Performance Comparison of Node-Redundant Multicast-Distribution Trees in SDN-Based Networks

Miroslav Popovic
EPFL, IC-LCA2
Lausanne, Switzerland
miroslav.popovic@epfl.ch

Ramin Khalili
Huawei
Munich, Germany
ramin.khalili@huawei.com

Jean-Yves Le Boudec
EPFL, IC-LCA2
Lausanne, Switzerland
jean-yves.leboudec@epfl.ch

***Abstract*—Some of the industrial processes with hard real-time constraints, such as those commonly found in the context of smart grids, require very reliable packet delivery and multicast. Solutions such as PRP [1] and iPRP [2] have been traditionally deployed over redundant (and often dedicated) networks. In this paper, we study how to construct node-redundant multicast trees that could be used *in parallel* over a *shared network infrastructure*. Our solution can be deployed over shared public networks, e.g., mobile carrier networks, hence it has an operating cost lower than traditional deployments of PRP and iPRP.**

We evaluate different algorithms proposed in the literature for providing node-redundant multicast-distribution-trees. We study specifically how to adapt these algorithms in an SDN network and compare them based on the number of forwarding rules that need to be installed on SDN switches, the number of hops between source-destination pairs given the installed forwarding rules, and the number of sources that can be placed in the network given the capacity constraints. In addition, we discuss the effects of topology changes (node failures, new source arrival and new destination arrival) on the activity of the SDN control plane.

I. Introduction

Some of the industrial processes with hard real-time constraints require the creation of redundant multicast-distribution trees to support 0-ms packet repair-time. For example, in the emerging smart-grids there are applications such as grid protection or grid control that are considered critical. We were faced with such requirements when we designed the communication network as part of the EPFL smart-grid project (smartgrid.epfl.ch).

Solutions such as PRP [1] and iPRP [2] have been deployed in these settings; they achieve 0-ms packet repair-time by duplicating packets over node-redundant multicast-distribution trees. This is commonly done in industry by having dedicated and duplicated networks. In this paper, we study how to construct node-redundant multicast trees that can be used *in parallel* over a *shared network infrastructure*. This is of interest to telecom operators and manufacturers who want to propose solutions over their shared infrastructures [3].

Our work differs from [4] and [5], as the solutions in [4] and [5] establish one multicast tree as a backup tree that is activated only when a link or node failure is detected in the primary tree. This require extra repair-time delay as the detection of a failure is a time consuming process. Our solution however relies on the parallel use of node-redundant trees, hence it provides a 0-ms packet repair-time.

We focus our study on SDN-based networks. SDN technology is the most likely candidate for the design of 5G systems [6], [7]; as it provides a fully programmable operator-network interface. An SDN controller has a global information of the situation in the network and can enforce appropriate forwarding rules to the SDN switches. This should therefore facilitate the implementation of the proposed algorithms. Nevertheless, there is a question of the choice of the appropriate algorithm to be implemented for the purpose of creating node-redundant multicast-distribution trees.

This problem is proven to be NP-complete [8] and approximation methods for solving it are the only possible. To that end, we evaluate the performance of three algorithms that we adapted to the smart-grid setting: (i) `ReducedCostV` [9] computes a pair of node-redundant spanning trees for a source, (ii) `MADSWIP` [10] computes a pair of maximally disjoint paths between a source and each of its destinations, and (iii) `Takahashi - Matsuyama` [8] finds a pair of minimum-cost Steiner trees between a source and all its destinations. Adaptations of the first two algorithms provide us with a pair of node-redundant multicast trees, whereas the third one constructs a pair of node-disjoint multicast trees.

We analyze the following metrics: network-utilization efficiency (number of sources placed, minimum/maximum number of hops to the destination) and the number of forwarding rules that need to be installed at SDN switches, with and without aggregation. We quantify performance of different methods based on these metrics when applied on networks with structured (operator) topology or on random networks. Our results show the following:

- In terms of the numbers of sources placed, for general networks, `ReducedCostV`[1] and `MADSWIP` perform the best as in some cases, especially for random

[1]In this paper we call "`ReducedCostV`" the adaptation of `ReducedCostV` and similarly with `MADSWIP` and `Takahashi-Matsuyama`.

topologies with an increasing number of destinations, `Takahashi-Matsuyama` is unable to place the second tree for any sources, hence it is not a viable solution. Nevertheless, for structured, dual-plane-like networks, `Takahashi-Matsuyama` gives results comparable to the two other algorithms.
- In terms of the number of hops to the destinations, `MADSWIP` and `Takahashi-Matsuyama` (if source placement is possible) outperform `ReducedCostV` as they construct paths that are 2 to 3 times shorter than those established by `ReducedCostV`. They can therefore provide a much better delay guarantee.
- Using `MADSWIP` and `Takahashi-Matsuyama` (if source placement is possible), the total number of rules that need to be installed in the network is 5 to 6 times smaller than `ReducedCostV` when no flow-rule aggregation is applied. If applied, the number of installed rules is reduced to handful of rules for all the algorithms.
- In terms of the SDN-control-plane activity in the case of change in network topology (node failure or new source), `MADSWIP` and `Takahashi-Matsuyama` require fewer changes compared to `ReducedCostV`. If the change of interest is an arrival or departure of a destination within a multicast group, `ReducedCostV` requires no changes, hence it outperforms the two other algorithms in this case.

The rest of the paper is structured as follows. In the next section, we cover the related work. In Section III, we provide the problem formulation. In Section IV, we introduce three different types of algorithms that we study in this paper, to construct node-redundant multicast trees. In Section V, we analyze their performance and in Section VI we discuss the effects of topology changes on the SDN control-plane activity. In Section VII, we provide the conclusion.

II. State of the Art

An extensive overview of the algorithms for network survivability was done by Kuipers in [11]; it served as an excellent reference in classifying methods for the creation of multicast-distribution trees.

We distinguish three different families of algorithms that are of our interest and we choose for this evaluation one representative from each of the families. First, there are algorithms that find *spanning trees*. In the literature, we find many solutions that are based on placing one tree and then removing links and nodes already used, before placing the second trees. Such solutions are suboptimal and can lead to the inability to place the second tree. A better solution is presented by Médard et al. in [12] and we choose its extension (algorithm `ReducedCostV`) by Zhang et al. [9] as the representative of the algorithms that find node-redundant spanning trees.

The second family of algorithms find *Steiner trees*. As for the spanning trees, there are many solutions based on placing one Steiner tree and then removing links and nodes already used, before placing the second one. Surprisingly, to the best of our knowledge, there are no practical algorithms for concurrent tree-construction. We leave this problem for future work and for this comparison we use the method of sequential placing of Steiner trees constructed with the algorithm from [8]. We refer to this algorithm as `Takahashi - Matsuyama`.

Finally, we have algorithms that find *disjoint paths*, where many algorithms are inspired by the work of Suurballe and Tarjan [13]. We single out the work of Nina Taft-Plotkin et al. [10] (MADSWIP algorithm) and Guo et al. [14] (DIMCRA algorithm). MADSWIP computes maximum-bandwidth maximally disjoint paths and minimizes the total weight as the secondary objective, whereas DIMCRA finds two link-disjoint paths subject to multiple quality-of-service constraints. In our smart-grid setting, metrics of interest are only bandwidth and delay, hence, for the comparison in this paper, we choose MADSWIP as the representative of the algorithms that find disjoint paths.

The application of SDN technology for reliable smart-grid communication networks is considered in [4] and [5]. Neverthless, despite insisting on the need for reliable packet delivery, the described settings are not compatible with a reliability brought by iPRP-like $0ms$ repair-time. In [4], the authors describe **`Pcount`**, an algorithm that uses OpenFlow to detect link failures that should trigger the activation of backup multicast-distribution trees. However, the time-scale in which **`Pcount`** operates is in the order of seconds, which is unacceptable for critical processes in smart grids. Similarly, in [5], the approach of bypassing the link that has failed *after* the failure occurs with the precomputed backup segment is not acceptable for critical traffic.

The parallel redundancy protocol (PRP) [1] requires redundant, and, moreover, cloned networks; hence it cannot be used in a shared infrastructure. TCP-like solutions are acceptable for applications with less strict delay-requirements, as it takes several round-trip times for losses to be repaired. Network coding [15] is commonly used to improve networks throughput and reliability but can perform poorly in terms of latency. Source coding (e.g., Fountain codes [16]) is suitable in the case of bursty packet transmissions where encoding and decoding are performed across several packets. However, in smart grids, usually there is a single packet per time-slot and it should be sent as soon as it is available to avoid unnecessary delays.

III. Problem formulation

Let $(\mathcal{V}, \mathcal{E})$ be a *directed* graph representing our network, with the set of vertices $\mathcal{V}$ and the set of edges $\mathcal{E}$. Let $B(i,j)$ be the bandwidth assigned to $(i,j) \in \mathcal{E}$ and $C(i,j)$ be the cost of using the link. We denote by $M(s) = \{s, D(s)\}$ a multicast session with source $s \in \mathcal{V}$ and destinations $D(s) \subseteq \mathcal{V} - s$. We study algorithms to construct a pair of node-redundant multicast trees for this session such that a failure of a node (vertex) in the network leaves each destination in $D(s)$ still connected to the source by using at least one of the trees.

These node-redundant trees are not necessarily disjoint and can contain some nodes or edges in common. Hence, our study is different from those that propose the use of disjoint spanning trees; those studies are based on placing one tree and then removing links and nodes already used, before placing

the second tree, because they are suboptimal and can lead to the inability to place the second tree [12]. For an example of one such case, we refer to Figure 1. Node 1 is the source of packets and there are two destinations of interest: nodes 5 and 7. The blue tree follows the path 1-3-5-6-7, whereas the red tree follows the path 1-4-7-6-5. We observe that the two trees share node 6 thus making them non-disjoint. Nevertheless, both destinations can be reached upon a failure of a single node (node-redundancy property), which is the goal we want to accomplish. Note that in this example it is not even possible to construct two trees that are disjoint.

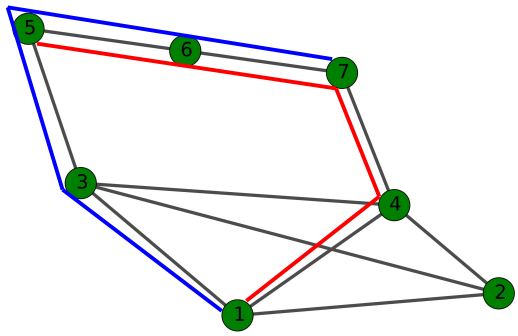

Fig. 1. Source: Node 1. Destinations of interest: 5 and 7. Blue and red trees are node-redundant but they are not disjoint (node 6 is shared).

For comparison, we define the following metrics of interest:

- Number of sources placed. This metric illustrates how many sources can be served, depending on the approach taken. We say that a source is placed only if it is possible to establish node-redundant trees that comprise all the destinations in the group of receivers, subject to the capacity constraints.
- Minimum number of hops between each source and each destination. Given that in all cases we have two paths between each source and each destination, this metric illustrates the effective number of hops (delay) in the case when both paths are operational.
- Maximum number of hops between each source and each destination. Given that in all cases we have two paths between each source and each destination, this metric illustrates the effective number of hops (delay) in the worst case if one path becomes unavailable.
- Number of flow rules installed per source placed. This metric illustrates how big the SDN flow tables are. It is desirable to have a smaller number of flow-table entries as the switches can be of limited resources.
- Number of flow rules installed per source placed. This metric illustrates how many rules need to be installed at the switches by the controller and how large the SDN flow tables are. The larger the number of rules installed is, the higher the volume of control traffic exchanged is between the controller and switches. This can, on one hand, overload the controller and, on the other hand, saturate the control plane. A larger number of installed rules also means larger flow tables, which can affect the performance of SDN switches with limited resources.

IV. Methods Used in this Comparison and Necessary Adaptations

As mentioned in Section II, we choose `ReducedCostV`, `Takahashi - Matsuyama` and `MADSWIP` as the basis for the methods based on spanning trees, Steiner trees, and disjoint paths, respectively. In order to compare these approaches, it was necessary to harmonize the assumptions and to adapt the algorithms so that they all produce multicast-distribution trees. We discuss below the necessary adaptations, after describing the algorithms themselves.

A. `ReducedCostV`

This algorithm finds a pair of directed node-redundant spanning trees (referring to as T^R and T^B trees) for a given source in the network. As stated by Xue et al. [17], the construction of these trees is related to ear decomposition of the graph. Adapting this idea, `ReducedCostV` constructs T^R and T^B from a DFS (depth first search) tree $\mathcal{T}$ by applying the ear decomposition technique. This is done by adding an ear whenever a back edge from $\mathcal{T}$ to the current T^R and T^B, which span a subset of the network nodes, is encountered. The resulting T^R and T^B trees have a total cost that is minimum among all possible node-redundant spanning trees.

As mentioned in Section III, we analyze directed graphs. `ReducedCostV` takes as input undirected graphs, whereas it outputs directed graphs. This means that, after placing the first source and updating the available link capacities, the resulting graph has available link capacities that are asymmetric. Hence, it cannot be treated as undirected, which is the problem for the next iteration of the simulation when the next source needs to be placed. For a solution, for every link, we keep track of the available capacities in both directions; and for the input for the algorithm, we take the minimum of the two.

Furthermore, for each source that offers traffic `ReducedCostV` computes a pair of spanning trees that already have a property of having no single point of failure. By construction, all the destinations from a group of receivers that belong to the same multicast group will be reachable, hence no further adaptation is required. We sort the sources based on the offered traffic and we treat them in that order. Before placing a source, we remove the links that do not have enough capacity to support the offered traffic.

B. `Takahashi - Matsuyama`

[8] proposes a heuristic to compute minimum-cost Steiner tree for a given source node and a set of destinations in a network. This heuristic performs as follows. It first finds the closest destination to the source and constructs the path between the source and the destination. It then selects the closest non-spanned destination to this tree and updates the tree by adding the path between this destination and the tree. It repeats this process until all the destination nodes are covered. The resulting Steiner tree has the minimum total cost.

`Takahashi - Matsuyama` also expects the input in the form of undirected graphs, whereas it outputs directed graphs. We deal with this aspect in the same way as in the case of the method based on spanning trees (see the description of `ReducedCostV` approach).

An additional adaptation is needed as `Takahashi – Matsuyama` provides a single Steiner tree (not a pair). Hence, we use the method (e.g., described in [18]) where after placing the first Steiner tree, we remove the already used nodes and links. After this we run again the `Takahashi – Matsuyama` algorithm for the creation of the second Steiner tree, that is now node-redundant by construction. We sort the sources based on the offered traffic and that is the order in which we treat them. Before placing a source, we remove the links without enough capacity to support the offered traffic.

C. *MADSWIP*

This algorithm computes a pair of maximally disjoint paths between the source and its destinations such that either the total cost of paths is minimized or the bandwidth is maximized. It performs as follows. First, a shortest-path tree from the source to the destination is computed.The edge costs and bandwidths are updated, based on the computed shortest path, and the nodes are labeled. Using this labelling information, a pair of disjoint paths from the source to all destinations are computed.

In the case of `MADSWIP`, both input and output graphs are directed, hence, in this respect, no changes are needed. Still, there is another adaptation that is required. Our goal is to find node-redundant multicast trees, whereas `MADSWIP` produces link-disjoint paths. Therefore, we needed firstly to adapt the `MADSWIP` to produce node-disjoint paths (instead of link-disjoint) before combining them into node-redundant multicast trees. The method that we used is described in [11]; we split each node u into two nodes u_1 and u_2, with a directed link (u_1, u_2), and the incoming links of u connected to u_1 and the outgoing links of u departing from u_2.

Once we have node-disjoint paths, there is a question of how to combine them into node-redundant trees. Each source has a number of destinations that correspond to the same multicast group. Each execution of [10] produces a pair of paths originating from the source, one pair per destination. This raises the question of how to combine disjoint paths into trees: For a single path-pair, which path should join which of the two output trees? We use the following heuristic to solve this problem. Again, we sort the sources based on the offered traffic and that is the order in which we treat them. For each source, we pick randomly the first destination (from the group of multicast receivers) and we compute disjoint paths. The two solution trees (named blue and red) are initialized with these paths. After every subsequent computation of disjoint paths (for the other destinations from the same multicast group), we decide which path should be combined with which tree, based on the minimal number of *new* links that need to be added. We also remove the links without enough capacity to support the offered traffic for the next computation.

Furthermore, to have a fair comparison, we accept paths from `MADSWIP` only if they are *completely* disjoint (`MADSWIP` finds *maximally* disjoint paths). For the same reason, we implemented a version of `MADSWIP` that computes minimum-cost disjoint paths (instead of maximum bandwidth).

V. Performance Evaluation

In this section, we provide performance analysis. We first describe our simulation setting.

A. Network scenarios

Random (ad-hoc) topology: We generate a topology that resembles the one of wireless sensor networks (see Figure 2). We place 100 switches in a $1000m \times 1000m$ area as follows: The positions are generated uniformly at random, with restrictions that no switches are within $75m$, and the connectivity between switches is ensured if the distance between them is below $150m$. All the links have capacity of 1.

We randomly select a subset of nodes that serve as destinations. Depending on the number of destinations, we distinguish scenarios that correspond to *sparse* multicast (3, 4, ... , 10 receivers) and *dense* multicast (15, 20, 25, ... , 40 receivers). In the sparse multicast case, we select destinations in such a way that the distance between any pair of destinations is at least $100m$. The idea is that several receivers are required for reliability reasons and thus they should not be too close to each other.All the nodes that are not designated as receivers are the candidate sources with the traffic rate uniformly distributed in the interval $[0.025, 0.05]$. Our algorithms try to compute node-redundant trees for as many of them as possible.

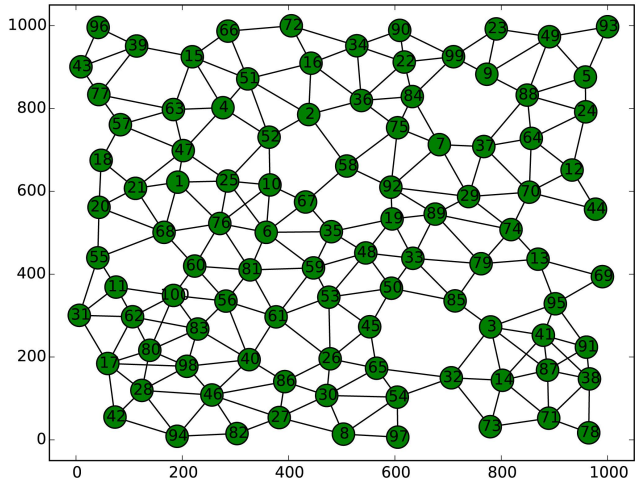

Fig. 2. Random (ad-hoc) topology.

Structured (operator) topology: We also study the performance of mechanisms proposed in the previous section on a topology generated from [19], [20]; it represents the backhaul of carrier networks. It is hierarchical and consists of three layers: access, aggregation, and core.

The access layer consists of clusters of 20 access switches, each connected to two neighboring aggregation switches (to provide redundancy between access and aggregation layers). The aggregation layer consists of 4 *pods*, each with 4 switches connected together in a full mesh.

Two of the switches in a pod are connected to each of 20 access switches in a cluster and the remaining two switches are connected to two core switches. The core layer consists of 4 switches connected together in a full mesh. We therefore have a network of size 100 with 80 access switches, 16 aggregation switches and 4 core switches.

The links between access switches and aggregation switches have a capacity of 1. The links between aggregation switches (within a pod) have a capacity of 20. The links between aggregation switches and core switches have a capacity of 40. The resulting network is depicted in Figure 3.

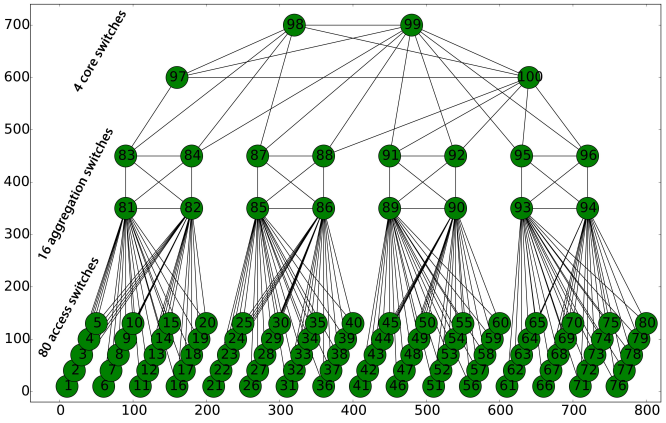

Fig. 3. Structured (infrastructure) topology.

We randomly select subset of nodes that serve as destinations from the group of access nodes. As for the random topology case, depending on the number of destinations, we distinguish scenarios that correspond to *sparse* multicast (3, 4, ... , 10 receivers) and *dense* multicast (15, 20, 25, ... , 40 receivers). All the nodes from the group of access nodes that are not designated as receivers are the candidate sources with the traffic rate uniformly distributed in the interval $[0.025, 0.05]$. Our algorithms try to compute node-redundant trees for as many of them as possible.

B. Results

The results presented her are the output of 20 simulations with different seeds. Hence, every simulation has a different group of sources/destinations and different offered traffic for the sources. We show the average values and the confidence interval for the number of sources placed and the number of installed rules (with and without aggregation).

Random (ad-hoc) topology, sparse multicast:

As shown in Figure 4, there is no significant difference in the number of sources that can be placed when the number of destinations is fewer than seven. Once the number of destinations grows bigger, `Takahashi - Matsuyama` starts paying the price of the fact that it is the only algorithm that does not construct trees in parallel. Consequently, with more than seven destinations, after placing the first of the two trees, it becomes impossible to place the second tree for more than only a few sources. The main bottleneck in how many sources can be placed for the other two algorithms is the aggregated capacity of the links to the destination node with minimum degree, among all destinations. For example, node 93 in the upper-right corner (Figure 2) has only two edges that connect this node to the rest of the graph. Hence, the aggregated capacity that is shared among two tree branches that reach this destination is 2. Therefore, we cannot place more than $21-23$ sources, as the aggregated traffic of this many sources in decreasing order of offered traffic is close to 1. In the case of node 42 (lower-left corner) this capacity is 3. So, if the node 42 is the node with minimum number of edges in the multicast group, the limiting capacity is 50% higher than in the case of node 93, and more sources can be placed (~ 35). The more destinations we have the higher probability is that one of the nodes with two edges will be in the group of receivers; and this is why we converge to $21-23$ sources placed.

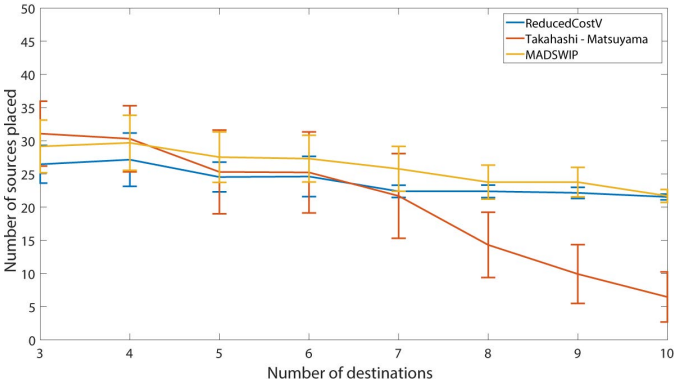

Fig. 4. Number of sources placed for random topology, sparse multicast.

When it comes to the minimum number of hops between sources and destinations (CDFs depicted in Figures 5 and 6), `ReducedCostV` is dominated by the other two. This is expected, as `ReducedCostV` does not have as a goal creation of short paths to specific nodes; the goal is simply to cover all the nodes of a given graph.

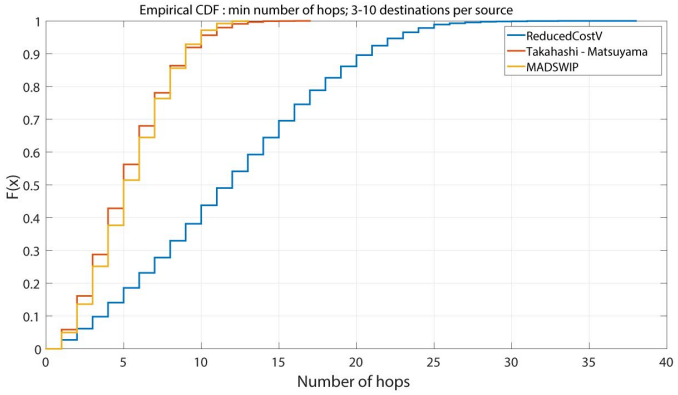

Fig. 5. Minimum number of hops for random topology, sparse multicast.

Looking at the number of rules that need to be installed, both with and without rule aggregation (Figures 7 and 8), we see that `ReducedCostV` is the worst one; but the difference is not so significant if the aggregation is applied. We do not show results for `Takahashi - Matsuyama` for more than seven destinations as, afterward, the number of sources placed becomes very low. For fewer destinations, we see that `Takahashi - Matsuyama` gives slightly better results than `MADSWIP`, as every additional destination is added by minimizing the distance from the nearest node in an already established tree to other destinations within the same multicast group. Whereas, in the case of `MADSWIP`, every destination within a multicast group is treated completely separately.

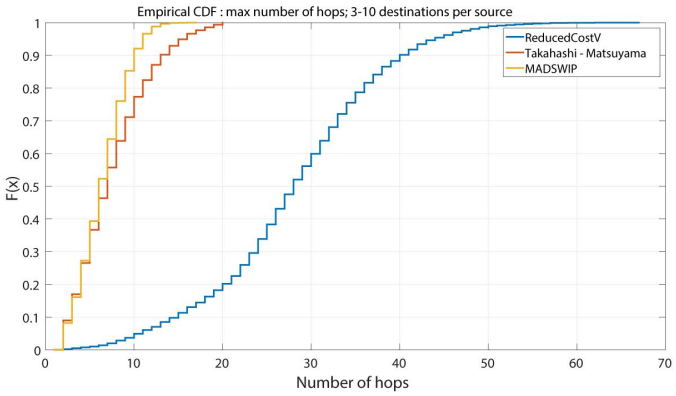

Fig. 6. Maximum number of hops for random topology, sparse multicast.

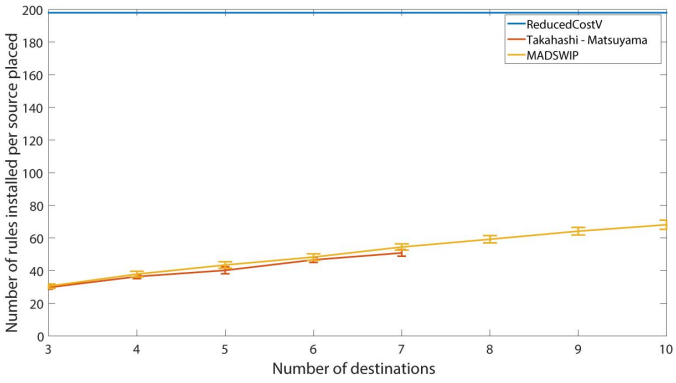

Fig. 7. Number of rules installed without aggregation for random topology, sparse multicast.

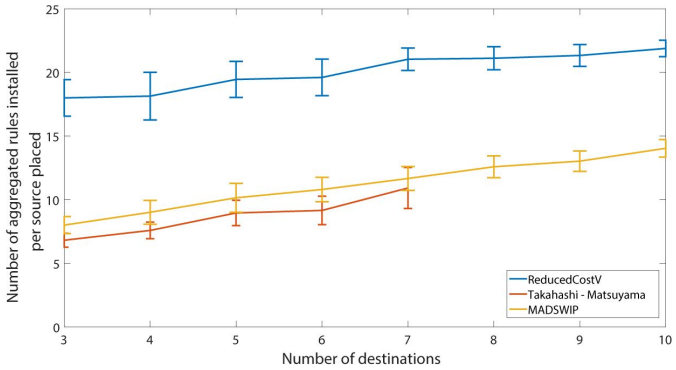

Fig. 8. Number of rules installed with aggregation for random topology, sparse multicast.

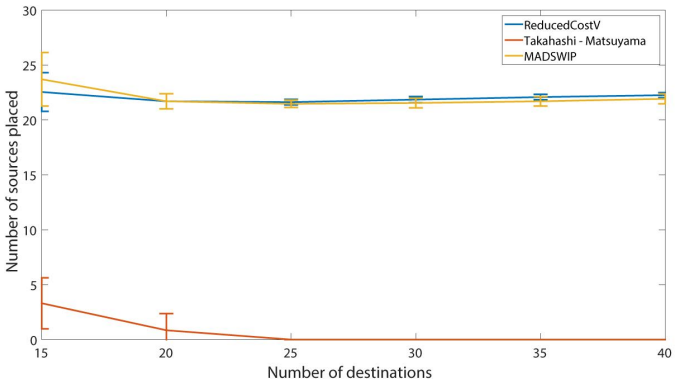

Fig. 9. Number of sources placed for random topology, dense multicast.

Random (ad-hoc) topology, dense multicast:

The trends stay the same when we analyze dense multicast case. The number of sources that can be places stays stable for `ReducedCostV` and `MADSWIP`, whereas the decreasing trend for `Takahashi – Matsuyama` continues, which means that no, or very few, sources can be placed (Figure 9).

The conclusions are unchanged when it comes to the minimum number of hops between sources and destinations (CDF shown in Figure 10), `MADSWIP` is better than `ReducedCostV` and `Takahashi – Matsuyama` is not shown because almost no sources were placed. The same conclusions are valid for the maximum number of hops between sources and destinations (CDF shown in Figure 11). However, `MADSWIP` loses its dominance when it comes to the number of rules that need to be installed both with and without rules aggregation (Figures 12 and 13). Simply, with the increasing number of destinations, the number of affected nodes grows for `MADSWIP` and, in the case of `ReducedCostV`, all the nodes are affected no matter how many destinations there are.

Structured topology, sparse multicast:

Before we analyze the results for the structured topology, we should make one observation. This topology resembles dual-plane topologies and, given that the capacities at the core and aggregation levels are sufficient, the bottlenecks will simply be the links that are connected to the destinations that are part of the multicast group. All the algorithms are "smart enough" to discover two planes, even though they are not explicitly defined so the number of sources places is comparable in all cases, see Figure 14. Again, we converge to $21 - 23$ sources placed, as the aggregated traffic of that many sources (in decreasing order of offered traffic) is close to 1, which is the capacity of the links to the destinations mentioned above.

As for the rest of the results (Figures 15 - 18), for the number of hops, we see again that `ReducedCostV` is dominated by the two others that are comparable. The same holds for the number of SDN rules that need to be installed, and again, the difference is not so significant with route aggregation.

Structured topology, dense multicast:

In this scenario, the bottlenecks are the same as for the sparse case: the links that are connected to the destinations that are part of the multicast group. Consequently, all the conclusions are the same as for the sparse case, and the corresponding graphs are depicted in Figures 19 - 23.

VI. Discussion about SDN-rules update-activity provoked by topology changes

In this section, we analyze the effect of different changes in scenarios to the activity of the SDN control plane. Concretely, we are interested in situations when there is a permanent change in the system that triggers the update of already installed forwarding rules. The goal of the reconstruction is the (re)establishment of node-redundancy property for all the sources and destinations, without disrupting services that are in-progress. Specifically, we analyze the SDN-control-plane activity in case of (i) node failure, (ii) the arrival of a new

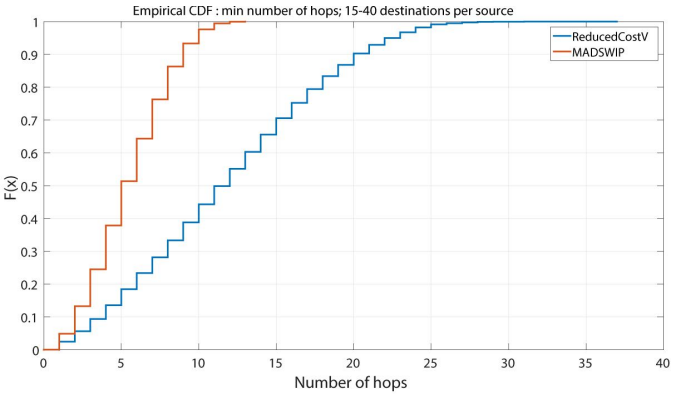

Fig. 10. Minimum number of hops for random topology, dense multicast.

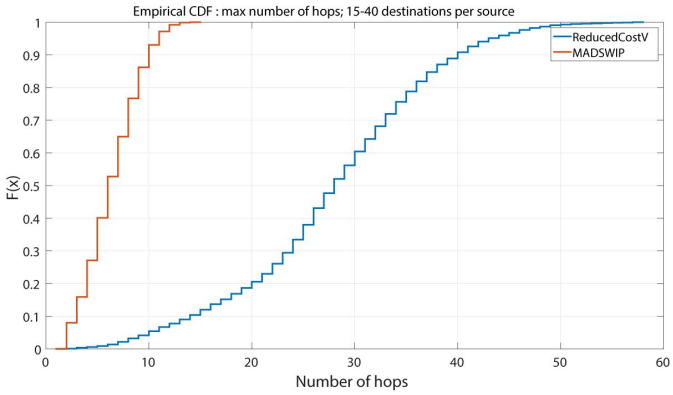

Fig. 11. Maximum number of hops for random topology, dense multicast.

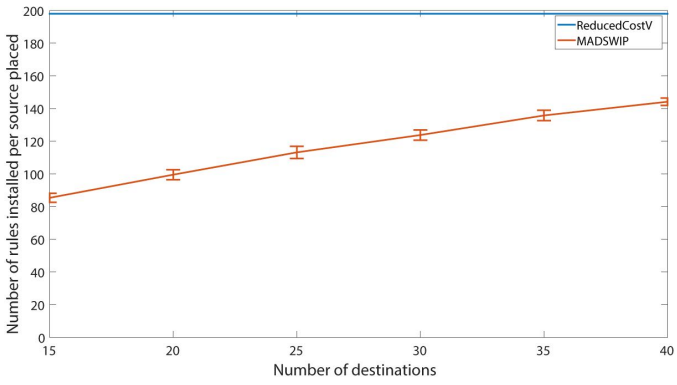

Fig. 12. Number of rules installed without aggregation for random topology, dense multicast.

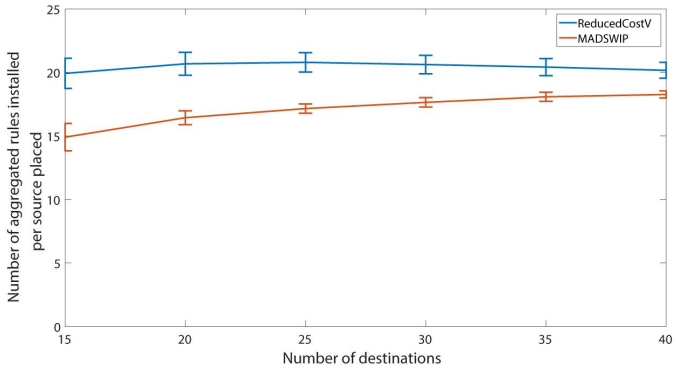

Fig. 13. Number of rules installed with aggregation for random topology, dense multicast.

destination within a multicast group, and (iii) the arrival of a new source. This analysis is carried out under the assumption that, after a change, it is still possible to construct node-redundant multicast-distribution trees without disrupting the connections that are already put in place.

SDN-rules update-activity in the case of node failure: Depending on the position of the failed node, and on the algorithm in use, the effect of such an event can be different. First, we analyze the case of random topology. For `ReducedCostV`, a node failure certainly breaks both spanning trees that are constructed. As a consequence, a new pair of spanning trees has to be computed. For `Takahashi - Matsuyama` and `MADSWIP` the answer is less straightforward; it depends on the position of the failed node. It can be part of none, or very few, of the already established trees, hence its failure will not affect significantly the network operation. Therefore, `Takahashi - Matsuyama` and `MADSWIP` are less vulnerable to node failures, compared to `ReducedCostV`. In order to compare them in more details, we made a simulation analysis to evaluate the probability that a placed pair of trees is affected in case of a random node failure. We did it for cases with 3 and 6 destinations within a multicast group, for the random topology. For both 3 and 6 destinations, the difference in probabilities for these two algorithms is negligible (around 1%). Specifically, we get 25% and 36%, for 3 and 6 destinations respectively.

For the structured topology, failure of an access switch means permanent disconnection for the directly connected source or destination. Failure of an aggregation switch disables one of the trees but the redundancy re-establishment is impossible. This is as we set the level of connectivity between access and aggregation switches to 2 and also because the pods are of size 4, hence, any lower layer aggregation switches is only connected to two upper layer aggregation switches. Therefore, the failure of interest is a failure of one of the core nodes as, in this case, the redundancy re-establishment is possible. Given the topology, almost all of the established trees will be affected and new computations will be needed for all the algorithms.

SDN-rules update-activity in the case of an arrival of a new destination within a multicast group: Here we assume that we add a new receiver in the group of multicast receivers. This does not affect `ReducedCostV` as all the nodes are reachable from all the existing sources and spanning trees are constructed and rules are already put in place. For `Takahashi - Matsuyama` and `MADSWIP`, new calculations will be needed and, as we see from the figures that show the CDFs of the number of hops, the effect is similar as up to 20 nodes will be affected in majority of cases, irrespectively of scenario. To conclude, contrary to the previous case, `ReducedCostV` outpeforms the two other algorithms.

SDN-rules update-activity in the case of an arrival of a new source: Here we assume that a new source starts

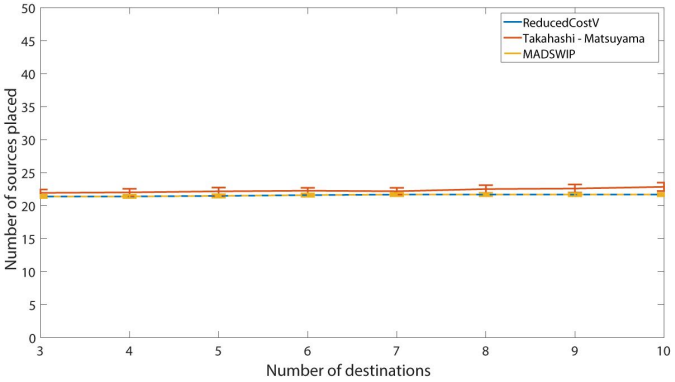

Fig. 14. Number of sources placed for structured topology, sparse multicast.

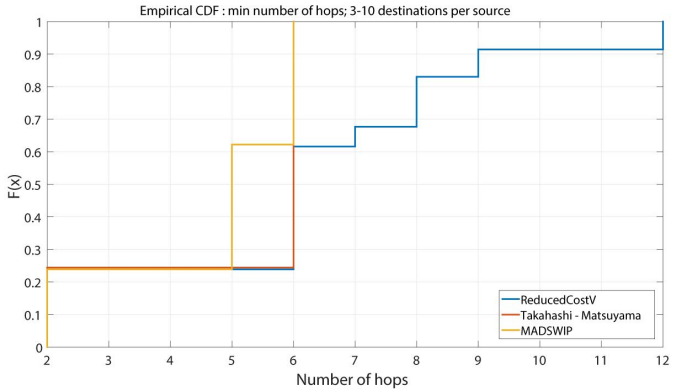

Fig. 15. Minimum number of hops for structured topology, sparse multicast.

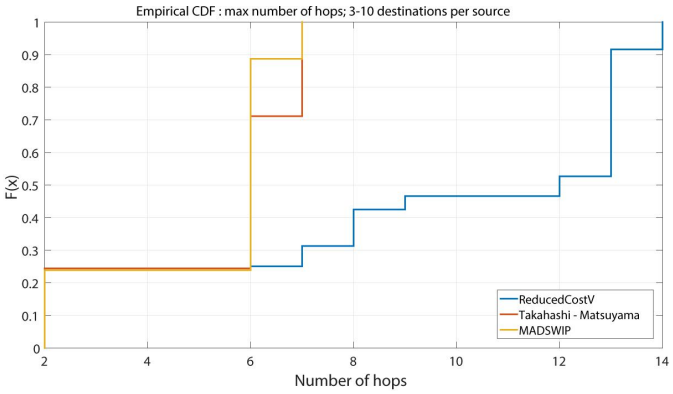

Fig. 16. Maximum number of hops structured topology, sparse multicast.

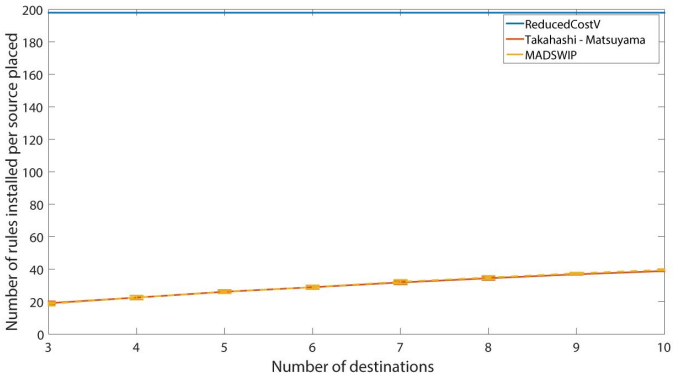

Fig. 17. Number of rules installed without aggregation for structured topology, sparse multicast.

sending traffic to already present group of receivers. For all the algorithms, a new computation of tree pairs is needed. In the case of `ReducedCostV`, an installation of new rules will be needed in all the nodes. For `Takahashi - Matsuyama` and `MADSWIP`, the fraction of the nodes that require new rules depends on the exact scenario. As fewer nodes are affected because we are not creating spanning trees, the conclusion is similar to the one in the first scenario: `Takahashi - Matsuyama` and `MADSWIP` are less affected by source arrivals, compared to `ReducedCostV`, and are very close to each other. Concretely, the figures that show the number of rules installed (with and without aggregation) per source placed are the best comparison of the effect of the arrival of a new source, depending on the used algorithm.

VII. Conclusion

In summary, we can say that `MADSWIP` is the overall winner in this performance comparison. It is robust and it performs the best or comparably to the best algorithm in a wide range of scenarios and metrics. However, there are two exceptions where it makes sense to consider other algorithms. First, if the topology of interest is truly dual-plane and if it is guaranteed that it will remain so, no matter what the changes in the network are, we might consider applying `Takahashi - Matsuyama` as, under such conditions, its performance is comparable to (or even slightly better than) the one of `MADSWIP`. Second, if the rate of arrivals/departures of new destinations within a multicast group is high, which provokes high SDN-control-plane activity, we should consider `ReducedCostV` as a solution. This, of course, if the higher number of hops between source-destination pairs can be tolerated (this translates to less-strict delay requirements).

References

[1] H. Kirrmann, M. Hansson, and P. Muri, "IEC 62439 PRP: Bumpless Recovery for Highly Available, Hard Real-Time Industrial Networks," in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, Sept 2007, pp. 1396–1399.

[2] M. Popovic, M. Mohiuddin, D.-C. Tomozei, and J.-Y. Le Boudec, "iPRP: Parallel Redundancy Protocol for IP Networks," in *Factory Communication Systems (WFCS), 2015 IEEE World Conference on*.

[3] M. Torchia, "Innovative ICT Empower a Better Connected Smartgrid - White Paper," Tech. Rep., August 2014.

[4] D. Gyllstrom, N. Braga, and J. Kurose, "Recovery from link failures in a smart grid communication network using openflow," in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, Nov 2014, pp. 254–259.

[5] T. Pfeiffenberger, J. L. Du, P. Bittencourt Arruda, and A. Anzaloni, "Reliable and flexible communications for power systems: Fault-tolerant multicast with sdn/openflow," in *New Technologies, Mobility and Security (NTMS), 2015 7th IFIP International Conference on*, 2015.

[6] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. ACM Sigcomm '15.

[7] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "Softcell: Scalable and flexible cellular core network architecture," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. ACM CoNEXT '13.

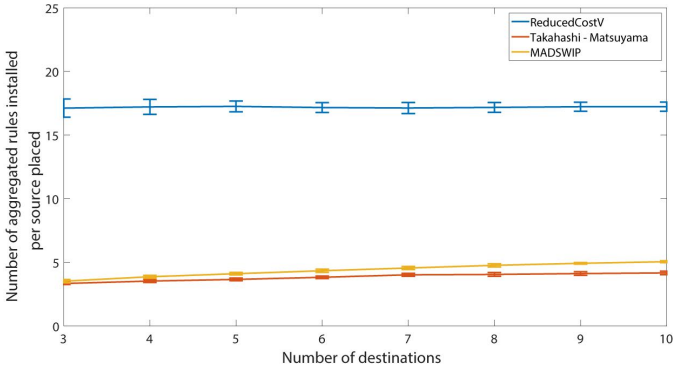

Fig. 18. Number of rules installed with aggregation for structured topology, sparse multicast.

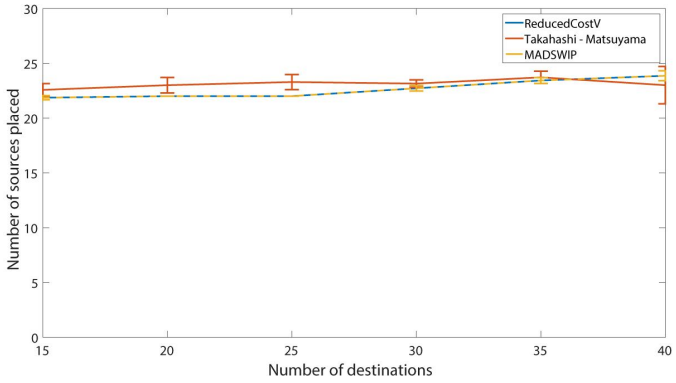

Fig. 19. Number of sources placed for structured topology, dense multicast.

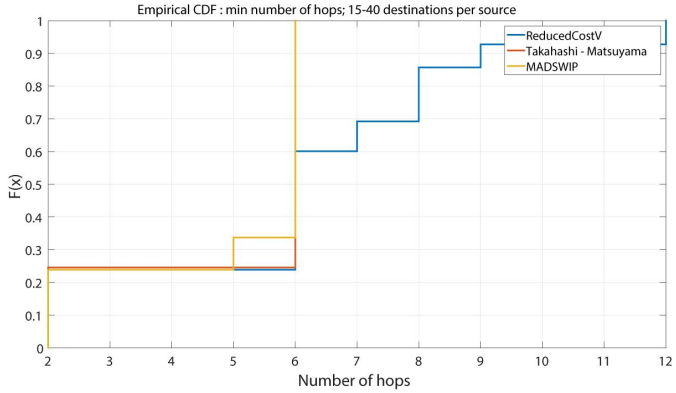

Fig. 20. Minimum number of hops for structured topology, dense multicast.

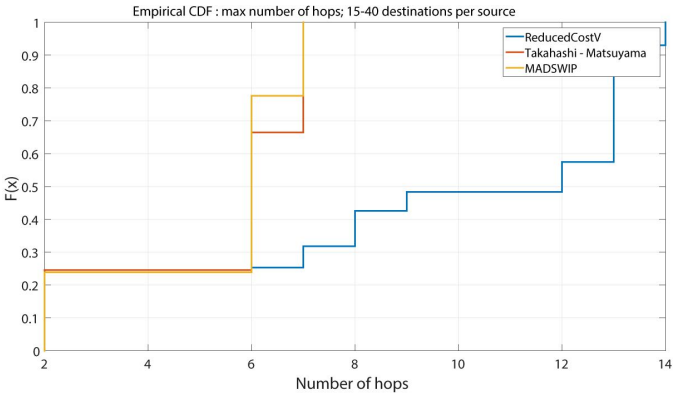

Fig. 21. Maximum number of hops for structured topology, dense multicast.

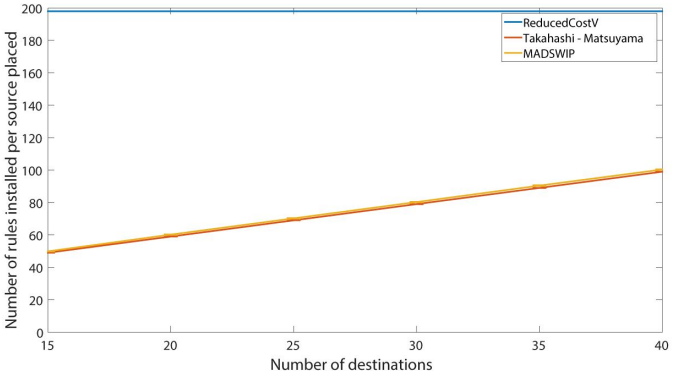

Fig. 22. Number of rules installed without aggregation for structured topology, dense multicast.

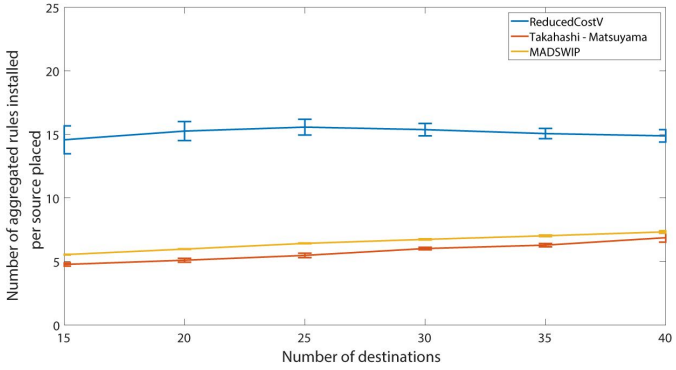

Fig. 23. Number of rules installed with aggregation for structured topology, dense multicast.

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

[9] W. Zhang, G. Xue, J. Tang, and K. Thulasiraman, "Faster algorithms for construction of recovery trees enhancing qop and qos," *Networking, IEEE/ACM Transactions on*, vol. 16, no. 3, pp. 642–655, June 2008.

[10] N. Taft-Plotkin, B. Bellur, and R. Ogier, "Quality-of-service routing using maximally disjoint paths," in *International Workshop on Quality of Service*, 1999.

[11] F. A. Kuipers, "An overview of algorithms for network survivability," *CN*, vol. 2012, pp. 24:24–24:24, Jan. 2012.

[12] M. Médard, S. G. Finn, and R. A. Barry, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM Trans. Netw.*, vol. 7, no. 5, pp. 641–652, Oct. 1999.

[13] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[14] Y. Guo, F. Kuipers, and P. Van Mieghem, "Link-disjoint paths for reliable qos routing," *International Journal of Communication Systems*, vol. 16, no. 9, pp. 779–798, 2003.

[15] M. Médard and A. Sprintson, Eds., *Network coding : fundamentals and applications*. Amsterdam, Boston, London: Elsevier, 2012.

[16] D. J. C. MacKay, "Fountain codes," *Communications, IEEE Proceedings*, vol. 152, no. 6, pp. 1062–1068, Dec 2005.

[17] G. Xue, L. Chen, and K. Thulasiraman, "Quality of service and quality protection issues in preplanned recovery schemes using redundant trees," in *IEEE J. Sel. Areas Commun., ser. Optical Communications and Networking series*, vol. 21, 2003, p. 13321345.

[18] N. Singhal, L. Sahasrabuddhe, and B. Mukherjee, "Provisioning of survivable multicast sessions against single link failures in optical wdm mesh networks," *Lightwave Technology, Journal of*, vol. 21, no. 11, pp. 2587–2594, Nov 2003.

[19] R. Nadiv and T. Naveh, "Wireless backhaul topologies: Analyzing backhaul topology strategies," White Paper, Ceragon, August 2010.

[20] M. Howard, "Using carrier ethernet to backhaul lte," White Paper, Infonetics Research, February 2011.