http://upm.ro/sereniti/optimalflow.html.

## IV. Experimental Results

In order to assess the performance of *OptimalFlow* in various scenarios and problem sizes, we conduct a series of tests including real and simulated settings. First, we perform a qualitative assessment in a scenario with a single SDN domain, which is followed by a scenario with two SDN domains connected in a hierarchical controller scheme. Then, we perform extensive simulations to evaluate the main features and the solutions of the proposed optimization problem. The qualitative tests are performed on an emulated network topology recreated with the Mininet network emulator [5] on Ubuntu LTS 14.04.3 64-bit OS, and a host with Pentium Dual Core 3.00GHz CPU and 4GB of memory. Simulation experiments are performed with the AIMMS software [6].

### *A. Single Domain Scenario*

In the single domain scenario (see Fig. 4) we use Mininet to create a topology of four SDN switches (*s1, s2, s3, s4*) connected to one *Floodlight* controller. The network topology is monitored and controlled by one *OptimalFlow* controller. We assume that switches are inter-connected by two types of links: 600Kbps (denoted by thicker links in Fig. 4) and 300Kbps (denoted by thinner links Fig. 4). We further assume twelve hosts (denoted by *h1, h2, ..., h12*) and a total of twelve `ARP` and `TCP` flows generated with the `iperf` tool. In Fig. 4 flows are numbered from 1 to 12 and are written on the links on which they are routed. Even numbers denote `ARP` flows, while odd numbers denote `TCP` flows. *OptimalFlow* is configured to route twelve flows, where `TCP` flows have a demand of 100Kbps and `ARP` flows have a demand of 1Kbps. For the sake of simplicity we assume the same security and reliability values on all links. We assume that flows 1, 2, 7 and 8 have a higher priority (200.000) than the others (100.000).

Based on the above-defined setting, *OptimalFlow* configures the network with the routing paths depicted in Fig. 4a. Then, at 46s we disable *Link4* (Fig. 4b). *OptimalFlow* solves the network optimization problem by using the variables associated to flows 1, 4 and 6 and by changing the status of the remaining flow's variables to parameters. Since the `TCP` flow 1 is routed on *Link4*, despite its high priority, flow 1 is briefly interrupted (for approx. 1s), an effect, which is shown in Fig. 5a. In the next phase we disable *Link5* at 92s (see Fig. 4c) and then *Link6* at 148s (see Fig. 4d). As shown in Fig. 5 when *Link5* is disabled flow 7 is successfully re-routed. However, by further disabling *Link6 OptimalFlow* concludes that the network does not have the capacity to route all flows. As a result, a low priority flow, i.e., flow 5, is disconnected from the network (Fig. 5c). It should be noted that in this scenario Algorithm 1 reduced the number of re-routed flows since the *Link5* down event caused only flows 7, 10, and 12 to be re-routed, while the path of the remaining flows was not affected. On the other hand, the adoption of priority-based provisioning of flows disconnected a low priority flow, preserving thus the state of critical communication flows.

### *B. Multi-Domain Scenario*

In the multi-domain scenario we create two SDN domains (*DomainA* and *DomainB*) configured identically as the single-domain scenario from the previous section (see Fig. 6). In

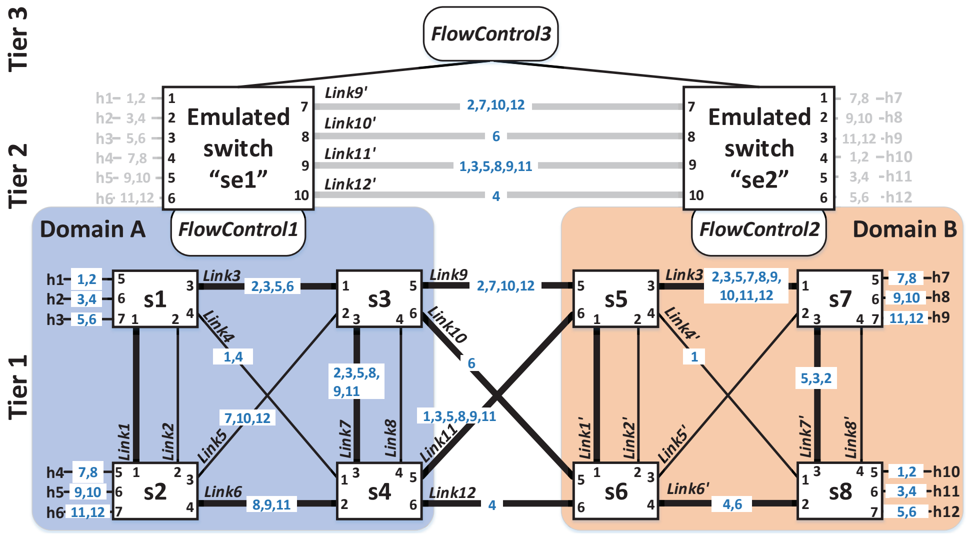
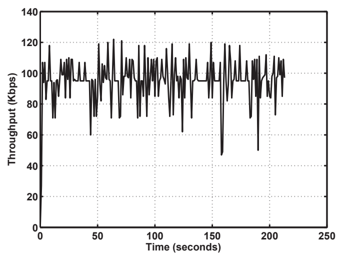
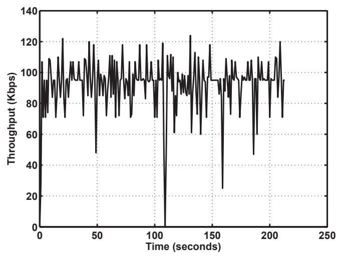
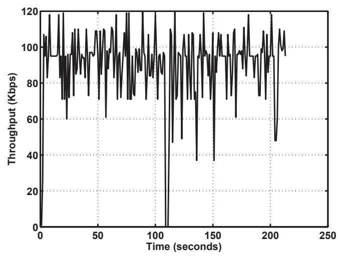


Fig. 6. Network topology and flow routing in the multi-domain experimental scenario.


(a) Flow1 (high priority).


(b) Flow3 (low priority).


(c) Flow5 (low priority).

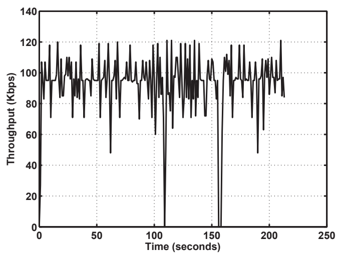
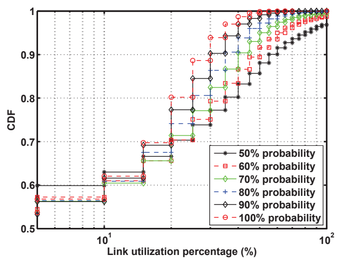

(d) Flow7 (high priority).

Fig. 7. Multi-domain experimental scenario: flow throughput. Link3' is disabled at 108s and Link9 is disabled at 155s.


(a) Distribution of link loads.

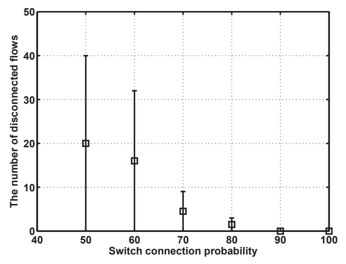
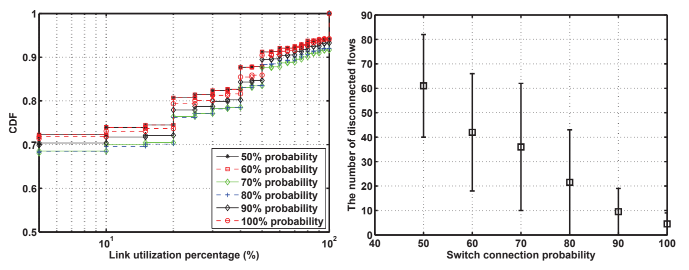

(b) The number of disconnected flows.

Fig. 8. Link load distribution and disconnected flows in the case of equal link capacity (1000Mbps).

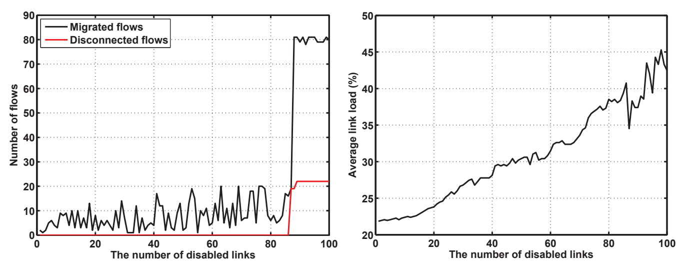
each domain we configure one *Floodlight* controller and one *OptimalFlow* controller. Each *OptimalFlow* controller starts an emulated switch at Tier-2, exposing ten emulated switch (edge) ports to the *FlowControl3* at Tier-3. *FlowControl3* includes one *Floodlight* controller and one *OptimalFlow* controller. The *OptimalFlow* controller's configuration at Tier-3 includes two switches interconnected by four virtual links. The mapping of virtual links at Tier-2 to the real physical links at Tier-1 is described in each *FlowControl*'s configuration file. Each *FlowControl* unit is configured to route the twelve flows as defined in the previous scenario.

First, we start *FlowControl1* and *FlowControl2*. In the initial configuration flows 7, 8, 9, 10 are routed from *DomainA* to *DomainB* via *Link9*, flows 11, 12 via *Link10*, flows 5, 6 via *Link11*, and flows 1, 2, 3, 4 via *Link12*. Next, we start *FlowControl3* to compute a new multi-domain optimal solution. As depicted in Fig. 6, this results in significant changes of routing decisions in the two domains. To start with, flows 2, 7, 12 are migrated to *Link9*, flow 6 is migrated to *Link10*, while flows 1, 3, 8, 9, 11 are migrated to *Link11*. Next, we create two disturbances in the network topology. At 108s we disable *Link3'* in *DomainB*. Since the disturbance does not affect the domain's edge ports, the change is detected by *FlowControl2*, which calculates a new optimal distribution of the affected flows. The disruption is clearly visible in Fig. 7, and more specifically on flow 3 (Fig. 7b), flow 5 (Fig. 7c) and flow 7 (Fig. 7b), which are briefly interrupted. Nevertheless, since flow 1 is routed on *Link4'*, the disturbance does not affect its throughput (see Fig. 7a). Finally, at 155s we disable the inter-domain *Link9*, which briefly interrupts the `TCP` flow 7 and the `ARP` flows 2, 10, and 12. The network change is detected by *FlowControl3*, which solves a new optimization problem and issues inter-domain routing changes. These are received by *FlowControl1* and *FlowControl2*, which also compute a new optimal solution for the affected flows.



(a) Distribution of link loads.  (b) The number of disconnected flows.

Fig. 9. Link load distribution and disconnected flows in the case of uniformly distributed link capacity (200Mbps, 500Mbps, and 1000Mbps).



(a) The number of migrated and disconnected flows.  (b) The average link load.

Fig. 10. Sequential disconnection of links and their impact on the number of migrated flows, disconnected flows and the average link load.

Nevertheless, despite the two-Tier decision-making process, as depicted in Fig. 7, this does not have a significant impact on flows, which successfully recover after only a brief, e.g., 1-2 second, interruption interval.

### C. Quantitative Assessment

By using the AIMMS software we evaluate the solutions of the proposed optimization problem. We assume a large-scale network topology including 50 SDN switches and 100 flows. Switches are structured sequentially in columns of 5; each set of 5 switches is connected to the next set of 5 switches with a certain probability. Flows are routed between the first and the last set of 5 switches. In the first case we assume that $d_i = 50$Mbps, $\forall i \in I$, $|B| = 1$ (one possible link), and $u^b_{jl} = 1000$Mbps. Links and flows are configured with the same security and reliability properties. We test the impact of switch connection probability on the link utilization rate and on the number of disconnected flows. Each configuration is run 50 times and average values are calculated. As shown by results (see Fig. 8a), almost 60% of links are loaded less than 10%. This is a significant aspect in the proposed optimization problem and in the design of a resilient infrastructure where the availability of bandwidth provides the opportunity to migrate flows. Nevertheless, the successful routing of flows also depends on the number of links between switches. As shown in Fig. 8b, with a 50% switch connection probability, on average, we measure 20 disconnected flows (out of 100 flows). However, by increasing the connection probability, at 90% the average disconnected number of flows reduces to 0. Next, we change the scenario and assume a uniform distribution of link capacities of 200Mbps, 500Mbps, and of 1000Mbps. In essence, compared to the first case, we decrease the overall capacity of the communication infrastructure. This effect is visible in Fig. 9a, where in approximately 70% of the cases links are loaded less than 10%. Apparently, in this case only 10% of links exhibit a load higher than 50%, while in the previous case links 15% showed a load higher than 50%. This is explained by the results in Fig. 9b, where the average number of disconnected flows increases to 60 for a 50% switch connection probability, as opposed to the average of 20 in the previous case. Furthermore, despite increasing the connection probability to 100%, there are still flows that cannot be routed due to the decrease in the capacity of paths.

Finally, by using the first configuration as described in this section, we measure the number of migrated flows in the case of randomly disabled links. As shown by the results in Fig. 10a, the number of migrated flows, that is, the number of flows for which a new optimal solution is computed, exhibits a slight increase from 1 to 20 for up to 80 disabled links. This means that *OptimalFlow* only needs to solve the optimization problem for a reduced set of variables, e.g., for the affected flows and for the switches on the paths of access and egress end-points. Nevertheless, after disconnecting 85 links *OptimalFlow* determines that a number of 19 flows cannot be routed, which triggers the execution of the optimization problem on the complete network topology, as described in Algorithm 1. As a result, this increases the number of migrated flows to 80, but keeps the number of disconnected flows at 19, which later increases to 22. By sequentially disabling links we also measure an increase in the average link load. As shown in Fig. 10b the average link load increases up to 40% for 84 disabled links, and decreases down to 35% when the optimization is executed on the complete network topology. Inevitably, by further disabling links, the average link load continues to increase up to 43% for 100 disabled links.

### D. Execution Time

The execution time of *OptimalFlow*'s network reconfiguration procedure depends on the size of the network. This translates to the number of variables in the network optimization problem and the number of flows that need to be installed. We generated several network topologies with a different number of flows ($N_f$), switches ($N_s$), possible links between switches ($N_b$), and switch connection probabilities. We measured the time in which *OptimalFlow* solves the complete optimization problem for each of these settings. As denoted by the results in Table II, for $N_f = 20$, $N_s = 30$, $N_b = 1$ and a connection probability of 50% *OptimalFlow* solves the optimization problem in 0.34s. However, by increasing $N_b$ to 3, the solve time also increases to 1.06s. A connection probability of 100% further increases the solve time to 1.28s. At the other end, for $N_f = 100$, $N_s = 50$, $N_b = 3$ and a connection probability of 100% the solver needs 20.20s to generate a solution. These

TABLE II
OPTIMIZATION PROBLEM SOLVE TIME.

| | | Switch conn. probab=50% | | Switch conn. probab=100% | |
|---|---|---|---|---|---|
| $N_f$ | $N_s$ | $N_b = 1$ | $N_b = 3$ | $N_b = 1$ | $N_b = 3$ |
| 20 | 30 | 0.34s | 1.06s | 0.39s | 1.28s |
| 50 | 30 | 0.95s | 2.85s | 1.16s | 3.59s |
| 100 | 30 | 2.35s | 6.53s | 2.72s | 7.48s |
| 20 | 50 | 0.94s | 2.92s | 1.1s | 3.47s |
| 50 | 50 | 2.67s | 7.79s | 3.00s | 9.69s |
| 100 | 50 | 6.33s | 17.25s | 7.15s | 20.20s |

TABLE III
FLOW INSTALLATION TIME.

| 5 flows | 20 flows | 40 flows | 80 flows | 160 flows | 240 flows |
|---|---|---|---|---|---|
| 16.9ms | 52.9ms | 129.6ms | 176.3ms | 372.9ms | 528.5ms |

high execution times, however, are addressed by *OptimalFlow* in several ways: (i) the full network optimization is only solved at network start-up; (ii) a change in the network topology will reduce the set of variables used in the optimization problem to the affected flows and switches; and (iii) *OptimalFlow*'s hierarchical structure specifically targets large-scale infrastructures, in which case network planning techniques [21], [22] may be used to optimize the placement of *OptimalFlow* controllers.

Finally, we measured the time in which *OptimalFlow* builds the dependency graph and pushes static flows to the *Floodlight* controller. We assumed the single-domain scenario, as presented in the previous sections, and that each flow is configured on four switches. As denoted by the results in Table III for 20 flows *OptimalFlow* runs the flow installation procedure in 52.9ms, which increases to 176.3ms for 80 flows and up to 528.5ms for 240 flows. It should be noted, however, that the execution time is linear and can be further decreased by a careful network planning strategy, as described earlier.

## V. CONCLUSIONS

We developed a novel hierarchical SDN control plane for ICS. The approach builds on the features of an SDN controller named *OptimalFlow* that addresses various requirements of a modern ICS communication infrastructure including scalability, dynamic network redesign as a response to failure or cyber attacks, harmonized routing decisions that encapsulate quality of service, security and reliability properties of communications. The effectiveness of the developed scheme was tested in various experimental and simulation-based scenarios. As shown by results, *OptimalFlow* can be adopted in single and in multi-domain SDN scenarios. To ensure a high performance, however, the parameters of *OptimalFlow* need to be integrated into network planning solutions, which would yield an optimal distribution of *OptimalFlow* controllers.

## ACKNOWLEDGMENT

This research was supported by a Marie Curie FP7 Integration Grant within the 7th European Union Framework Programme (Grant no. PCIG14-GA-2013-631128).

## REFERENCES

[1] M. Hagerott, "Stuxnet and the vital role of critical infrastructure operators and engineers," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 4, pp. 244 – 246, 2014.
[2] B. Genge, F. Graur, and P. Haller, "Experimental assessment of network design approaches for protecting Industrial Control Systems," *International Journal of Critical Infrastructure Protection*, vol. 11, pp. 24–38, 2015.
[3] B. Genge and C. Enachescu, "Shovat: Shodan-based vulnerability assessment tool for Internet-facing services," *Security Comm. Networks*, 2015.
[4] V. M. K.Stouffer, S.Lightman and A.Hahn, "Guide to industrial control systems (ics) security," *NIST Special Publication 800-82, Revision2, National Institute of Standards and Technology*, 2015.
[5] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, (New York, NY, USA), pp. 253–264, ACM, 2012.
[6] AIMMS, "Advanced Interactive Multidimensional Modeling System." http://www.aimms.com/aimms/, 2015. [accessed December 2015].
[7] Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A hybrid hierarchical control plane for flow-based large-scale software-defined networks," *Network and Service Management, IEEE Transactions on*, vol. 12, pp. 117–131, June 2015.
[8] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive resource management and control in software defined networks," *Network and Service Management, IEEE Transactions on*, vol. 12, pp. 18–33, March 2015.
[9] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 539–550, Aug. 2014.
[10] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zupdate: Updating data center networks with zero loss," *SIGCOMM Comput. Commun. Rev.*, vol. 43, pp. 411–422, Aug. 2013.
[11] A. Goodney, S. Kumar, A. Ravi, and Y. Cho, "Efficient pmu networking with software defined networks," in *Smart Grid Communications, 2013 IEEE International Conference on*, pp. 378–383, Oct 2013.
[12] E. Molina, E. Jacob, J. Matias, N. Moreira, and A. Astarloa, "Using software defined networking to manage and control IEC 61850-based systems," *Comput. Electr. Eng.*, vol. 43, pp. 142 – 154, 2015.
[13] N. Dorsch, F. Kurtz, H. Georg, C. Hagerling, and C. Wietfeld, "Software-defined networking for smart grid communications: Applications, challenges and advantages," in *Smart Grid Communications, 2014 IEEE International Conference on*, pp. 422–427, Nov 2014.
[14] IBM and Cisco, "Cisco and IBM provide high-voltage grid operator with increased reliability and manageability of its telecommunication infrastructure," *IBM Case Studies*, 2007.
[15] West Nippon Expressway Company Limited, "Software-defined networking (SDN) solution." http://www.nec.com/en/case/w-nexco/pdf/brochure.pdf, 2015. [Online; accessed December 2015].
[16] C. Meixner, F. Dikbiyik, M. Tornatore, C. Chuah, and B. Mukherjee, "Disaster-resilient virtual-network mapping and adaptation in optical networks," in *Optical Network Design and Modeling (ONDM), 2013 17th International Conference on*, pp. 107–112, April 2013.
[17] C. Chiang, M. Hwang, and Y. Liu, "An alternative formulation for certain fuzzy set-covering problems," *Mathematical and Computer Modelling*, vol. 42, no. 34, pp. 363 – 365, 2005.
[18] "POX openflow controller." http://www.noxrepo.org/pox/about-pox/, 2015. [Online; accessed December 2015].
[19] T. Achterberg, "Scip: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.
[20] "Project Floodlight." http://www.projectfloodlight.org/floodlight/. [Online; accessed December 2015].
[21] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 7–12, ACM, 2012.
[22] B. Genge, P. Haller, and I. Kiss, "Cyber-security-aware network design of industrial control systems," *Systems Journal, IEEE*, vol. PP, no. 99, pp. 1–12, 2015.