Reliable and Flexible Communications for Power Systems:

Fault-tolerant Multicast with SDN/OpenFlow

Thomas Pfeiffenberger and Jia Lei Du
Salzburg Research Forschungsgesellschaft mbH
Jakob Haringer Str. 5/III
5020 Salzburg, Austria
{first.last}@salzburgresearch.at

Pedro Bittencourt Arruda and Alessandro Anzaloni
Instituto Tecnologico de Aeronautica (ITA)
50 Vila das Acacias
12228-900 S. Jose dos Campos, Brazil
pedrobittencourt@yahoo.com.br and anzaloni@ele.ita.br

***Abstract*—Our work evaluates the use of software-defined networking (SDN) for reliable communication. Reliable communication has become an important topic in many areas, including energy communication networks or, more generally, automation control networks. Electrical grids are developing into smart grids, which depend heavily on reliability, robustness and optimized resource usage. On the other side, the separation of communication and network control proposed by SDN opens new possibilities for reliable and flexible networks. In this work, we show how OpenFlow, the leading SDN framework, could be used to solve the problem of robust multicast better than existing technologies used by substations. Our solution uses the fast-failover groups feature of OpenFlow to provide one-link fault tolerance with little packet loss and can provide routes that use resources efficiently and are less likely to fail. Robust shortest path routing and minimum spanning tree broadcast routing come as special cases. We also show how this solution can be extended to handle more link failures (even an arbitrary number of them) or to provide more efficient routes.**

***Index Terms*—Fault tolerant, Reliability, Software-Defined Networking, Multicast, Power Systems, Automation**

I. Introduction

Electrical grids worldwide are developing into so-called smart grids. Smart grids will need to rely heavily on modern information and communication technologies to ensure reliability, robustness and optimized resource usage. One center piece for automation and communication in modern power systems at substation level is the IEC 61850 standard [1]. Two main communication protocols in the standard are GOOSE (Generic Object Oriented Substation Event) and SV (Sampled Values). For both protocols transfer time is strictly regulated by IEC 61850 and extended packet loss can lead to critical system conditions. Additionally, both message types are usually sent as multicast traffic. These issues make efficient and reliable delivery of messages a complex task.

T. Pfeiffenberger; J. Du; P. Arruda; A. Anzaloni, "Reliable and Flexible Communications for Power Systems: Fault-tolerant Multicast with SDN/OpenFlow", 7th IFIP on New Technologies, Mobility and Security (NTMS), 27-29 July 2015, ©2015 IEEE http://ieeexplore.ieee.org/

Software-Defined Networking proposes the separation of control layer and data layer of the communication network and uses one or more central controllers that instruct comparatively simple network devices how to forward packets in the network. As the controller can have a complete and up-to-date view of the entire network, policies can be globally optimized and implemented in form of network applications that run on top of the controller which automatically and centrally deploys suitable forwarding rules in the network devices. This new approach can offer multiple advantages over existing solutions that are based on traditional technologies. Potential advantages include centralized network management and deployment of network configurations, simplified testing, increased robustness, reliability and bandwidth through traffic engineering and fast failover, and finally increased security through traffic isolation and rerouting of traffic flows for inspection purposes. Today, the main standard for SDN is the OpenFlow specification [2].

Advantages related to auto-configuration and security in substation environments using OpenFlow have been studied in [3]. We concentrate on the efficient delivery of multicast traffic and fault tolerance instead, two features that can be implemented using the traffic engineering capabilities of OpenFlow and so-called fast-failover groups. Because efficient multicast is a difficult problem, very few current technologies implement and use it, relying instead on less efficient techniques such as flooding or multiple unicasts. Through the use of central configuration and traffic engineering, software-defined networking simplifies the implementation of efficient multicasting. And to increase robustness, fast-failover groups are used. They are supported since OpenFlow version 1.1 and are ordered lists of actions which are associated with ports. Only the action that is associated with the first active port is executed. In the case that actions consist of simple send commands, an incoming packet would thus be send out once through the first active port. This is a simple, but very powerful local failover mechanism without the need of controller intervention that can be used to build network recovery solutions without reconvergence phases which are typical of spanning-tree based algorithms used in traditional networks. It has been shown [4] that fast-failover groups can allow network recovery within tens of milliseconds, being therefore equal or probably better

in performance than for example the Rapid Spanning Tree Protocol, one of the most commonly used fault tolerance mechanisms nowadays, which can have recovery times ranging up to multiple seconds [5].

In this work we implement a robust multicast forwarding solution for substation environments using OpenFlow. In the following sections we describe i) the related work in the area ii) our approach to implement efficient multicast and fault tolerance in the network iii), our system's building blocks, from network description to final automatized switch configuration, and iv) the simulation results of our system in Mininet [6], a network simulator for OpenFlow networks. Though the implementation described here only handles single link faults, we also indicate how our approach can be expanded to handle multiple link or node faults.

II. Existent Work

In [7], we find a survey of existent algorithms used for fault-tolerance in IEC 61850. Of these, the most well known is the Rapid Spanning Tree Protocol (RSTP), which can achieve sub-second fault tolerance for small networks. Unfortunately, this is not always the case, and the so called convergence phase will last more than a second in more complex topologies, which is unacceptable in many situations in a substation, where high availability is a must. To solve this issue, other algorithms have been proposed. The Media Redundancy Protocol (MRP) is sub-second in the worst case scenario, but can only handle ring topologies (proprietary additions can extend this to multiple rings). The Parallel Redundancy Protocol (PRP) and the High Availability Seamless Redundancy (HSR) achieve, on the other hand provide instant fault-tolerance, but at the cost of flooding the network with redundant packets. Our solution differs of these protocols by providing sub-second recovery, depending basically on the speed of physical layer to identify faults, in *any* topology without resorting to flooding. Besides, it provides efficient multicast route calculation, which is not part of the standard Ethernet technology.

There has already been some interest in fault-tolerant multicast form the OpenFlow/SND community. We cite here the FatTire [8] project as an example of fault-tolerance using OpenFlow's fast-failover groups. They define a syntax for route description and fault-tolerance level description and then propose a compiler to transform these statements onto an OpenFlow configuration. Their compilation involves the calculation of a *forwarding graph*, which is similar to our problem restatement approach for generic fault-tolerance. Unfortunately, the authors do not address the problem of multicast in this paper. Robust multicast has already been treated for OpenFlow 1.0 in [9]. Since version 1.0 does not support fast-failover groups, the proposed approach is completely reactive (i.e. the OpenFlow controller needs to act) and therefore is much slower than a solution that uses fast-failover groups, where fault management is done locally. Recently, a new work using fast-failover groups to solve fault-tolerance appeared [10]. This new approach can additionally deliver QoS to the network, but resorts to optimization algorithms to calculate the network configuration, which have a high computational cost. Our approach, on the other hand, does not guarantees any QoS, but uses only standard graph theory algorithms, which are known to be efficient and scalable. As we will show later, this allows us to build a scheme for unlimited fault-tolerance with OpenFlow (given topology constraints, naturally). As a last note on [10], the authors use OpenState, a non-standard and up to now unimplemented extension to OpenFlow, while we use the standard OpenFlow version 1.3, which already has commercial implementations, even though with shortcomings [4].

III. Approach to Multicasting

The implementation of a fault-tolerant multicast forwarding scheme is a twofold problem: one must first know how to route packets correctly in the network and then have measures in place for the case of failures. We discuss how to realize efficient multicast forwarding using OpenFlow in this section and how to integrate fault tolerance in our solution in the next. The core problem in multicasting is to forward the same information from a publisher to multiple subscribers. Using naive approaches, multicast can be achieved by either simply flooding the packet to all hosts in the network or by cloning the packet at the source and then forward each clone to one subscriber using some unicast technique. While the first solution squanders the bandwidth of the whole network (and possibly leads to security problems, as hosts receive information not addressed to them), the second one saturates links with identical copies of the same packet. The best solution is therefore to clone the packets on the way to the subscribers only when strictly necessary. This ensures that each link will forward the same packet only once and that no host that is not a subscriber receives it. To achieve an even more efficient use of the resources, it can be tried to minimize the number of links used for the multicast transmission. Minimizing the number of links also makes the communication more robust to failures. If failures in the network are identical and independent with equal failure probability p, the probability of a failure in the transmission to any subscriber is $1 - (1 - p)^n$, where n is the number of links through which information flows. Thus minimizing the number of used links also minimizes the probability of failure for any p.

The problem of connecting groups of nodes in a graph using the minimum number of links, or, more generally, using the links that add up to a minimum weight, is known as the Steiner tree problem. Unfortunately, this problem is long known to be NP-hard [11]. However, there are efficient approximations with polynomial runtimes available. A 2-approximation is described in [12]. It generates a solution that uses at most twice the number of minimally necessary links. The approximation is based on calculating the distances between all pairs of nodes in a multicast group (consisting of all subscribers and the publisher) and constructing a complete graph whose nodes are the multicast group members and whose edge weights

are the distances between them, the so-called closure graph of the group. The approximate solution to the Steiner tree is then obtained by calculating the minimum spanning tree in the closure graph and mapping the selected edges into the paths they represent in the original graph. Note that there are other approximations that are faster or improve the quality of the solutions (e.g. [13] or [14]), however the used 2-approximation is simple to implement and already achieves valuable results[1]. The result of a 2-approximation in our test topology is simlare to the Fig. 2.

IV. Approach to fault Tolerance

A. Reactive, Proactive and Hybrid Approaches

Fault tolerance in OpenFlow can be achieved in two ways: reactively or proactively. In a reactive fault tolerance scheme, the OpenFlow controller is responsible for recalculating the configuration rules only when a fault occurs. The main advantage of this scheme is that it is possible to achieve strong fault tolerance because the solution can continuously reconfigure the network in the case of multiple faults. However, this technique can also be too slow for real-time applications, as it requires the switches to transmit an OpenFlow message warning the controller of the fault and the controller to recalculate the forwarding rules and send them to the switches. These are time consuming operations and in the meantime the switches could be dropping or delaying packets. The strategy adopted in [9] is an example of a (mainly) reactive fault tolerance.

In a proactive fault tolerance scheme, the controller preemptively installs all rules and groups necessary for managing a fault. This is possible due to the fast-failover groups introduced in version 1.1 which allow an OpenFlow switch to change its forwarding behavior by itself when a port goes down. Since the detection of a port down event is very simple and performed directly in the switch, this approach has very low reaction times. Its main disadvantage is that it achieves fault tolerance against multiple faults only at the price of higher resource usage in the switches as all possible relevant failure scenarios need to be considered and all possible reactions pre-installed, most of which will never be executed. Thus, it is difficult to scale this approach to big networks while maintaining high levels of fault tolerance. One of the first works to take advantage of fast-failover groups is FatTire [8].

A hybrid approach to fault tolerance would be the best in terms of robustness and rational use of switch resources. We propose making the network proactively tolerant to one fault, so that failover times are very low and there is very little packet loss on disconnection, but we also propose that the network be able to reconfigure itself to the new topology after the failure. This is similar to the approach described in [9] , however we take advantage of local fault recovery which does not require communication with the controller during failover. With such a hybrid approach, the network is not only tolerant to a fault with extremely low reaction times, but it is also able to restore bandwidth efficiency and fault tolerance after a fault. Of course, the algorithm controlling the network must run fast enough to avoid that a second failure happens before the network is reconfigured. If this situation is possible, it would be recommended to make the network proactively tolerant towards up to two faults, something that can be achieved with small extensions to the described solutions (basically by reapplying the algorithms, see below), though at a greater cost in switch resources. Note that we have so far only implemented the proactive part in this work, leaving the reactive part for future work.

[1]It bounds probability of failure for a route to the double of the optimal. Besides, the 2–approximation is exact for unicast and broadcast. Also , it is very good for a small number of nodes (see [12]).

B. Fault-Tolerant OpenFlow Group Structures

Our approach to proactive fault-tolerance is based on restating the multicast problem for each possible link failure, a method that the authors of [8] applied in a similar way to unicast routing. We start with the initial problem of creating a multicast tree from one publisher to a group of subscribers, given a certain topology (see Sec. III). The solution to the problem is a set of links in the network that will be used to carry traffic. However, if a link in this set fails, the solution to the problem is not valid anymore. At the point of failure the original problem is *restated* and a new multicast tree is built from the switch affected by the failure (the new publisher) to the affected subscribers (new subscribers group) without using the affected link (the new topology). If this problem has been solved for each link in the original multicast tree, a one-link fault-tolerant multicast route has been created. Note that the problem restatement can be repeated recursively until either the desired fault tolerance level is achieved or the routing problems become unsolvable as it exhausts the backup possibilities. At this point, the maximum fault-tolerance for that topology has been reached. This approach to fault tolerance can be summarized in a tree, where each node represents a problem statement and each edge represents a possible one-link failure. However, an important optimization can be made, noting that that different cuts of links can actually result in the same problem statement. This allows the reuse of already encountered solution and reduces the problem restatement tree to a problem restatement DAG.

In an actual implementation, however, one must still translate this graph into an actual switch configuration. This is composed by a flow entry to identify the packet traffic type and an associated *OpenFlow group structure*. This group structure is a DAG of all groups and fast failover groups that correspond to the decisional process implied by the problem restatement DAG. Figure 1 shows a complex example of such group structure in the switch's group table.

From a concrete implementation point of view packets cannot be forwarded just based on source address, destination address and switch ingress port number as this would lead to ambiguities. Two packets can have the same source, the

same destination and the same ingress port and yet be on different multicast routes (e.g. original route and backup route). Thus our approach requires the packets to have some kind of memory to identify the exact tree in which they are being forwarded. Currently, we use VLAN tags to carry this information. At the ingress point of the OpenFlow network an initial VLAN tag is assigned to incoming packets. In case of a of a fault, a new VLAN tag needs to be assigned to the packet to ensure that it will be routed along a different multicast tree. For this reason VLAN tags that are already contained in the packet before they enter the OpenFlow network need to be removed at the OpenFlow network ingress point and restored at the egress point to make the process transparent to devices outside of the OpenFlow network.

V. Verification

To demonstrate our solution, we chose two topology test-cases: one topology that closely resembles a real substation [7], but provides more redundancy (see Fig. 2) and a fat-tree topology, which is a standard testcase for fault-tolerance (see Fig. 3). The first topology consists of multiple rings attached to an internal backbone ring. Each outer ring represents a bay in the substation with IEDs (hosts) connected to switches. For the sake of simplicity, only one host was attached to each switch. In practice either many IEDs are attached to a single switch or the IEDs themselves are connected in a ring, eliminating the need of switches inside the bay area. This is possible as IEDs can have more than one network interface, allowing them to perform switching. In future even OpenFlow-enabled IEDs are conceivable. For our tests we set up a sample multicast group, send out multicast messages to be routed in the test network and check if all subscribers successfully receive the messages sent by the publisher. One sample multicast group is highlighted in Fig. 2. This group consists of a larger number of hosts that span the entire network thus involving both bay and inter-bay traffic. We then disconnect selected links in the network that are part of the constructed multicast tree and verify that messages are still successfully delivered. In Fig. 2 the alternative multicast route used by our solution to deliver packets after a link fault is shown.

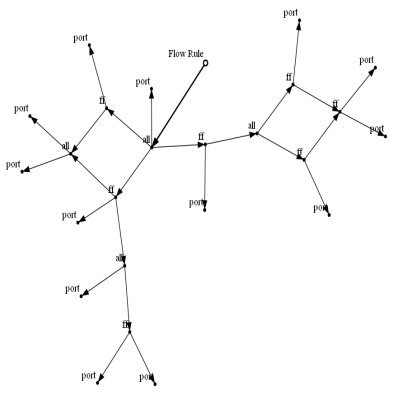

Fig. 1. Example for an OpenFlow group structure that sends out packets (port), clones packets (all) and activates alternative routes for packets (ff) and is installed in switches by our solution.

TABLE I. Relevant figures for network resource consumption for the multicast routing problem in Fig. 2. From left to right, number of elements in the group, number of VLAN tags used, number of groups and number of flows used in average, maximum number of groups and flows in a switch and lastly number of links used for the main (faultless) solution.

n	VLANs	$\overline{n_{grp}}$	$\overline{n_{flw}}$	max n_{grp}	max n_{flw}	links
2	2	0,03	0,17	1	2	3
3	8	0,22	1,14	3	7	10
4	14	0,58	2,97	3	13	17
5	20	0,94	5,53	3	19	24
6	24	1,11	5,72	5	13	29
7	27	1,25	6,44	6	13	33

We also measured figures relating to consumed resources in the network, namely, number of flows and groups used per switch (average and maximum) as well as number of VLAN tags used and number of links used for the main (faultless) route. Besides, we did so for a varying number of members in the multicast group. Instead of using only the full group for measurement, we added the members one by one in anti-clockwise manner, starting by $h12$ and repeated the testes for each case. With this, we show qualitatively how our solution drains resources for different number of groups members. The results for these measurements are shown in table I. We also performed the same testing on the fat-tree topology adding hosts in numerical order until the group spanned all of them. The results for these measurements are shown in table II. Analyzing these two tables, we perceive that our approach is perfectly scalable for bigger networks and do not consume much resources, except for VLAN tags. This bottleneck can be solved by using MPLS tags instead of VLAN tags, which have a bigger range and are also part of the OpenFlow specification. Unfortunately, real implementations, such as the HP2029 switch, tend to not implement support to them.

There are two reasons why we do not include any measurements of time efficiency of our solution. First, we could not run our configuration in a real network, for we lacked the resources for that. Mininet [6] uses the loopback interface and virtualized switches for simulation and therefore may not accurately model layer 1 physical link fault detection, which does contribute to packet loss, which is small but non-negligible in our scenario. Second, our tests ran static configurations, and therefore depend solely on the OpenFlow implementation's efficiency in identifying a link-down. This subject has already been treated elsewhere. For example, [4] analyses packet loss in a simple scenario involving fast-failover groups.

VI. Future work

Our work is only one part of a more comprehensive solution. Ideally parameters like latency and bandwidth must also be considered when constructing fault-tolerant multicast trees, for GOOSE and SV messages must arrive within strict time intervals, and SV often has high bandwidth requirements. So far

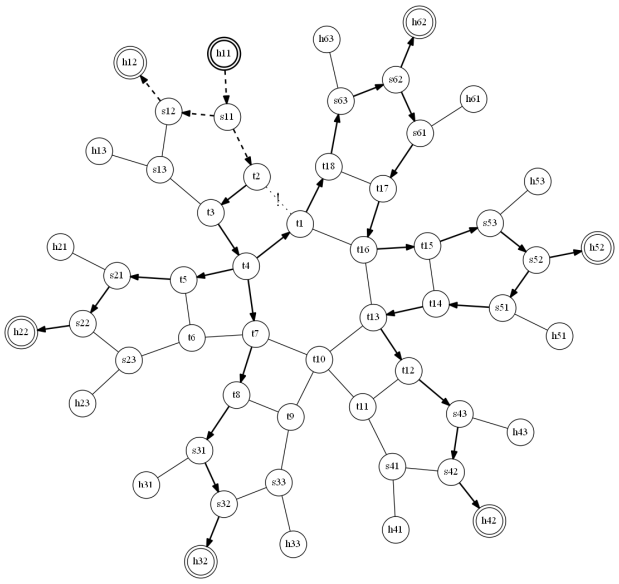

Fig. 2. Shows the network behavior when link t2-t1 fails. The multicast message starts from h11 on the main tree (dashed bold edges). When it hits the faulty link, the switch forwards the packet to a different tree (bold edges) which delivers message to all its destinations [**?**].

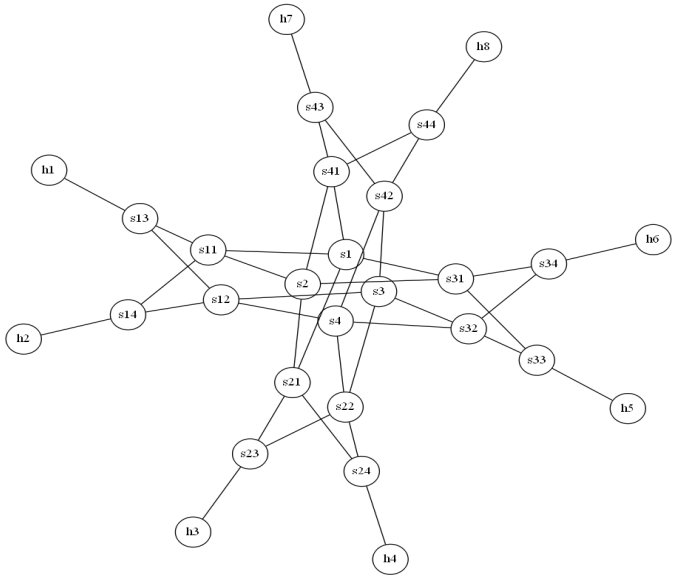

Fig. 3. Shows two-level eight-host fat-tree topology used as the second testcase. The hosts are added to the group anti-clockwise, starting from $h1$.

our code relies on the minimization of edges to limit latency and offers no mechanism to prevent links from congesting. (Note that this is not the case when using traditional technologies either.) Fortunately, only moderate changes are necessary to take these two factors into account. The authors of [15] show how to generate approximations for the delay constrained Steiner tree problem. It can be difficult to determine link delay at local level for the approximation, but a simplified, conservative model can be used to make routing decisions. Determination of link delay for the approximation becomes even more difficult when taking network dynamics into account and the dependency of delay on network load. But as a simple though possibly suboptimal solution the delay model could also at the same time be used to avoid link congestion in the network. By assigning a higher delay rating to crowded links, the delay constrained version of the approximation will naturally avoid those links. If the link capacity has been fully used up, the delay is set to infinity to prevent the algorithm from further using the link. Another area of improvement is adding reactive behavior to the solution as outlined in Sec.IV. It allows greater fault tolerance at no additional resource usage in the switches. One challenge with network device reconfigurations during operation is to ensure consistency of updates, i.e. updating switch configurations in such a way that switches do not have contradictory configurations and that no packets are lost at any time. A solution for this issue has been developed in [8]. Because we use VLAN tags to identify our routes, we can use the simpler per-packet consistency model proposed in that article to make safe configuration transitions. Finally, we have restricted ourselves exclusively to link failures so far and switch failures need to be investigated in future. Implementing switch fault tolerance using our solution is not difficult as a switch failure can be modelled as multiple link failures. If a multicast tree is build that aims at avoiding links that are connected to the failed switch, we obtain a solution that is tolerant both to link and switch failures. However, this simple approach may yield suboptimal solutions in some cases as avoiding a switch entirely in the case of a link failure introduces unnecessary constraints for the solution.

TABLE II. Relevant figures for network resource consumption for the multicast routing problem in Fig. 3. From left to right, number of elements in the group, number of VLAN tags used, number of groups and number of flows used in average, maximum number of groups and flows in a switch and lastly number of links used for the main (faultless) solution.

n	VLANs	$\overline{n_{grp}}$	$\overline{n_{flw}}$	max n_{grp}	max n_{flw}	links
2	3	0,10	0,40	1	3	4
3	7	0,35	1,30	3	5	9
4	9	0,65	1,85	3	7	12
5	9	0,80	2,25	4	6	13
6	10	1,05	2,70	4	6	15
7	14	1,40	4,05	5	9	20
8	16	1,80	4,85	5	11	23

VII. Conclusion

Our work shows that software-defined networking based on OpenFlow can be used to build efficient solutions to handle fault-tolerant multicast in substation environments. One open question that remains is how suitable current OpenFlow implementations are for use in productive networks. To date, many OpenFlow switches are implemented in software only, like the CPqD software switch [16], and are often only used for prototyping purposes, or the OpenFlow specification is only partially implemented in hardware, like in the case of HP 2920 switches, and the devices are not sufficiently mature yet [4]. However, based on our results we believe when first full hardware implementations of OpenFlow switches suitable for productive use become available, software-defined networking

will be a strong competitor against current communication technologies in substation environments and beyond.

To the best of our knowledge, the only paper to date using a similar approach is [17]. In difference to their solution, we do not explicitly reuse paths in our solution as this might increase the probability of encountering additional link-failures in an affected area and we propose a method for a hybrid approach to recovery from link-failures based on a mixture of proactive and reactive behavior to ensure both low recovery times from faults and continuity of fault tolerance over time.

ACKNOWLEDGMENT

The work described in this paper was part of the project Open Flow Secure Grid (OFSE-Grid) and SDN OpenFlow-based communication system for multi-energy domains (OPOSSUM) which was funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT).

REFERENCES

[1] *IEC 61850: Communication networks and systems in substations*, International Electrotechnical Commission (IEC) Std., 2013.

[2] *Open Flow Switch Specification 1.4.0*, Open Networking Foundation Std., June 2013.

[3] A. Cahn, J. Hoyos, M. Hulse, and et al., "Software-defined energy communication networks: From substation automation to future smart grids," in *Smart Grid Communications (SmartGridComm), 2013 IEEE Int. Conf. on*, 2013, pp. 558–563.

[4] T. Pfeiffenberger and J. L. Du, "Evaluation of software-defined networking for power systems." Proceedings of the 2014 IEEE Conference on Intelligent Energy and Power System, 2014.

[5] A. Myers, T. E. Ng, and H. Zhang, "Rethinking the service model: Scaling ethernet to a million nodes," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Hot Topics in Networking*, 2004.

[6] B. Lantz, B. Heller, and N. Mckeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *In ACM SIGCOMM HotNets Workshop*, 2010.

[7] Hirschmann, "Data communication in substation automation system (sas): Substation network topology," White Paper WP 1004HE – Part 3, Tech. Rep., 2012.

[8] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "Fattire: declarative fault tolerance for software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 109–114.

[9] D. Kotani, K. Suzuki, and H. Shimonishi, "A design and implementation of openflow controller handling ip multicast with fast tree switching," in *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on*. IEEE, 2012, pp. 60–67.

[10] A. Capone, C. Cascone, A. Nguyen, and B. Sans, "Detour planning for fast and reliable failure recovery in sdn with openstate," 2014, pp. 1–8.

[11] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.

[12] H. Takahashi and A. Matsuyama, "An approximate solution for the steiner problem in graphs," *Math. Japonica*, vol. 24, no. 6, pp. 573–577, 1980.

[13] K. Mehlhorn, "A faster approximation algorithm for the steiner problem in graphs," *Information Processing Letters*, vol. 27, no. 3, pp. 125–128, 1988.

[14] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità, "An improved lp-based approximation for steiner tree," in *Proceedings of the 42nd ACM symposium on Theory of computing*. ACM, 2010, pp. 583–592.

[15] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 286–292, Jun. 1993. [Online]. Available: http://dx.doi.org/10.1109/90.234851

[16] (2014, June) Open flow 1.3 switch specification. Open Networking Foundation. Accessed: 2014-11-24. [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[17] B. N. Gyllstrom, D. and J. Kurose, "Recovery from link failures in a smart grid communication network using openflow," in *Proc. of the IEEE International Conference on Smart Grid Communications*, ser. Smart Grid Com '14, 2014.