Peer-to-Peer Communication Trade-Offs for Smart Grid Applications

Preprint

Purboday Ghosh,[1] Shashank Shekhar,[2] Yashen Lin,[3] Ulrich Muenz,[2] and Gabor Karsai[1]

1 Vanderbilt University
2 Siemens Technology
3 National Renewable Energy Laboratory

Presented at the IEEE 31st International Conference on Computer Communications and Networks (ICCCN 2022)
July 25–27, 2022

Peer-to-Peer Communication Trade-Offs for Smart Grid Applications

Preprint

Purboday Ghosh,[1] Shashank Shekhar,[2] Yashen Lin,[3] Ulrich Muenz,[2] and Gabor Karsai[1]

1 Vanderbilt University
2 Siemens Technology
3 National Renewable Energy Laboratory

NOTICE

Peer-to-Peer Communication Trade-Offs for Smart Grid Applications

Purboday Ghosh*, Shashank Shekhar†, Yashen Lin‡, Ulrich Muenz†, Gabor Karsai*

*Vanderbilt University, Nashville, TN, USA; †Siemens Technology, Princeton, NJ, USA
and ‡ National Renewable Energy Laboratory (NREL), Golden, CO, USA

Email: *{purboday.ghosh; gabor.karsai}@vanderbilt.edu, †{shashankshekhar; ulrich.muenz}@siemens.com
and ‡yashen.lin@nrel.gov

Abstract—**Virtual topologies in peer-to-peer networks can reduce the traffic consumed by altering the logical connectivity of peers without altering the underlying network. However, such sparsely connected virtual topologies do not focus on the needs for smart grid applications, which is information dissemination throughout the network, and in turn degrade the performance of distributed control algorithms running on peer-to-peer networks. This paper provides a flexible solution for application developers to prototype and deploy different virtual topologies that balances these trade-offs. First, it introduces a configurable virtual communication topology framework, TopLinkMgr, which enables users to specify any chosen connectivity configuration and deploy peer-to-peer applications using it. Second, it proposes a novel fault-tolerant self-adaptive virtual topology management algorithm, Bounded Path Dissemination, that can ensure the dissemination of information to all peers within a specified number of hops. Experiments show that the algorithm improves on convergence speed and accuracy over state-of-the-art methods and is also robust against node failures while consuming significantly less communication bandwidth.**

I. INTRODUCTION

In peer-to-peer architectures deployed for the modern electric grid, a computational *cyber* component with communication capabilities is associated with the various *physical* elements of the grid, such as distributed energy resources (DERs), solar photovoltaic (PV), sensors, and relays. This realizes the *cyber-physical* system, on top of which various energy management schemes are applied. The integration of intelligent algorithms into the electric grid has essentially given rise to the concept of *Smart Grid* [1].

A peer-to-peer energy management system (EMS) for a smart grid comprises a set of algorithms that achieve a common goal of the system or perform some action. It achieves this through the exchange of information with every other participant without going through a centralized controller. Thus, each component combines its local measurements with the information received from its peers at each iteration to produce an output. The output is usually a type of optimization cost or consensus value (which may be used as a control signal). The output is improved iteratively as more and more information is exchanged.

The traffic pattern for peer-to-peer EMS agents is always any-to-any. In such cases, a *fully connected* peer-to-peer configuration ensures the highest amount of information dissemination throughout the network, but it also generates the highest network traffic at each round. To be precise, in a network of n nodes, fully connected communication generates $n \times (n-1)$ messages per round. This becomes an issue once the size of the system becomes larger as it might cause messages to be delayed or dropped by the network. One way to overcome this bottleneck is to establish a virtual topology on top of the communication network that creates a sparsely connected communication graph. A virtual topology can be defined as an application-layer overlay that determines how each peer shares information with the rest of the network. Using such virtual topologies, peers can choose a subset of neighbors to share information with instead of flooding the network. However, in such a setup, since fewer information is exchanged at each iteration, the convergence performance of distributed algorithms might be affected both in terms of accuracy and speed. Thus, there exists a trade-off between convergence efficiency and the amount of network traffic generated that must be considered by system designers for peer-to-peer EMS. Application designers need to evaluate their algorithms for different communication connectivity configurations before deciding on the best approach.

To address this need, this paper introduces an integrated, configurable framework, *Topology Manager for Peer-to-Peer Links, or TopLinkMgr*, for implementing and deploying various topology configurations on remote edge devices for evaluation as well as production deployments. It allows users to specify any custom connectivity graph using a novel text-based domain specific modeling language, *TopLink*.

However, while expert users can use TopLinkMgr to come up with their own topology, manually altering a given topology each time through trial and error is a tedious task for non-expert users. The initial topology that is deployed can often be improved by a few tweaks to the peer-to-peer connectivity without going through a complete redesign. As discussed, the key criteria for a peer-to-peer smart grid application is the effective dissemination of information to all parts of the network. Thus, the final chosen topology should be able to improve on that characteristic.

This brings about a second challenge - how to devise a

1

strategy that can automatically adapt and improve the dissemination of information to all peers in the case of a sparsely connected peer-to-peer communication topology, especially for non-expert users? In terms of graph theory, the problem can be formally defined as, given a set of peers and V, determining a set of virtual connections $E : V \times V$, such that there exists a virtual path $P_{ij}\{e : e \in E\}$ between any two nodes v_i and v_j and the minimum path length satisfies $1 \leq min(n(p_{ij})) \leq th$. Here, $n()$ denotes the cardinal number of a set and th denotes a maximum threshold.

We introduce a novel group-based topology management and peer selection algorithm *Bounded Path Dissemination (BPD)*. It dynamically adapts a given virtual communication topology to limit the minimum path length within an adjustable threshold. For a safety-critical field such as smart grids, fault-tolerance also becomes an important factor. The system must still be able to maintain its operational goal under multiple node failures. Thus, the proposed algorithm has dedicated fault-tolerance protocols that can prevent partitioning of the virtual network if one or more nodes drop out, thus ensuring information dissemination.

There has been considerable work on the topic of peer to peer network overlays for distributed applications such as large-scale file sharing and look up, load balancing and so on [2]. The aim of this work is to develop a light-weight virtual topology creation algorithm that ensures reliable delivery of application messages for an any to any traffic pattern.

In summary, the main contributions of this paper are as follows:

- **TopLinkMgr**, a configurable framework for implementing and deploying various topology configurations over RIAPS platform that allows the deployment of different peer to peer connectivity configurations.
- **Bounded Path Dissemination (BPD)**, a topology management algorithm that improves the speed of information dissemination without flooding the network and is resistant to node failures.

The rest of the paper is organized as follows: Section II describes existing works found in literature. Section III explains the architecture and process flow of the TopLinkMgr framework, section IV explains the different steps of the BPD algorithm and section V shows the experimental results of applying the algorithm and its effect on peer-to-peer algorithm convergence and fault-tolerance in comparison with some of the alternative methods. Concluding remarks and possible future work are mentioned in Section VI.

II. RELATED WORK

There is a substantial amount of literature available in the field of distributed cloud and fog computing on the topic of overlay networks. An overlay is a virtual network built on top of the network layer, supported by its infrastructure that is capable of selectively forwarding packets to peers in the network, in an application-specific way [3]. Various types of overlay topologies exist in the literature, and they can be categorized into two categories, namely *structured*
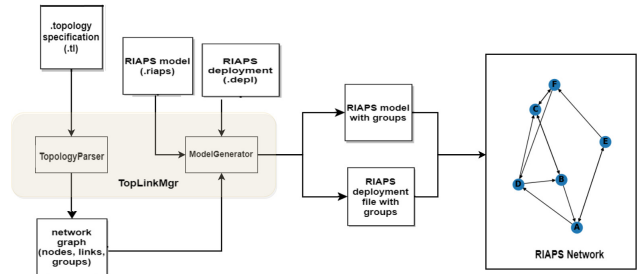

Fig. 1: TopLinkMgr Architecture

and *unstructured*. Structured overlays are static in nature, where the connections are predefined and remain unchanged throughout the operation lifecycle [4], [5], [6]. On the other hand, in unstructured peer-to-peer overlays, connections are determined at run-time and can change at each round [7], [8], [9]. Among them, gossip-based protocols have found widespread adoption in industrial distributed systems use cases such as [10], [11]. In a gossip-style communication, at each iteration a peer randomly selects a subset of peers to send its updates to. Although the ideas and concepts used in these approaches can be generalized and have influenced our design, all are focused mainly on the efficient organization of nodes to optimize parameters such as latency [12], for file storage and look up [13], [14]. In [15], a peer selection strategy is introduced which is built by improving the randomized shuffling protocol. Using this protocol, an efficient topology management scheme was developed in [16] for a distributed file sharing application. Other algorithms seek to minimize proximity to centralized cloud data centers [17] or dynamic clustering in the case of moving nodes, such as vehicles [18]. Although the general principles and solution techniques used in these works have inspired this paper, all of them focus on factors important for large-scale dynamic distributed systems where better load balancing and faster response times are desired. For a peer-to-peer smart grid application, the key factor is the distribution of information generated from all peers to all other peers throughout the network efficiently and quickly.

Overlay solutions that have been implemented recently for smart grid platforms [19], [20], [21]. These approaches mainly look at using some heuristics to optimize the routing efficiency of messages based on information about the power network. However, the experiments performed always assume a network model that has a few hops to any node, which is the problem that we are trying to solve.

Thus, the existing approaches do not solve the performance network usage trade-offs that are important for peer-to-peer smart grid applications, nor do they provide the flexibility for application developers to implement and customize the communication topology for their algorithms.

III. TOPOLOGY MANAGER FOR PEER-TO-PEER LINKS

This section describes our configurable virtual communication topology framework, *Topology Manager for Peer-to-Peer*

Links (TopLinkMgr). It acts agnostic to the overlying application logic and can be leveraged by application developers readily without worrying about the implementation details. The main goal of TopLinkMgr is to allow application users to deploy distributed peer-to-peer applications using different communication structures to evaluate and determine the best option depending on the application needs. In order to facilitate that, the manager also comes with a library of topologies which are commonly employed, along with providing flexibility to users to design a completely custom topology from scratch. All these options can be provided through the use of a simple text-based user specification language that has been developed for the utility, called *TopLink*. Using *TopLink*, users can specify the names of participating nodes, the type of communication topology, as well as define the individual links for a custom topology in a `.tl` file, which is then fed to a *TopologyParser* tool that uses the information to translate the user specifications into the appropriate network graph.

RIAPS Overview: To develop and deploy the smart grid applications for this article, we use an integrated, decentralized software framework *Resilient Information Architecture Platform for Smart Grids (RIAPS)* [22]. RIAPS uses a distributed component-based application model with communicating interfaces called ports, using which application developers can realize any peer-to-peer algorithm. It also comes with several platform-level services, such as remote deployment and control, network discovery, time synchronization and fault-tolerance [23]. These platform features make RIAPS highly suitable as our chosen implementation platform.

The TopLinkMgr framework utilizes one of the features provided by the RIAPS platform called `groups`, that enables the user-specified virtual topology to be realized into a working RIAPS application. A group defines a dynamic grouping of components at the application level. A group is defined as a group name in the application model which can then have multiple instances defined during deployment. RIAPS groups provide several functions that include:

Group communication: Members of groups can send and receive private messages that are only circulated within that group. Sometimes it also might be required to share information across groups. This can be achieved by forming a separate group comprising all leaders as a cross-group communication channel.

Membership management: Components can dynamically join or leave a group at any time. RIAPS also provides API-s for detecting and handling membership changes.

Leader election: RIAPS groups can also perform leader election using a RAFT-based algorithm [24]. Members can also communicate directly with the group leader using special methods.

Voting and consensus: Group members also have the ability to start a voting process on values or actions and gather the final result based on the votes of all other members.

Process Flow: A virtual topology can be constructed by assigning peers to groups based on how one peer is logically connected to another. This is done by the module
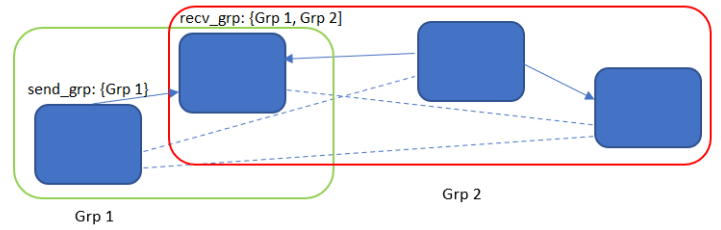

Fig. 2: Group formation. The rectangular outlines represent the groups. The send and receive groups as perceived by the two left-most nodes respectively are illustrated.

TopologyParser, which takes as input a RIAPS model file containing the actor, component and message definitions and a topology specification file written using *TopLink*. For each set of communicating links that connect a source peer to any number of destination peers, a group needs to be created. In this context, a group can be defined as a means of establishing the information flow relation between nodes. In TopLinkMgr, there are two categories of groups, a *send group* for outward communication, and a *receive group* for inward communication. The number of groups is determined by the defined virtual topology, and then each node is assigned to be part of these groups based on the virtual connections. Group categories are relative to each node.

For example, consider the graph of Figure 2 with four nodes, represented as blue rectangles that are logically connected according to the solid arrows. The direction of the arrows specifies whether the node can send messages to the other node or receive messages from it. The dotted lines represent unused communication links that are not part of the virtual topology. In order to implement this configuration, it would require the formation of two groups, *Grp 1* comprising the left two nodes and *Grp 2* comprising the right three nodes. Since the left most node acts as the sender, TopLinkMgr will assign *Grp 1* as its *send group*. Similarly, the node second from the left receives from the other nodes of both *Grp 1* and *Grp 2*. Thus, ToplInkMgr will assign both groups as its *receive group*.

Finally, once the group assignments are calculated, a RIAPS model deployment file is generated by the *ModelGenerator* utility which also weaves the group configurations and necessary commands into the model file. If a deployment file already exists, the *ModelGenerator* modifies it by adding group arguments, or else it generates a new file. It also automatically generates the required component code to join the appropriate groups. Thus, the output of the entire process flow is a complete working application that implements the specified communication topology among the specified peers. The complete architecture and the process flow are depicted in figure 1.

TopLink Specification Language TopLink is a text-based domain-specific modeling language that can be used by application developers to describe their chosen topology, either a pre-configured or a completely custom one. It provides special

3

```
// RIAPS Application Model
app GroupTestApp {
    // message types
    message SensorData;
    message NodeData;
    ...
    // Sensor component
    component Sensor(value=1.0) {
        timer clock 10000;
        pub sensorReady : SensorData ; // Publish port
    }
    // Averager component
    component Averager(Ts=1.0){
        sub sensorReady : SensorData ;   // Subscriber port
        pub thisReady : NodeData; // Publish port
        sub nodeReady : NodeData; // Subscriber port
        ...
    }
...
    // Averager actor
    actor Averager(value=0.0,Ts=100.0){
        local SensorData;         // Local message types
        {
            sensor : Sensor(value=value);
            averager : Averager(Ts=Ts);
        }
    }
...
}
}
```

(a) RIAPS application model

```
// topology configuration for RIAPS app
// Actor.Component
for Averager.Averager {
create topology custom; // topology name
over (bbb1, bbb2, bbb3, bbb4, bbb5, bbb6); // nodes
using (bbb1, bbb2), (bbb2, bbb3), (bbb2, bbb6), (bbb3, bbb4), (
    bbb3, bbb6), (bbb4, bbb5), (bbb5, bbb3), (bbb6, bbb1), (bbb6, bbb5) ;
    // links
inter-group communication on; // turn on communication between group leaders
}
```

(b) TopLink input file (all features not shown)

Fig. 3: TopLinkMgr input files showing an example RIAPS application model and a TopLink specification

keywords through which users can define different aspects of the topology. Properties that can be defined using the file include:

- The RIAPS application actor and component for which the connectivity is being defined.
- The type of topology: either preset or custom. Currently, the preset topologies that are supported are a ring type and a fixed fan-out randomly generated graph, but in the future we plan to add more.
- The node identifiers for the network (IP addresses and host names).
- The individual communication links between the nodes (if the topology is custom).
- Weights for individual links if the network links are not uniform.
- If communication is desired among the leaders of the formed group, it can also be enabled.

Figure 3 shows a RIAPS application model for a peer-to-peer algorithm that defines the various components, their communication ports and port message types, the actor that encapsulates them, and a corresponding topology specification file that describes a custom topology. Running TopLinkMgr produces a modified model file, a RIAPS deployment file, and the various components' starter code as its output. The modified model file contains the initialization of the TopLink group and the associated group message type. The deployment

file contains the details of how the application actors will be deployed to the various remote nodes (denoted by their IP addresses or their hostnames) and the specific arguments for each actor deployed, if required. The TopLinkMgr generated deployment sets the send and receive group names (which are instances of the TopLink group) as arguments for the corresponding actors based on the topology. Although some features of the language described are based on the RIAPS modeling architecture, the language itself is not tied to RIAPS and can be made to work with other platforms as well using dedicated parsers.

IV. BOUNDED PATH DISSEMINATION ALGORITHM FOR TOPOLOGY MANAGEMENT

Let us consider a fairly common peer-to-peer use case for smart grids, the distributed optimal power flow (OPF) problem [25] for a network of microgrids [26]. OPF is an optimization problem that minimizes the total generation dispatch cost while satisfying physical and technical constraints on the network [25]. Using the primal-dual algorithm to solve the given problem leads to equations 1 (primal update) and 2 (dual update). In both equations, k stands for the k-th node and t for time instant t. x and $\hat{\lambda}_t$ are the primal and dual variables respectively. α, β, γ act as weights. In practice, each microgrid control node keeps a local copy of the dual variable. At each iteration, this dual variable is updated both by local measurements as well as communication received from other controllers. This is reflected in the dual update equation 2 where the first term is updated from measurement data and the second term is updated from communication data. Of course, the implementation of such an algorithm requires that the nodes be synchronized (time- or event-based) to drive the iterations. The performance of the algorithm compared to a fully centralized system for sparse communication connectivity becomes worse than all-to-all connectivity.

$$x_{k,t+1} = Proj_\chi[x_{k,t} + \alpha_{k,t}(\nabla f_k(x_{k,t}) + A_k\hat{\lambda}_{k,t})] \quad (1)$$

$$\hat{\lambda}_{t+1} = Proj_\mathbb{R}[\hat{\lambda}_t + \beta_t M(y - b)] - \gamma_t L\hat{\lambda}_t \quad (2)$$

As discussed previously, the convergence of peer-to-peer smart grid algorithms such as the one discussed above, both in terms of how close the final solution is to an optimal one and how many iterations it takes to reach it, depends on the proper dissemination of information. Thus, any virtual topology deployed must achieve that balance between proper distribution of messages and limiting network traffic. Another important consideration for a safety-critical system such as smart grids is its resilience to unexpected faults that cause crashes of individual nodes and lead to partitioning of the virtual topology.

In order to address these requirements, we introduce the *Bounded Path Dissemination* algorithm. This algorithm improves the dissemination of information to all peers in the network by reducing the hop length of the path from one

4

node to all the other nodes within a threshold, which can be specified as an input parameter. Threshold of 1 implies a fully connected graph. BPD limits the minimum number of iterations required for the message generated by a particular node in the network to reach all its peers. This results in speeding up of the information coverage throughout the network which effectively improves the performance of a peer-to-peer application algorithm over the unmodified scenario. Additionally, BPD also has dedicated self-healing capabilities that can ensure that peers remain connected even if one or more drop out due to faults.

The following notations and definitions will be used throughout the rest of this section.

- $node_i = (id, send\_grp_i, recv\_grp_i)$ denotes a participating node in the network. It acts as the endpoint for peer-to-peer communication. It consists of an identifier id, a sending group object $send\_grp_i$ and a receiving group object $recv\_grp_i$ that realizes the topology.
- id is a unique identifier for a node. In the case of RIAPS, it is a generated universally unique identifier string (uuid).
- $grp_m = (id, size, send(), recv())$ denotes a group object. It consists of an identifier for the group, the attribute $size$ that stores the number of members currently present in the group and two methods $send()$ and $recv()$ for sending and receiving messages to and from that group.
- $send\_grp_i = \{grp_m.id, weight send()\}$ refers to the sending group object of $node_i$. It contains the id of the group that $node_i$ has joined as a send_grp. The weight attribute represents the cost incurred for the outgoing links of $node_i$. The default value is 1. The send and receive operation here refers to invoking them on all groups with ids present in send_grp.
- $recv\_grp_i = \{grp_m.id, recv()\}$ refers to the receiving group object of $node_i$. It contains the ids of the groups that $node_i$ has joined as a recv_grp. The send and receive operation here refers to invoking them on all groups with ids present in send_grp.

The algorithm comprises two stages, namely *Discover Peers* and *Group Update*. The first stage dynamically gathers information about the various node-to-node paths within the topology and their lengths, while the second stage then uses that information to alter some of the paths to ensure that they all lie within the defined threshold. For our setup, an external agent called TopologyManager was responsible for sending commands to each peer to initiate the algorithm stages. However, the same can be triggered by using an internal clock. For smart grids, the number of nodes is determined by the elements of the grid. The network is closed and private for security. Thus, there is no node churn (unlike cloud-distributed systems [27]). Therefore, the algorithm starts with an initial configuration of the nodes generated by the topology framework. However, the discovery phase still ensures generalizability since the path lengths are calculated dynamically without any knowledge of the initial configuration.

A. Stage 1: Discover Peers

In this stage, all peers share information regarding themselves which is propagated through the topology. This information is used by each node to ascertain and store the minimum paths to all other peers in the network in an internal table. It begins when TopologyManager sends a discoverPeers message to all peers in the network. On receiving the message, each node creates a new message containing its own id, a depth variable initialized to 0, and sends it to each of the groups in its recv_grp. It also puts its recv_grp id into the message.

On receiving the message, the recipient checks if the group id matches with its send_grp id. If the depth field is lower than the previously recorded depth of the same node, then it updates the table entry. Next, it increases the depth by the weight associated with that send_grp and forwards it to its recv_grp. At the end of this round, each node will have a complete table containing the node ids and the depth corresponding to that node, indicating the path cost. The steps are shown in Algorithm 1.

Algorithm 1 BPD Stage 1: DiscoverPeers

Input: Initial $send\_grp_i$ and $recv\_grp_i$ for $node_i$
Output: $path_i.node_j \leftarrow (\{node_j : depth_{ij}\} \forall node_j)$ for $node_i$, $i \neq j$
1: **on event** $DiscoverPeers$ **do**
2: $msg_i \leftarrow \{$ id : $node_i.id$, depth: 0, 'grp' : $recv\_grp_i$, 'order' : 0, 'weight': $recv\_grp_i.weight \}$
3: $recv\_grp_i.send(msg_i)$
4: **on event** $send\_grp_i.recv(msg_j)$ **do**
5: **if** $msg_j.grp = send\_grp_i.id$ **then**
6:    $msg_j.depth \leftarrow msg_j.depth + msg_j.weight$
7:    **if** $msg_j.depth < path_i.node_j.depth$ **then**
8:      $path_i.node_j \leftarrow msg_j$
9:      $recv\_grp_i.send(msg_j)$
10:   **end if**
11: **end if**

B. Stage 2: Group Update

This stage uses the fully fleshed-out table entries from Stage 1 to limit the minimum path cost from all nodes of the network to every other node based on a threshold. This threshold is a parameter that can be decided by the user using the connectivity and the desired information dissemination rate. For the experiments performed in this paper, a threshold was selected using the formula $thresh = (N-1)/2$, where N is the number of nodes in the network, which is half the number of edges required to form a spanning tree, the minimal fully connected graph comprising N nodes. Application designers can select a threshold based on how much they want to compromise convergence performance while trying to reduce bandwidth consumption. It begins when TopologyManager sends a *groupUpdate* message to all peers in the network. On receiving the message, all nodes look at the paths that they have stored locally. If for any path $depth > threshold$, then

that node sends a message containing its id, new depth = 0 and weight to its send_grp.

On receiving the message, the recipient node increases the depth by the weight of its send_grp and forwards the message to it. When the message reaches a node for which $depth = thresh - weight$, it appends its send_grp id to the message and forwards it. When the message reaches the target node, the node reads the group id from the field and joins that group. This effectively means that a new link is added to the network graph linking the target node to an intermediate node between it and the source such that the total path depth from the source does not exceed $thresh$. It terminates when there are no new messages to forward. The steps are shown in Algorithm 2.

Algorithm 2 BPD Stage 2: GroupUpdate

Input: $path_i$ from Stage 1 for $node_i$, $thresh$
Output: $path_i.node_j \leftarrow (\{node_j : depth_{ij} \mid depth_{ij} \leq thresh\} \forall node_j)$ for $node_i$, $i \neq j$
1: **on event** $GroupUpdate$ **do**
2:   **for** $node_j, depth_j$ in $path_i$ **do**
3:     **if** $depth_j > thresh$ **then**
4:       $msg_i \leftarrow \{$'req': $node_i.id$, 'id' : $node_j.id$, 'length' : 0, 'grp': "", 'send_grp' : $send\_grp_i.id \}$
5:
6:     **end if**
7:   **end for**
8: **on event** $recv_g rp_i.recv(msg_j)$ **do**
9: **if** $msg_j.req! = node_i.id$ **then**
10:   **if** $msg_j.depth <= thresh - weight_k$ and $msg_j.grp ==$ "" **then**
11:     $msg_j.depth \leftarrow msg_j.length + weight_k$
12:     **if** $msg_j.depth == thresh - weight_k$ **then**
13:       $msg_j.grp \leftarrow send\_grp_i.id$
14:     **end if**
15:   **end if**
16:   $send\_grp_i.send(msg_j)$
17: **else**
18:   $joinGroup(msg_j.grp)$
19:   $send_g rp_i.id \leftarrow msg_j.grp$
20: **end if**

Figures 4a and 4b illustrate the working of the two stages of the algorithm for an example 6-node topology. The threshold selected is 3. In Figure 4a, it can be seen that after running the *DiscoverPeers* protocol to determine the relative path lengths for all nodes, for node 4, nodes 1 and 2 are further than the threshold. Thus, when the group update protocol is triggered, nodes 1 and 2 join the send_grp of node 3, which establishes a path from node 4 to nodes 1 and 2 via node 3. Thus. it can be seen that after completion of the round, the new (shortest) path of nodes 1 and 2 from node 4 becomes 2, which is within the set threshold. Thus, BPD ensures that any message sent from node 4 would reach nodes 1 and 2 within 3 iterations, provided that node 3 is intact. However, that might not always be the case, as unexpected faults might cause nodes to drop out of the network. How BPD can handle such scenarios is

(a) Node 4 after Stage 1
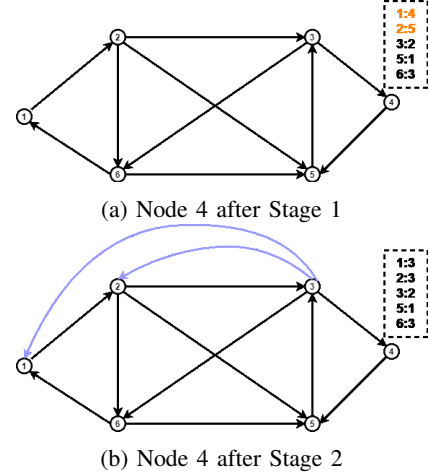

(b) Node 4 after Stage 2

Fig. 4: Bounded Path Dissemination Algorithm on a 6 node topology

discussed in the next subsection. It must be noted that there can be multiple solution topologies that satisfies the requirement. BPD currently does not differentiate between them since in terms of minimum path length they are all equivalent. However, in future works, the algorithm can be optimized, say for e,g., the solution time, or minimum number of edges etc. The algorithm will converge, provided that the starting topology is connected. The speed of convergence depends on both the number of nodes and the threshold selected.

C. Fault Tolerance

The objective of the fault tolerance protocol is to ensure that the information originating from healthy nodes is distributed to all other peers when one or more nodes drop out due to failure. When such faults happen, it can lead to two things, either some of the path lengths can be altered and exceed the threshold since an intermediate route is removed, or one or more nodes can become isolated or partitioned from the rest of the network. In the first case, the algorithm can self-repair by running the two stages periodically. The period used for the experiments in this paper was 2 min. For the second case, there can be two scenarios. The fault-tolerance protocol utilizes the leadership feature of the groups.

1) When a member of the send_grp goes offline: Since for each group, the source node joins it as a send_grp itself and all other nodes that receive from it join that group as a recv_grp, if a group has more than one member, it implies that there exists a path from one peer to the other. If a peer is the only member of its send_grp, it implies that that node is partitioned from the rest of the network. The algorithm then proceeds to connect that node to the rest of the network by joining a new group. If a peer is the lone member in a group, then it sends a *join_req* message containing its own id and 'grp_type' as 'send_grp' to the group leaders. The leaders respond with a *join_rep* message by adding to *join_req* the group id and the group size of the groups in their recv_grp. On receiving the message, the requesting peer chooses the entry with the minimum size and
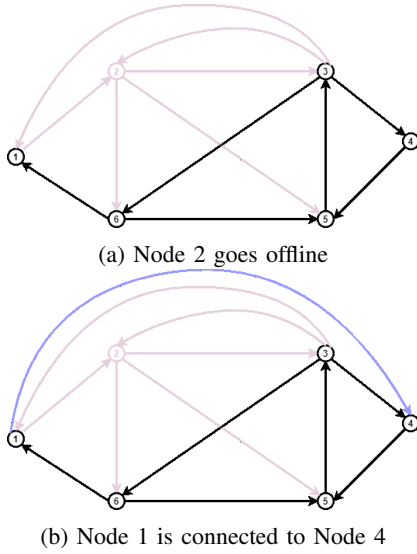
6

(a) Node 2 goes offline

(b) Node 1 is connected to Node 4

Fig. 5: Fault-tolerance protocol for the scenario when a member of the send_grp foes offline

joins that group as specified in the 'grp_type' field, in this case as a send_grp. The steps are described in Algorithm 3

In Fig. 5, when node 2 goes offline, node 1 becomes the lone member of its send_grp, it then initiates the protocol and joins the recv_grp of node 4, thus reconnecting it.

Algorithm 3 BPD Fault-tolerance: If send_grp member leaves

1: **on event** $MemberLeft(group)$ for $node_i$ **do**
2: **if** $group.id$ in $send\_grp_i.id$ **then**
3:    **if** $group.size < 2$ **then**
4:       $join\_req_i \leftarrow \{$ 'req' : $node_i.id$, 'grp_type' : 'send_grp'$\}$
5:       $leader\_grp.send(join\_req_i)$
6:    **end if**
7: **end if**
8: **if** $Memberof(leader\_grp)$ **then**
9:    **on event** $leader\_grp.recv(join\_req_j)$**do**
10:    $mingrp \leftarrow minSize(recv\_grp_i)$
11:    $join\_rep_j \leftarrow \{$ 'req' : $node_i.id$, 'grp_type' : 'recv_grp', 'grp': $mingrp.id$, 'size' $mingrp.size$ $\}$
12: **end if**
13: **on event** $leader\_grp.recv(join\_rep_i)$
14: **if** $count(join\_rep_i) = leader\_grp.size - 1$ **then**
15:    $mingrp \leftarrow minSize(join\_rep_i.grp)$
16:    $joinGroup(mingrp)$
17:    $send_grp_i.id \leftarrow mingrp.id$
18: **end if**

---

2) When a member of the recv_grp goes offline: When a node detects a member of the recv_grp has left, it can imply two possibilities, one, that one of the other receivers in the group has crashed, or two, that the sender node for that group has crashed. For possibility one, the node does not need to act because it still remains connected to its sender, but for

possibility two, the node must join a new group. Once a peer detects it, it sends a *grp_qry* message on that particular group containing the group id. If a recipient has the same group as its send_grp, it responds with its own id. Once the node receives all the responses, if none of the responses contains the sender's id, it implies that the sender is offline. It then proceeds to send a *join_req* as described previously but with 'grp_type' as 'recv_grp'. The steps are described in Algorithm 4.

In Fig. 6, when both node 3 and node 6 go offline, node 4 and node 1 become isolated from the rest of the topology, it then initiates the protocol and node 4 joins the send_grp of node 1 while node 1 joins the send_grp of node 5. As a result, all the nonfaulty nodes 1,2,4 and 5 become connected again with both incoming and outgoing routes to each other. In this case, none of the new paths exceeds the threshold distance of 3. However, there might be certain scenarios where that might be the case. In such cases, the next round of the *DiscoverPeers* and the *GroupUpdate* stages can again restructure the connectivity graph to ensure a bounded path.

Algorithm 4 BPD Fault-tolerance: If recv_grp member leaves

1: **on event** $MemberLeft(group)$ for $node_i$ **do**
2: **if** $group.id$ in $recv\_grp_i.id$ **then**
3:    $grp\_qry_i \leftarrow \{$ 'req' : $node_i.id$, 'grp_type' : 'recv_grp', 'grp' : $group.id$ $\}$
4:    $recv\_grp.send(grp\_qry_i)$
5: **end if**
6: **on event** $recv\_grp_i.recv(grp\_qry_j)$**do**
7: **if** $grp\_qry_j.grp = send\_grp_i.id$ **then**
8:    $grp\_ans_i \leftarrow \{$ 'req' : $node_i.id$, 'grp_type' : 'recv_grp', 'grp' : $group.id$, 'rep' : $node_i.id$ $\}$
9: **else**
10:    $grp\_ans_i \leftarrow \{$ 'req' : $node_i.id$, 'grp_type' : 'recv_grp', 'grp' : $group.id$, 'rep' : '' $\}$
11: **end if**
12: **on event** $recv\_grp_i.recv(grp\_ans_i)$
13: **if** $count(grp\_ans_i) = recv\_grp_i.size - 1$ **then**
14:    **if** $grp_ans_i.rep == $ '' $\forall grp\_ans_i$ **then**
15:       **do** Algorithm 3 with grp_type = 'recv_grp'
16:   **end if**
17: **end if**


V. EVALUATION AND RESULTS

The experiments performed mainly looked at evaluating two aspects: the performance of a peer-to-peer algorithm in terms of its convergence accuracy and speed as a result of using the algorithm as well as the resilience properties of the algorithm under fault conditions, which were the objectives that the algorithm was designed for.

A. Experimental setup

The experiments were carried out using a network of six beaglebones [28] running RIAPS. The communication was carried out using Ethernet. A virtual machine running Linux

7
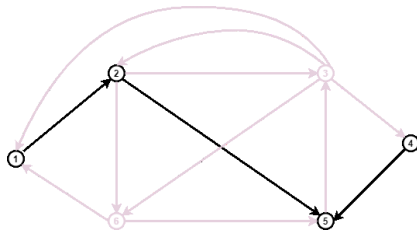
(a) Node 3 and Node 6 go offline
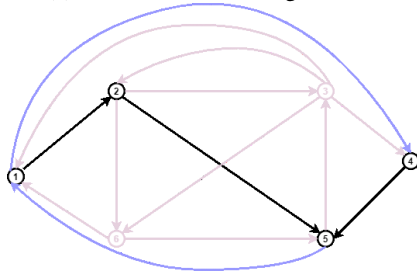
(b) Node 4 and Node 5 are connected to Node 1

Fig. 6: Fault-tolerance protocol for the scenario when a member of the recv_grp foes offline

Ubuntu 18.04 was used as the control node from which the applications were deployed to the target nodes using the RIAPS control graphical user interface.

The algorithm was compared with the base topology (of Fig 1) without any modification, a fully peer-to-peer approach with all-to-all communication, and a randomized gossip style *pull* pattern that is used in most state-of-the-art schemes. The peer-to-peer algorithm used to perform the experiment was a simple distributed consensus algorithm. For a linear system of N nodes with the local state of the i-th node being denoted as x_i, the distributed consensus algorithm updates the state according to the equation 3. For discrete-time, it can be modified to the form of equation 4, with the derivative being replaced by the difference operator for the discrete case.

$$\dot{x}_i(t) = \sum_{j \in N \setminus i} a_{ij}(x_j(t) - x_i(t)) \qquad (3)$$

$$x_i[k+1] = x_i[k] + \sum_{j \in N \setminus i} a_{ij}(x_j[k] - x_i[k]) \qquad (4)$$

Here, a_{ij} represents the connectivity between node i and node j. It can be shown that the algorithm theoretically converges to the average of the initial states $x_i[0]$ of the system.

The above algorithm was coded into the Averager RIAPS component logic for the application model described in figure 3a in Python. A logger component was also added to the model to collect data for the experiments. The application was allowed to run for 10 minutes for each test to ensure sufficient time for the averaging algorithm to converge. The metrics evaluated for algorithm convergence were the percentage deviation from the optimal value (true average) and the

Method	Deviation from optimal value	Min. number of iterations to reach $\pm 5\%$	Messages per iteration	Time taken (s)
All-to-All	4.55%	445	30	3
Gossip	10.76%	172	18	1.8
Unmodified	11.32%	320	10	1
BPD	6.28%	60	13	1.3

TABLE I: Convergence Performance Comparison

number of iterations taken to reach within a 5% tolerance band of that value. The number of messages generated per iteration was also recorded. As expected, all-to-all had the highest with 30 (6×5), followed by Gossip which had 18 (3×6), the graph of Figure 4 initially had 10 links and BPD added 3 more to make the paths bounded. The initial values for the different for the peer-to-peer averaging algorithm were changed such that it led to a different optimal value for each run of the experiment to eliminate any bias in the results.

B. Algorithm Convergence Results

Table I lists the convergence performance of the different topology algorithms. The time taken to execute one pass of the control function was 0.01 ms, averaged across all nodes using the above described setup. In terms of peer selection strategy, as expected, a fully connected network produces the most accurate results with a deviation within 5% of the theoretical optimal value. However, in doing so, it also consumes the most messages per round (iteration), further emphasizing the trade-off that was discussed previously. Bounded Path Dissemination (BPD) improves upon the other approaches on both convergence accuracy and speed. It reaches the steady state the quickest, but the value is not optimal. In our experiments, we also observed sharper fluctuations in the values as the algorithm progressed through successive rounds compared to the all-to-all configuration. This is because the new information received caused a larger correction in some rounds than in others. Gossip performance lies somewhere in between. Due to its inherently random nature, some nodes perform well, since they can receive all the information, but other nodes might take longer to receive the same. However, Gossip will perform better when a large number of nodes are incoming and outgoing, since the algorithm is automatically scalable.

C. Fault-tolerance Results

Since the operational efficiency of a peer-to-peer smart grid application depends on the effective dissemination of information to all participating nodes in the network, we measure the success of our fault-tolerance logic on the basis of its ability to maintain network coverage in the presence of faults. Gossip reliability is a measure that has been traditionally used to evaluate broadcast algorithms. It is defined as the percentage of active nodes that can transmit a gossip broadcast, with 100% denoting a perfectly reliable broadcast [29]. However, reliability only considers messages originating from one source and forwarded by others until they reach all

Method Failure	Gossip	Unmodified	BPD
17 % (1/6)	0.82	0.54	0.83
33 % (2/6)	0.66	0.17	0.66

TABLE II: Dissemination Performance Comparison under Faults
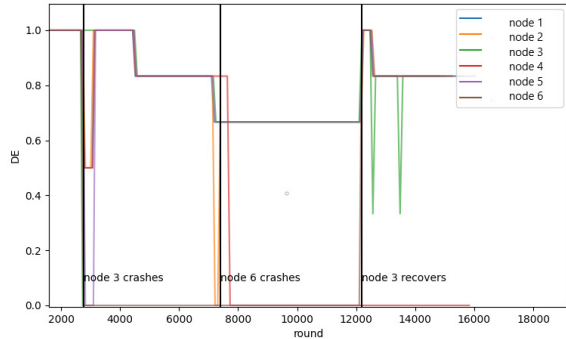

Fig. 7: Node-wise Dissemination Efficiency for BPD showing faults and recovery

other nodes. It does not capture the information that shows whether messages originating from all nodes reach all other nodes, which is the case for a peer-to-peer application. Thus, we slightly modify the definition of reliability to be the fraction of messages originating from distinct source nodes that were received by other peers in a peer-to-peer network. We can call this altered reliability the *dissemination efficiency (DE)*. Its value ranges from $0 - 1$, where 1 implies that all peers received information generated by every other active member, while 0 implies that no messages were received.

Table II shows the average DE for the various techniques under 17% and 33% node failure scenarios. It can be seen that the unmodified topology suffers heavily due to the network being partitioned. Gossip is inherently robust since it randomizes the recipients and ensures that faulty nodes are eventually replaced in the sending list. BPD performs similarly to Gossip since it is able to reconnect the graph, thus ensuring that the remaining nodes are able to effectively receive information from each other.

Figure 7 shows how the algorithm affects individual nodes in real-time under fault conditions for the starting topology of Fig 4a. The plot shows the change in the DE for each node in response to node faults as the number of rounds of message exchange progresses. For the experiments conducted, the period chosen for each round was 10 ms. It shows that the DE stabilizes to the expected values of around $0.83(5/6)$ and $0.667(4/6)$ after node 3 and node 6 crash respectively It also shows that the algorithm is able to restore the connectivity once a faulty node is repaired and it rejoins the network. The average time taken by BPD to restore the topology connectivity following a fault was 16 ms. There are some cases where it shows the DE to jump up to 1 right after a

fault. This is because to calculate the metric, each node stores a history of the values that it receives and its source, which is refreshed periodically. Thus, it takes some time for the residual entries to be removed, and the true value is then reflected. We will investigate more efficient methods for recording metrics in the future, possibly by adding tags to all outgoing messages or using sequence numbers to differentiate between current and older entries. However, this is only related to the way the data shows up and has no effect on the algorithm performance. The sharp downward spikes are due to the time it took for the BPD protocol to be completed after detecting that a group member left. In all our experiments this delay was within the range of 10 ms, which is quite low compared to the usual sampling time period of peer-to-peer EMS algorithms of > 100 ms.

D. Network Measurements

Table III shows the average bandwidth consumed per node and latency data captured for the different methods. As seen from the data, BPD uses about 80% less network bandwidth compared to the fully connected configuration, with the performance of the other two lying in between. This is expected since those two do not take any steps to optimize the virtual topology. However, the methods that employ virtual topologies (BPD modifed and the unmodified) produced a higher latency than the other two. This is due to the fact that using a virtual topology implies that each message needs to go through an additional layer of routing on top of the physical network routing, while in the other two cases it only needs to go through the physical network routing. Although the extra delay is still negligible for it to affect the experiments in this paper, we plan to study it in more detail for larger networks in the future.

Method	Bandwidth consumed (kB/s)	Latency (ms)
All-to-All	33	1.2
Gossip	21.5	0.88
Unmodified	12	3.6
BPD	6.8	2.4

TABLE III: Bandwidth consumed and latency data for the different methods

VI. CONCLUSION

Peer-to-peer communication plays an important role in implementing several of the core functionalities for Energy Management Systems. An important question that designers must address is how to achieve a balance between the amount of network traffic generated as the network scales and the preservation of performance goals with respect to the various algorithms that are deployed at each functional layer of the grid. The choice of communication topology is an important design decision that power system engineers must take into account with regard to that trade-off. In this paper, we introduce an integrated framework for smart grid applications that allows users to deploy and prototype different virtual communication topologies for peer-to-peer smart grid applications and empirically evaluate their performance.

We also propose a new algorithm, Bounded Path Dissemination, which ensures the dissemination of information throughout all participating peers in the network within a specified threshold. It also has dedicated fault-tolerance features to prevent partitioning of the virtual topology if one or more nodes drop out. Experimental evaluations that compare the performance of BPD with other state-of-the-art approaches show improved convergence accuracy and speed for a peer-to-peer application. Similar studies under faulty conditions also show that the algorithm is able to maintain network communication connectivity in the presence of such faults and the Dissemination Efficiency factor is on par with a robust randomized gossip technology. With the integration of more ad hoc participants in a grid such as hybrid electric vehicles, a self-adaptive virtual topology management algorithm such as BPD could potentially enable these ad hoc players to join or drop out dynamically. Thus, it would be interesting to study how the algorithm performs in the presence of such entities.

This paper was mainly used to introduce the algorithm concepts and show it on a working example. In the future, large-scale experiments on more complex networks need to be performed. Comparison between the proposed scheme and some other node linking choices such as connecting two intermediate nodes should also be considered. Techniques for optimizing the topology based on the number of communication links and desired convergence, as well as a time-synchronized implementation of the averaging algorithm are also interesting directions to explore.

References

[1] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid—the new and improved power grid: A survey," *IEEE communications surveys & tutorials*, vol. 14, no. 4, pp. 944–980, 2011.

[2] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[3] L. L. Peterson and B. S. Davie, *Computer networks: a systems approach*. Elsevier, 2007.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 299–314, 2002.

[5] K. Dhara, Y. Guo, M. Kolberg, and X. Wu, "Overview of structured peer-to-peer overlay algorithms," in *Handbook of Peer-to-Peer Networking*. Springer, 2010, pp. 223–256.

[6] N. Shukla, D. Datta, M. Pandey, and S. Srivastava, "Towards software defined low maintenance structured peer-to-peer overlays," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1242–1260, 2021.

[7] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Peer-to-peer membership management for gossip-based protocols," *IEEE transactions on computers*, vol. 52, no. 2, pp. 139–149, 2003.

[8] Z. Genç and Ö. Özkasap, "Peer-to-peer epidemic algorithms for reliable multicasting in ad hoc networks," *World Academy of Science, Engineering and Technology*, vol. 3, 2005.

[9] X. Jin and S.-H. G. Chan, "Unstructured peer-to-peer network architectures," in *Handbook of peer-to-peer networking*. Springer, 2010, pp. 117–142.

[10] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[11] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[12] C. Chen, R. Vitenberg, and H.-A. Jacobsen, "Building fault-tolerant overlays with low node degrees for topic-based publish/subscribe," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[13] S. K. Monga, S. K. Ramachandra, and Y. Simmhan, "Elfstore: A resilient data storage service for federated edge and fog resources," in *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 2019, pp. 336–345.

[14] M. Shojafar, Z. Pooranian, P. G. V. Naranjo, and E. Baccarelli, "Flaps: bandwidth and delay-efficient distributed data searching in fog-supported p2p content delivery networks," *The journal of supercomputing*, vol. 73, no. 12, pp. 5239–5260, 2017.

[15] S. Voulgaris, D. Gavidia, and M. Van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and systems Management*, vol. 13, no. 2, pp. 197–217, 2005.

[16] M. Jelasity and O. Babaoglu, "T-man: Gossip-based overlay topology management," in *International Workshop on Engineering Self-Organising Applications*. Springer, 2005, pp. 1–15.

[17] P. A. Costa, P. Fouto, and J. Leitão, "Overlay networks for edge management," in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2020, pp. 1–10.

[18] S. A. Rashid, L. Audah, M. M. Hamdi, M. S. Abood, and S. Alani, "Reliable and efficient data dissemination scheme in vanet: a review," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 6, pp. 6423–6434, 2020.

[19] S. Marzal, R. González-Medina, R. Salas-Puente, E. Figueres, and G. Garcerá, "A novel locality algorithm and peer-to-peer communication infrastructure for optimizing network performance in smart microgrids," *Energies*, vol. 10, no. 9, p. 1275, 2017.

[20] E. Tebekaemi and D. Wijesekera, "Secure overlay communication and control model for decentralized autonomous control of smart microgrids," *Sustainable Energy, Grids and Networks*, vol. 18, p. 100222, 2019.

[21] L. Orda, O. Gehrke, and H. Bindner, "Applying overlay networks to the smart grid and energy collectives," *Electric Power Systems Research*, vol. 192, p. 106702, 2021.

[22] S. Eisele, I. Mardari, A. Dubey, and G. Karsai, "Riaps: Resilient information architecture platform for decentralized smart systems," in *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2017, pp. 125–132.

[23] P. Ghosh, S. Eisele, A. Dubey, M. Metelko, I. Madari, P. Volgyesi, and G. Karsai, "On the design of fault-tolerance in a decentralized software platform for power systems," in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2019, pp. 52–60.

[24] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 305–319.

[25] R. Jabr, A. Coonick, and B. Cory, "A primal-dual interior point method for optimal power flow dispatching," *IEEE Transactions on Power Systems*, vol. 17, no. 3, pp. 654–662, 2002.

[26] W. I. Bower, D. T. Ton, R. Guttromson, S. F. Glover, J. E. Stamp, D. Bhatnagar, and J. Reilly, "The advanced microgrid. integration and interoperability," Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2014.

[27] IBM, "What is distributed cloud — ibm," https://www.ibm.com/topics/distributed-cloud, (Accessed on 12/19/2021).

[28] G. Coley, "Beaglebone black system reference manual," *Texas Instruments, Dallas*, vol. 5, p. 2013, 2013.

[29] J. Leitao, J. Pereira, and L. Rodrigues, "Epidemic broadcast trees," in *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE, 2007, pp. 301–310.

10