ROBUST IMPLEMENTATION OF ALGORITHMS FOR DISTRIBUTED GENERATION CONTROL OF SMALL-FOOTPRINT POWER SYSTEMS

BY

STANTON T. CADY

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Assistant Professor Alejandro D. Domínguez-García

ABSTRACT

Together with advancements in communication and computer processing technologies, the widespread integration of distributed energy resources (DERs) in the form of renewable energy sources, e.g., wind and solar, will make available new and valuable ancillary services to power systems such as voltage support and frequency regulation. Given the relative size of the resources, however, the provision of these services will require the coordination of several DERs such that their collective capabilities have sufficient impact on a system level. This thesis proposes a method for controlling distributed generation resources (DGRs) without the need for a centralized decision maker. In particular, we discuss a class of iterative algorithms which are capable of coordinating a set of DGRs in order to collectively achieve a predetermined goal. We begin by formulating an unconstrained algorithm which we later extend to account for individual DGR capacity constraints. A convergence analysis of the algorithms is presented, followed by the discussion of a modification that enhances the resiliency of the algorithms when the communication links are imperfect. Next, the development of a hardware testbed comprised of low-complexity devices equipped with wireless transceivers that implements the algorithms is described. We conclude by illustrating the efficacy of the algorithms by utilizing the hardware testbed to control the synchronous generators to regulate the electrical frequency in a small-footprint power system.

To my parents

ACKNOWLEDGMENTS

I would like to begin by thanking my adviser, Assistant Professor Alejandro Domínguez-García, for everything he has done to help me complete this thesis as well as for the support he has provided me throughout my graduate studies at the University of Illinois. In particular, I would like to express my sincere gratitude for the countless hours he spent with me in the lab while I was conducting the experiments presented in this thesis. I look forward to continuing to work with him and I am excited for what the future may hold.

I would also like to thank my friends and family for everything they have helped me achieve. Without their support and encouragement along the way, I would not be where I am today. Mom and Dad, I am truly grateful for everything you have given me and I could not ask for better parents. Thank you for always being there for me and for helping me to pursue my dreams.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Motivated by initiatives such as the US Department of Energy Smart Grid [1], and given advancements in communications and computer processing, electrical energy systems are undergoing radical transformations. In particular, the pursuit of increased efficiency and reliability has led to changes in the ways which power systems are monitored and controlled. Beyond improvements in communication and control, the introduction of distributed energy resources (DERs) in the form of new loads such as plug-in hybrid electric vehicles (PHEVs) and renewable-based electricity generation such as photovoltaic (PV) solar systems has enabled researchers to propose several methods in which DERs can provide ancillary services to power systems [2], [3], [4].

One example is the utilization of inverter-interfaced DERs (e.g., PV systems or motor drives with active rectifier inputs) to provide reactive power support. Although the primary function of these power electronics-based systems is to provide active power, many of them are capable of producing reactive power if appropriately controlled [5]. Another example is utilizing distributed energy storage (e.g. PHEVs or uninterruptible power supplies (UPS)) to control active power for up and down regulation. Such resources could provide energy peak-shaving during hours of high demand and load leveling when demand is low [6].

In order for DERs to provide these ancillary services to electric grids, however, appropriate control and coordination mechanisms need to be developed. One potential control architecture relies on a centralized strategy in which each DER is coordinated through direct communication with a central decision maker. An alternative approach is to remove the central decision maker and coordinate the DERs in a distributed fashion. Using the latter control architecture to solve the *resource coordination* problem as it applies to the control of distributed generation resources (DGRs) in small-footprint power

systems will be the primary focus of this thesis. Specifically, we develop and implement several algorithms that solve the problem. Although the original motivation for this work was to develop algorithms to coordinate DERs for use in power systems, the algorithms and implementations provided could be used to coordinate any multicomponent system of resources.

Given several discrete components that are each capable of providing some resource, the objective of the resource coordination problem is to utilize a communication network to allow these components to exchange information with neighboring devices in order to collectively provide some amount of resource that is known by a leader. It is assumed that the leading component can only communicate with a limited number of other devices in the system and may not necessarily be aware of the total number of components available. The leader initiates a request for resource by dividing the total resource demand equally among all neighboring components; however, a leading component is not required, as a variation of the initialization procedure could be used in which any node could initiate the request for resource. To address component limitations, upper and lower bounds on the amount of resource each component can provide are considered when solving the resource coordination problem in order to find a feasible solution.

In the experimental setup described in this thesis, each component is a small synchronous generator which will be referred to as a distributed generation resource (DGR). Each DGR is outfitted with a wireless transceiver to create a communication network that can be thought of as a stationary, yet unplanned, ad-hoc network. An iterative process is used to exchange information among components such that they collectively meet the generation demand. At the end of the iterative process, the generation output of each DGR is computed based upon the result of the algorithm and the capacity constraints of the respective DGR.

The intention of this thesis is to develop and demonstrate distributed algorithms that are suitable for coordinating DGRs without the need for a centralized controller. Specifically, the purpose of this work is to document the development and application of a hardware testbed that implements the algorithms proposed in [7], [8], [9]. The remainder of the discourse presented herein elaborates and extends the author's previously published work in [5], [10] and is organized as follows.

Chapter 2 begins by providing a model to describe the communication

between DGRs and introduces the notion of distributed generation control. We next formulate a distributed algorithm that serves to iteratively disperse generation demand among a set of DGRs with no limits on the amount of resource they can provide. The *unconstrained algorithm* is extended to account for upper and lower capacity constraints, and it is shown how the result of this algorithm can be used by each DGR to independently determine when the collective capacity of the system has been reached. The *constrained algorithm* is then adapted to create the *robust algorithm* which converges despite imperfect communication links.

Chapter 3 discusses the development of a hardware testbed created to implement the algorithms presented in Chapter 2. The testbed is based upon Arduino Mega microcontroller boards equipped with XBee modules executing software that realizes each of the proposed distributed algorithms. Results are presented which demonstrate the convergence of each algorithm running on the hardware testbed. To conclude, we illustrate a case in which the constrained algorithm is adapted to evenly split demand among all DGRs and demonstrate the ability for each node to independently determine feasibility.

Utilizing the hardware testbed, Chapter 4 discusses a set of experiments in which the robust algorithm with constraints is used to control the generators in a small-footprint power system. A model of the generator is developed which leads to a two-stage control architecture that is used to regulate the frequency in the power system subject to load changes. Results for several experiments are shown, including one in which a spinning reserve is added to the system to demonstrate the ability of the DGRs to independently determine when the collective capacity has been reached.

Chapter 5 provides some concluding remarks and discusses future work.

CHAPTER 2

ALGORITHM FORMULATION

In this chapter we formulate three algorithms that are suitable for controlling a set of distributed resources without relying on a centralized controller. We begin by developing a model to represent the communication network linking resources that will be used to facilitate analysis and development of the algorithms. Next, we formulate and analyze the convergence of an unconstrained algorithm. We then extend the unconstrained algorithm to account for individual capacity constraints. Finally, the constrained algorithm is adapted to be more resilient to imperfect communication links.

2.1 Communication Model

Let $\mathcal{G}$ be a directed graph describing the communication network in system of distributed generation resources (DGRs) capable of exchanging packetized information via wireless links. Define $\mathcal{V} := V(\mathcal{G})$ to be the set of vertices with each vertex corresponding to a DGR and $\mathcal{E} := E(\mathcal{G})$ to be the set of directed edges with each edge corresponding to a communication link between a pair of DGRs. The exchange of information between two DGRs i and j need not be bidirectional; thus, the ordered pair $(i, j) \in \mathcal{E}$ if and only if DGR i can receive information from DGR j. For each DGR $i \in \mathcal{V}$, we define the set of DGRs from which i can receive information to be the in-neighborhood of i, i.e., $\mathcal{N}_i^- := \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$. Similarly, we define the out-neighborhood of i to be the set of DGRs that can receive information from i, i.e., $\mathcal{N}_i^+ := \{j \in \mathcal{V} : (j, i) \in \mathcal{E}\}$, and we denote the cardinality of the out-neighborhood by $\mathcal{D}_i^+ := |\mathcal{N}_i^+|$. We allow all vertices to have self loops, i.e., $(i, i) \in \mathcal{E},\ \forall i \in \mathcal{V}$; thus, each DGR is included in both its own in- and out-neighborhood. For the algorithms formulated in the following sections, it is assumed that the graph $\mathcal{G}$ is strongly connected; that is, for each ordered

pair of vertices i, j there is a path from i to j [11].

2.2 Problem Definition

Consider a set of n DGRs as described by the aforementioned communication model, i.e., $|\mathcal{V}| = n$, and assume that there exists one leader that knows the amount of electric power generation, ρ_e, to be added to or removed from the system in order to operate according to some predetermined criterion, e.g., at an electrical frequency of 60 Hz. Let x_i be the output of DGR i and define $\rho := \sum_{i=1}^{n} x_i$ to be the total generation provided by the set of DGRs. Furthermore, define $\rho_d := \rho + \rho_e$ to be the total system output required to meet the operating criterion and $l := \mathcal{D}^+_{leader}$ to be the out-degree of the leading DGR, with $l \geq 2$ since $\mathcal{G}$ is strongly connected. Given a nonzero mismatch, i.e., $\rho_e \neq 0$, we define **distributed generation control (DGC)** to be the process by which available DGRs are coordinated in order to collectively meet generation demand without a centralized controller. In particular, DGC is a method which allows DGRs to drive the generation mismatch to zero, i.e., $\rho_e \to 0$, in order for the collective generation provided to equal the generation demand. Throughout the remainder of this chapter, we develop three algorithms that can used to implement DGC for small-footprint power systems.

2.3 Unconstrained Algorithm

The case when there are no limitations on the capacity of each DGR is considered first. Despite being unrealistic, the formulation of an unconstrained algorithm will provide the basis for developing an algorithm that can account for upper and lower bounds on individual DGR capacity.

Without constraints, a trivial method for driving the generation mismatch to zero is to have the DGRs in the out-neighborhood of the leader adjust their generation by $\frac{\rho_e}{l}$ while the remaining $n - l$ DGRs maintain a constant output. For the case when the capacity of each DGR is limited, however, this method would be infeasible if the desired operating point lies outside the collective bounds of the l DGRs in the out-neighborhood of the leader. In

order to provide a more adaptable solution, a distributed iterative algorithm is formulated which, after m iterations, divides the total demand among all n DGRs.

2.3.1 Algorithm Description

Each DGR participating in the distributed algorithm maintains an internal state variable that is updated at each iteration. Let $k = 0, 1, \ldots$, index the iterations, and let $\pi_i[k]$ be the value of the internal state variable of DGR i at round k, where $\pi_i[0] = x_i + \rho_e$ if i is the leader, and $\pi_i[0] = x_i$ otherwise. For convenience, we define $\theta[k] := \sum_{i=1}^{n} \pi_i[k],\ \forall k$.

One method that can be used to distribute the generation demand throughout the system is to have each DGR update its state at each iteration to be a linear combination of its current state and the states of the DGRs in its in-neighborhood. That is, DGR i updates the value of its state variable to be

$$\pi_i[k+1] = p_{ii}\pi_i[k] + \sum_{\substack{j \in \mathcal{N}_i^- \\ i \neq j}} p_{ij}\pi_j[k], \tag{2.1}$$

where p_{ii} is the *self-weight* of DGR i and p_{ij} is *outgoing-weight* of DGR j, $\forall i \in \mathcal{V}$, and $\forall j \in \mathcal{N}_i^-, j \neq i$. After performing m iterations DGR i adjusts its output to be $x_i = \pi_i[m]$ and, for the algorithm to be effective, the total generation should meet the demand, that is, $\rho = \rho_d$.

After some analysis, we will see that a carefully chosen set of weights will take advantage of the distributed nature of the system while ensuring that the algorithm meets the aforementioned objective. To find appropriate weights, we first write (2.1) in matrix form as

$$\begin{aligned} \pi[k+1] &= P\pi[k], \\ \pi[0] &= \pi_0, \end{aligned} \tag{2.2}$$

where $\pi_0 = \Big[\pi_1[0], \pi_2[0], \ldots, \pi_i[0], \ldots, \pi_n[0]\Big]^{\mathrm{T}}$, with $\pi_i[0] = x_i + \rho_e$ if i is the

leader and $\pi_i[0] = x_i$ otherwise, and the matrix P is of the form

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1i} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2i} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ p_{i1} & p_{i2} & \cdots & p_{ii} & \cdots & p_{in} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{ni} & \cdots & p_{nn} \end{bmatrix}, \tag{2.3}$$

where $p_{ij} = 0$ if and only if $(i, j) \notin \mathcal{E}$.

In a distributed system where individual components have only local knowledge of the network, component i is limited to choosing its self-weight, $p_{ii},\ \forall i \in \mathcal{V}$, and outgoing-weights, $p_{ji},\ \forall j \in \mathcal{N}_i^+$, which correspond to the columns of P. Furthermore, since the initial states of algorithm (2.1) are chosen such that $\theta[0] = \rho + \rho_e = \rho_d$, and since the objective is for the demand to be distributed among all n DGRs after m iterations, i.e., $\theta[m] = \rho_d$, it is sufficient for each DGR to choose weights such that the sum of internal states remains constant throughout the iterative process. If the weights are chosen in such a way that the matrix P is column stochastic, i.e., each entry is nonnegative and the columns sum to one, we will see that the sum of the entries of the vector $\pi[k]$ will remain constant for all k.

A simple choice that maintains column stochasticity of P is for each DGR to set its self- and outgoing-weights to be the reciprocal of its out-degree, i.e., $p_{ii} = p_{ji} = \frac{1}{\mathcal{D}_i^+},\ \forall i \in \mathcal{V}$ and $\forall j \in \mathcal{N}_i^+$. Thus DGR i will update its state according to

$$\pi_i[k+1] = \sum_{j \in \mathcal{N}_i^-} \frac{1}{\mathcal{D}_j^+} \pi_j[k], \tag{2.4}$$

and adjust its output to be $x_i = \pi_i[m]$ after performing m iterations. Given this choice of weights, it should be noted that the algorithm in (2.4) does not necessarily split the total generation demand evenly.

2.3.2 Convergence Analysis

By rewriting the algorithm in (2.4) in matrix form according to (2.2), we use the characteristics of the matrix P to prove that $\theta[k]$ remains constant at every iteration k. Furthermore, we prove that the algorithm ensures the overall generation demand is met, i.e., $\theta[m] = \rho_d$, and that the solution obtained is unique.

In addition to being column stochastic by design, P is also primitive since the underlying connectivity graph is assumed to be strongly connected and at least one of its diagonal entries is nonzero [12]. Given a column stochastic primitive matrix, the Perron-Frobenius theorem for nonnegative matrices (see, e.g., [12]) states that the matrix will have a unique eigenvalue with largest modulus at $\lambda_1 = 1$.

Let v and w be the right and left eigenvectors of P associated with λ_1 normalized such that $v^{\mathrm{T}}w = 1$. Given that P is column stochastic, all the entries of the vector w must be equal. Without loss of generality, let w be the vector of all ones, i.e., $w = \Big[1, 1, \ldots, 1\Big]^{\mathrm{T}}$, and given that $v^{\mathrm{T}}w = 1$, the entries of v must sum to one. Define $\pi^{ss} = \Big[\pi_1^{ss}, \pi_2^{ss}, \ldots, \pi_i^{ss}, \ldots, \pi_n^{ss}\Big]^{\mathrm{T}}$, where π_i^{ss} is the steady-state solution of (2.4). Then by the Perron-Frobenius theorem, we have that $\lim_{k\to\infty} P^k = vw^{\mathrm{T}}$ and the vector of steady-state solutions is given by

$$\pi^{ss} = vw^{\mathrm{T}}\pi_0 = \left(\sum_{i=1}^{n} \pi_i[0]\right) v. \tag{2.5}$$

Since the entries of v are nonnegative and add up to one and $\sum_{i=1}^{n} \pi_i[0] = \rho_d$, it follows that the entries of the steady-state solution are nonnegative and sum to ρ_d. Although this proof implies that an infinite number of iterations are required to reach the steady-state solution, experimental results have shown that a finite number of iterations are adequate for convergence to a sufficiently accurate solution, thus the proposed distributed algorithm can be used as a practical method for implementing DGC [7].

2.4 Constrained Algorithm

Any physically realizable network comprised of DGRs will necessarily have limits on generation capacity. Upper bounds on generation are the most familiar—the maximum electrical power output of a generator is limited by the available input energy as well as the device ratings—but it may also be necessary to enforce lower bounds due to operational limitations. Thus, to develop an algorithm that is useful in practical systems, the unconstrained algorithm in (2.4) is extended to account for both constraints.

2.4.1 Algorithm Description

Let x_i^{min} and x_i^{max} for $i = 1, 2, \ldots, n$, be the minimum and maximum output of DGR i and define the corresponding capacity vectors as

$$x^{min} = \left[x_1^{min}, x_2^{min}, \ldots, x_n^{min}\right]^{\mathrm{T}}, \tag{2.6}$$

$$x^{max} = \left[x_1^{max}, x_2^{max}, \ldots, x_n^{max}\right]^{\mathrm{T}}, \tag{2.7}$$

respectively. Define the collective lower and upper capacity limits of the DGRs to be $\chi^{min} = \sum_{i=1}^{n} x_i^{min}$, and $\chi^{max} = \sum_{i=1}^{n} x_i^{max}$. As in the unconstrained case, the total amount of generation provided by the system is $\rho = \sum_{i=1}^{n} x_i$ and the overall generation demand is denoted by $\rho_d = \rho + \rho_e$. It is assumed that the collective capacity of the DGRs is sufficient to drive the generation mismatch to zero, i.e., $\chi^{min} \leq \rho + \rho_e \leq \chi^{max}$.

Instead of maintaining a single state variable, DGRs participating in the constrained distributed algorithm maintain two variables, each with different initial conditions that are linear combinations of the capacity constraints. Let $\mu_i[k]$ and $\sigma_i[k]$ be the state variables maintained by DGR i at iteration k, where $\mu_i[0] = \rho_e + x_i - x_i^{min}$ if i is the leader and $\mu_i[0] = -x_i^{min}$ otherwise, and $\sigma_i[0] = x_i^{max} - x_i^{min}$, $\forall i \in \mathcal{V}$. The algorithm given in (2.4) is used to

update the state variables of DGR i as

$$\mu_i[k+1] = \sum_{j \in \mathcal{N}_i^-} \frac{1}{\mathcal{D}_j^+} \mu_j[k], \tag{2.8}$$

$$\sigma_i[k+1] = \sum_{j \in \mathcal{N}_i^-} \frac{1}{\mathcal{D}_j^+} \sigma_j[k]. \tag{2.9}$$

After m iterations, DGR i computes its output to be

$$x_i = x_i^{min} + \frac{\mu_i[m]}{\sigma_i[m]}(x_i^{max} - x_i^{min}), \tag{2.10}$$

and we have that $\rho = \rho_d$ and $x_i^{min} \leq x_i \leq x_i^{max},\ \forall i \in \mathcal{V}$.

2.4.2 Convergence Analysis

To prove that the constrained algorithm coordinates the DGRs to meet the overall demand without violating individual constraints, we first rewrite (2.8) and (2.9) in matrix form as

$$\begin{aligned} \mu[k+1] &= P\mu_0, \\ \sigma[k+1] &= P\sigma_0, \end{aligned} \tag{2.11}$$

with P as defined in the formulation of the unconstrained algorithm and where the initial vectors μ_0 and σ_0 are given as

$$\begin{aligned} \mu_0 &= \Big[\mu_1[0], \mu_2[0], \ldots, \mu_i[0], \ldots, \mu_n[0]\Big]^{\mathrm{T}}, \\ \sigma_0 &= \Big[\sigma_1[0], \sigma_2[0], \ldots, \sigma_i[0], \ldots, \sigma_n[0]\Big]^{\mathrm{T}}, \end{aligned} \tag{2.12}$$

with $\mu_i[0]$ and $\sigma_i[0]$ as defined above.

From the proof of the unconstrained algorithm, it follows that the steady-

state solutions of the iterations in (2.11) are given by

$$\begin{aligned}\mu^{ss} = vw^{\mathrm{T}}\mu_0 &= \left(\sum_{i=1}^{n}(\pi_i[0] - x_i^{min})\right)v \\ &= \left(\rho_d - \sum_{i=1}^{n} x_i^{min}\right)v, \\ \sigma^{ss} = vw^{\mathrm{T}}\sigma_0 &= \left(\sum_{i=1}^{n}(x_i^{max} - x_i^{min})\right)v,\end{aligned} \tag{2.13}$$

where $\pi_i[0] = \rho_e + x_i$ if i is the leader and $\pi_i[0] = x_i$ otherwise. Combining (2.10) and (2.13), the output of DGR i is given as

$$\begin{aligned}x_i &= \lim_{k\to\infty}\left(x_i^{min} + \frac{\mu_i[k]}{\sigma_i[k]}(x_i^{max} - x_i^{min})\right) \\ &= x_i^{min} + \frac{\mu_i^{ss}}{\sigma_i^{ss}}(x_i^{max} - x_i^{min}),\end{aligned} \tag{2.14}$$

where the ratio of the steady-state solutions is defined to be

$$\alpha_i := \frac{\mu_i^{ss}}{\sigma_i^{ss}} = \frac{\rho_d - \sum_{i=1}^{n} x_i^{min}}{\sum_{i=1}^{n}(x_i^{max} - x_i^{min})}. \tag{2.15}$$

After the algorithm has converged, $\alpha_i \in [0,1],\ \forall i \in \mathcal{V}$, if and only if the overall generation demand can be met by the system, i.e., $\chi^{min} \leq \rho_d \leq \chi^{max}$. Thus, if the value of $\alpha_i \notin [0,1]$, DGR i can determine that the collective capacity of the system is insufficient to meet demand.

Similar to the proof of the unconstrained algorithm, this proof implies that an infinite number of iterations are required to converge to the steady-state solution. Examples in the next chapter, however, illustrate that convergence to a sufficiently accurate solution can be reached for a small network of DGRs in as few as 10 iterations.

2.5 Robust Algorithm with Constraints

Throughout the derivation of the previous two algorithms, it was implicitly assumed that the communication links used to exchange information between DGRs were completely reliable. In an uncontrolled environment, however,

conditions such as temperature and humidity as well as obstructions between DGRs can negatively affect link availability. To provide an algorithm that can be useful in systems subject to such non-idealities, the constrained algorithm is extended to be resilient to packet loss.

2.5.1 Communication Model Modifications

Before the algorithm described by (2.8) and (2.9) can be made more robust, the graph modeling the exchange of information between DGRs needs to be modified to account for the possibility that communication links may not be available at every iteration. In this case, the graph is a function of the iteration index k, and is denoted $\mathcal{G}[k]$, where $\mathcal{V} = V(\mathcal{G}[k])$ is independent of k, and $\mathcal{E}[k] = E(\mathcal{G}[k])$ is the set of edges where $(i, j) \in \mathcal{E}[k]$ if DGR i can receive information from DGR j at iteration k. It is assumed that $\mathcal{E}[k] \subseteq \mathcal{E}$, $\forall k \geq 0$, where $\mathcal{E}$ is the set of available edges given completely reliable communication links. Furthermore, it is assumed that each DGR determines the size of its out-neighborhood during an initialization procedure that is perfectly reliable.

If the packets used to exchange information for the distributed algorithm are broadcasted and no acknowledgments are sent, each DGR assumes that all transmitted information is successfully delivered to the intended receiving DGR(s). However, if DGR i attempts to send its weighted values to DGR j at iteration k and $(j, i) \notin \mathcal{E}[k]$, this assumption is invalid and the information intended for DGR j is lost. In order to mitigate the effects of packet loss without increasing the number of packets exchanged at each iteration, we modify the distributed algorithm with constraints to allow the DGRs to collectively meet the overall demand regardless of communication link reliability.

2.5.2 Algorithm Description

One method that can be used to recover information lost due to dropped packets is for each DGR to broadcast the sum of its weighted values up to and including the current iteration k as proposed in [9]. In the case where no packets are lost, the weighted values received from the in-neighbors of a DGR can be inferred at each iteration k, and the proposed method is effectively

the same as the constrained algorithm presented above. If packets are lost, however, the algorithm seamlessly recovers any lost information.

At each iteration k, DGR i broadcasts two values that are linear combinations of its internal state maintained throughout the iterative process. Let $y_i[k]$ and $z_i[k]$ be the values of the internal state maintained by DGR i at iteration k and let $\mu_i[k]$ and $\sigma_i[k]$ be the values broadcasted to all out-neighbors of DGR i at iteration k. The value of $\mu_i[k]$ is simply the sum of $y_i[k]/\mathcal{D}_i^+$ since the iterative process began and is given as

$$\mu_i[k] = \mu_i[k-1] + \frac{1}{\mathcal{D}_i^+} y_i[k] = \sum_{r=0}^{k} \frac{1}{\mathcal{D}_i^+} y_i[r]. \tag{2.16}$$

Similarly, the value of $\sigma_i[k]$ is the sum of $z_i[k]/\mathcal{D}_i^+$ up to and including the current iteration k and is given as

$$\sigma_i[k] = \sigma_i[k-1] + \frac{1}{\mathcal{D}_i^+} z_i[k] = \sum_{r=0}^{k} \frac{1}{\mathcal{D}_i^+} z_i[r]. \tag{2.17}$$

At each iteration, DGR i will update the value of its state variables as

$$\begin{aligned} y_i[k+1] &= \frac{1}{\mathcal{D}_i^+} y_i[k] + \sum_{\substack{j \in \mathcal{N}_i^- \\ i \neq j}} (\nu_{ij}[k] - \nu_{ij}[k-1]), \\ z_i[k+1] &= \frac{1}{\mathcal{D}_i^+} z_i[k] + \sum_{\substack{j \in \mathcal{N}_i^- \\ i \neq j}} (\tau_{ij}[k] - \tau_{ij}[k-1]), \end{aligned} \tag{2.18}$$

where the values of $\nu_{ij}[k]$ and $\tau_{ij}[k]$ depend on the successful receipt of a packet from DGR j during iteration k and are given as

$$\begin{aligned} \nu_{ij}[k] &= \begin{cases} \mu_j[k], & \text{if } (i,j) \in \mathcal{E}[k], \\ \nu_{ij}[k-1], & \text{if } (i,j) \notin \mathcal{E}[k], \end{cases} \\ \tau_{ij}[k] &= \begin{cases} \sigma_j[k], & \text{if } (i,j) \in \mathcal{E}[k], \\ \tau_{ij}[k-1], & \text{if } (i,j) \notin \mathcal{E}[k]. \end{cases} \end{aligned} \tag{2.19}$$

The initial values of the state variables are $y_i[0] = \rho_e + x_i - x_i^{min}$ if i is the leader and $y_i[0] = -x_i^{min}$ otherwise, and $z_i[0] = x_i^{max} - x_i^{min} > 0$; whereas the initial conditions for the broadcasted values are set to $\mu_i[0] = y_i[0]/\mathcal{D}_i^+$

and $\sigma_i[0] = z_i[0]/\mathcal{D}_i^+$. After m iterations, DGR i computes its output as

$$x_i = x_i^{min} + \frac{y_i[m]}{z_i[m]}(x_i^{max} - x_i^{min}), \tag{2.20}$$

and we have that $\rho = \rho_d$ and $x_i^{min} \leq x_i \leq x_i^{max}$, $\forall i \in \mathcal{V}$ (for a proof see [9]). Similar to the basic algorithm with constraints, we define the ratio of the values of the internal states after m iterations as found by DGR i to be

$$\alpha_i := \frac{y_i[m]}{z_i[m]}. \tag{2.21}$$

Thus, DGRs participating in the robust algorithm with constraints can independently determine if the collective capacity of the system is sufficient to meet the overall generation demand if $\alpha_i \geq 0$ and $\alpha_i \leq 1$

In order to compute the values in (2.19), each DGR needs to keep the most recent set of values received from the DGRs in its in-neighborhood and thus needs to know the source of all packets received. To accommodate this, each DGR creates a list of addresses corresponding to the DGRs in its in-neighborhood during initialization that will remain unchanged throughout the iterative process. Furthermore, when DGR i broadcasts its values $\mu_i[k]$ and $\sigma_i[k]$, it also includes its address in the packet.

CHAPTER 3

HARDWARE IMPLEMENTATION

This chapter describes a hardware testbed created to implement the algorithms formulated in the previous chapter. The testbed is centered around nodes with embedded processors capable of wirelessly exchanging information with other nearby nodes. The nodes are designed to be independent of the DGRs, enabling the testbed to be portable to various applications. Throughout the remainder of the chapter, the hardware chosen is described while the software used to implement the algorithms is explained. In the final section, experimental results for the three algorithms are presented.

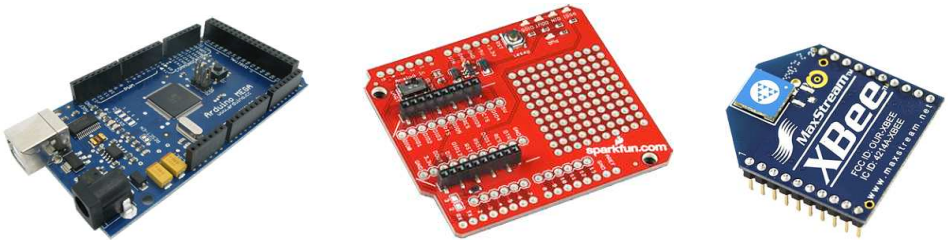
3.1 Communication Hardware Platform

In this section, we describe the hardware chosen to create the testbed and provide a brief overview of the software used to exchange information between devices and to implement the algorithms.

3.1.1 Node Hardware

The hardware testbed is based around Arduino, an open-source electronics prototyping platform. Arduino was chosen for its flexibility and ease of use as well as for the numerous software libraries and extension circuit boards, called shields, that are available [13].

Each node in the testbed contains an Arduino Mega 2560 [14] microcontroller (μC) board which is based on the AVR ATmega2560 [15]. The Arduino board, shown in Fig. 3.1a, provides access to the digital I/O and analog input ports on the μC and contains a USB connection for flashing and powering the device. The ATmega2560 μC has 256 kB of flash memory and a clock speed of 16 MHz as well as four universal asynchronous re-



(a) Arduino Mega 2560 (b) SparkFun XBee Shield (c) MaxStream XBee Module

Figure 3.1: Hardware

ceiver/transmitter (UART) ports that enable it to communicate with several devices independently.

In order to enable the nodes to exchange information wirelessly, each Arduino Mega is connected to a MaxStream XB24-DMCIT-250 revB XBee module [16] via a SparkFun Electronics XBee shield [17]. The XBee shield, shown in Fig. 3.1b, serves as an interface between the Arduino board and the XBee module while providing the requisite 3.3 V power supply via a voltage regulator. Furthermore, each shield is modified to allow the Arduino board to communicate with a computer via USB and the XBee independently. The XBee, shown in Fig. 3.1c, is an embedded RF module operating at 2.4 GHz that utilizes a built-in chip antenna and requires only a single connection to the μC via one of the UART ports.

3.1.2 Software Setup

To facilitate the exchange of values for the distributed algorithms, each XBee module is put into API mode (AP=2 with escapes), and the three-layered communication protocol stack shown in Fig. 3.2 is implemented. The lowest layer of the stack is based on the ZigBee (IEEE 802.15.4) protocol [18], and is contained entirely on the XBee modules. The middle layer consists of a modified version of the xbee-arduino API [19]. The modifications allow wired communication between the nodes and a computer to continue uninterrupted while the nodes exchange information wirelessly. Additionally, the API was altered to enable incoming and outgoing messages to be time-stamped immediately upon receipt and just before being sent to increase the accuracy of the time synchronization mechanism discussed in the next section. The

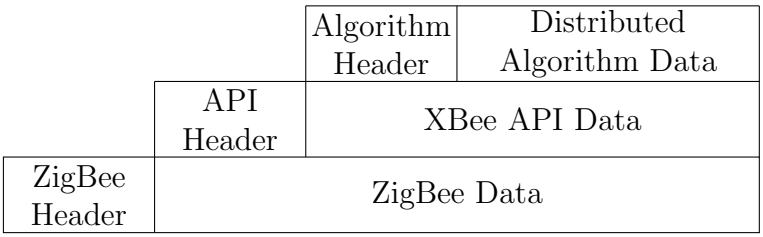


Figure 3.2: Communication protocol stack

header of the top layer contains information about the distributed algorithm being used while the payload holds the values exchanged during the iterative process.

All of the software created for implementing the distributed algorithms on the nodes is written in C++. Furthermore, an object-oriented approach is taken where possible to encourage code reuse and to simplify the initialization of the algorithms. The Arduino software environment is used to program the μCs and for monitoring the serial port to gather data.

3.2 Distributed Algorithm Implementation

In order to take advantage of the wireless medium used for communication among nodes, all of the packets used to exchange values are broadcasted; that is, packets are not addressed to a particular node. Furthermore, to minimize network traffic, no acknowledgements are sent upon successful receipt of packets. To create a partially connected network despite the close proximity of the nodes during testing, each μC is programmed to only accept messages received from nodes in its in-neighborhood. In a more realistic setup, however, the testbed could be adapted to allow the availability of links between nodes to be based upon signal strength.

Throughout the formulation of the algorithms in the previous chapter, we assumed that all participating DGRs update the value of their state variables in unison; i.e., DGR i updates its state at iteration k at the same time DGR j updates its state, $\forall i, j \in \mathcal{V}$. Without a common time reference and with no acknowledgements, however, it is possible for the DGRs to update their states at different times which could cause the DGRs to converge to the wrong solution or possibly diverge. Thus, to ensure convergence to the correct solution, all nodes are synchronized to a common reference before

initializing the distributed algorithm.

The synchronization mechanism used in the hardware testbed is based on the hierarchy referencing time synchronization (HRTS) protocol proposed in [20]. This protocol requires very little overhead and is capable of synchronizing the clocks of several nodes to the clock of a reference node using only three packets. As mentioned previously, given the close proximity of the nodes during testing, the graph representing the communication structure in the network is completely connected; thus, in order to simplify the process, no communication restrictions are placed on the nodes during synchronization.

To initiate the time synchronization process, the reference node (e.g. the leader node) broadcasts a `sync_begin` packet at time t_1, specifying a target node from its out-neighborhood chosen randomly. The target node then responds using a unicast packet that contains the time the `sync_begin` packet was received, t_2, and the time the response packet was sent, t_3. All other nodes interested in synchronizing to the reference node record the local time at which the `sync_begin` packet was received, t_2', but do not respond. At time t_4, the reference node receives the response packet from the target node and thus owns all of the timestamps required to determine the offset between its local clock and the local clock of the target node. Assuming negligible propagation delay, the reference node computes the offset as

$$d = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{3.1}$$

and broadcasts it in a final packet also containing t_2. At this point, the target node can complete the synchronization process by adjusting its clock to be $T = t + d$, where t is the local clock reading before synchronization. The timestamp t_2 included in the final packet from the reference node is used by all other nodes to estimate the offset between their local clocks and the local clock of the target node as $d' = t_2 - t_2'$. Using this estimate, the remaining nodes can now adjust their clocks to be $T = t + d + d'$, where t is the local clock of the respective node before synchronization. In the testbed, rather than adjust the clocks of synchronized nodes, a function extending the low-level clock `timer0_millis` is used which adds the offset found using HRTS to the local time, providing a clock that is common throughout the network.

As mentioned above, the computation of the clock offset between nodes in the HRTS protocol assumes there is negligible communication delay. Thus

Procedure 1: General distributed algorithm

Input: iteration period, number of iterations, initial command, *(optional)* constraints
Output: new resource command

```
begin
    generate random transmit time;
    foreach iteration do
        begin timer;
        while timer < iteration period do
            look for incoming packet;
            if packet available then
                if sender ∈ in-neighborhood then
                    store incoming value(s);
            if transmit time = time elapsed then
                broadcast current value(s);
        compute next value;
    compute final command;
```

packets exchanged during the synchronization process should be time stamped at the lowest possible protocol to reduce error resulting from data propagating up the protocol stack. In the testbed, however, the bottom layer of the stack cannot be modified, so all time stamps are generated at the middle protocol layer. Given this configuration, the delay present in the system results in a worst case clock error on the order of 10 ms. To mitigate the effects of this error on the distributed algorithms, the nodes are restricted from transmitting data for a period of time which exceeds the clock error during the beginning and end of each iteration.

After synchronizing the clocks of all of the nodes in the network, the distributed algorithm begins. The number of iterations, m, and the period of each iteration is known by all of the nodes *a priori* to ensure that synchronism is maintained throughout the iterative process. The function in Procedure 1 outlines the basic routine executed at each node participating in the distributed algorithm. The required arguments of this function are the initial value, the iteration period and the number of iterations to be performed while resource constraints can be passed as optional arguments. Although the ZigBee protocol seeks to minimize packet collisions at the lowest layer

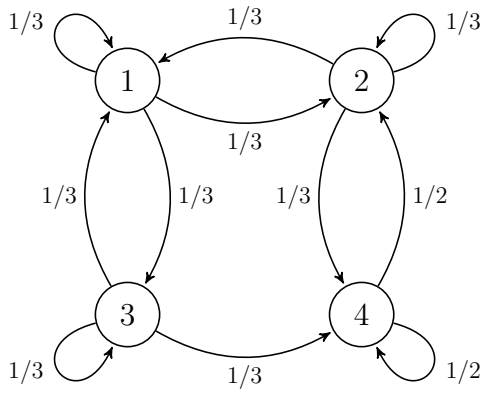


Figure 3.3: Graph of 4-node network

of the protocol stack, the nodes attempt to avoid collisions by broadcasting their values at randomly chosen times within the iteration period.

3.3 Experimental Results

In this section, experimental results generated from the unconstrained, constrained and robust algorithms as implemented on the hardware testbed are presented. Throughout this section, the inputs and outputs of the algorithms are unit-less and the nodes are not controlling a DGR. Despite this, we use the terms node and DGR interchangeably.

3.3.1 Unconstrained Algorithm

The hardware testbed is used to implement the unconstrained algorithm on the 4-node network depicted by the graph in Fig. 3.3. For this experiment, the leader node is indexed by 1 and the generation mismatch is $\rho_e = \frac{1}{2}$. Initially, $x_2 = \frac{1}{2}$ and $x_1, x_3, x_4 = 0$, thus, $\pi_1[0] = \rho_e + x_1 = \frac{1}{2}$, $\pi_2[0] = \frac{1}{2}$, $\pi_3[0], \pi_4[0] = 0$ and the nodes update their values according to algorithm

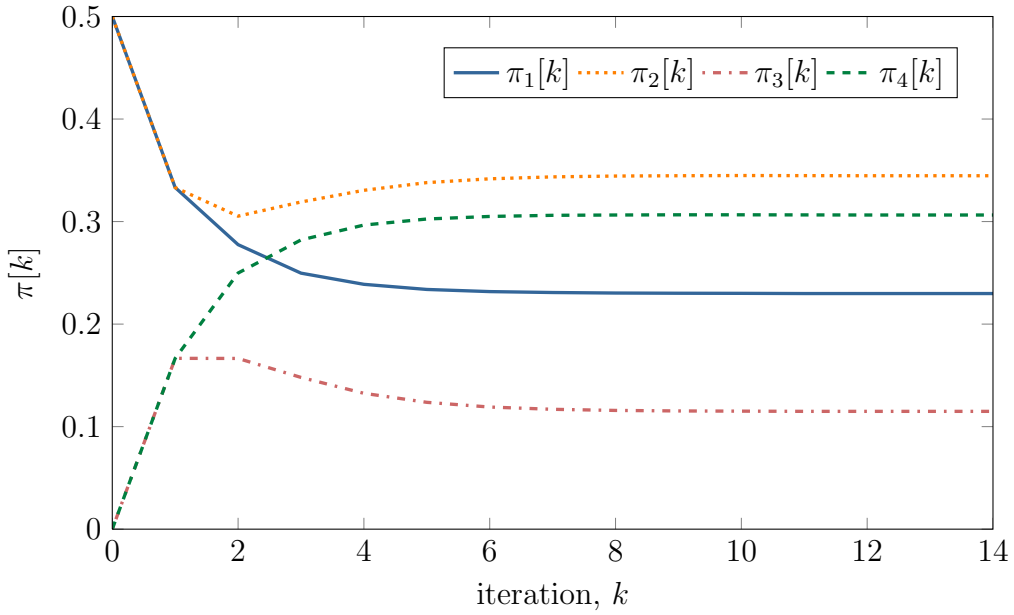


Figure 3.4: Unconstrained results

(2.4) as

$$\begin{aligned}
\pi_1[k+1] &= \frac{1}{3}(\pi_1[k] + \pi_2[k] + \pi_3[k]), \\
\pi_2[k+1] &= \frac{1}{3}(\pi_1[k] + \pi_2[k]) + \frac{1}{2}\pi_4[k], \\
\pi_3[k+1] &= \frac{1}{3}(\pi_1[k] + \pi_3[k]), \\
\pi_4[k+1] &= \frac{1}{3}(\pi_2[k] + \pi_3[k]) + \frac{1}{2}\pi_4[k].
\end{aligned} \tag{3.2}$$

Equation (3.2) can be written in matrix form according to (2.2), where $\pi[0] = \left[\frac{1}{2}, \frac{1}{2}, 0, 0\right]^{\mathrm{T}}$ and

$$P = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 1/2 \\ 1/3 & 0 & 1/3 & 0 \\ 0 & 1/3 & 1/3 & 1/2 \end{bmatrix}. \tag{3.3}$$

The evolution of the values of $\pi[k]$ computed at each node is plotted in Fig. 3.4. From the plot, it can be seen that the nodes converge to their steady-state values in approximately 8 iterations. For this experiment, the nodes are programmed to perform 14 iterations; thus the vector of final

values corresponding to the amount of generation each DGR should provide is given as $x = \pi[14] = \Big[0.230, 0.345, 0.119, 0.306\Big]^{\mathrm{T}}$. Due to the directed edge between nodes 3 and 4, this is an example where the DGRs do not equally split the total generation demand among themselves.

3.3.2 Constrained Algorithm Results

Similar to the unconstrained example, the 4-node network represented by the graph in Fig. 3.3 is constructed using the hardware testbed to evaluate the convergence of the constrained algorithm. To illustrate the effects of link availability on convergence, we present a case in which the constrained algorithm converges to the correct solution and a case in which it does not.

Correct Convergence

In order to allow sufficient time for nodes to exchange information and compute their next value, a period of 500 ms is apportioned for each iteration. Furthermore, nodes are restricted from transmitting during the first and last 50 ms of each iteration to account for any synchronization errors. As in the unconstrained example, the leader node is indexed by 1 and the generation mismatch is chosen to be $\rho_e = \frac{1}{2}$, while $x_2 = \frac{1}{2}$, and $x_1, x_3, x_4 = 0$. The lower and upper constraints are given by the vectors $x^{min} = \Big[0.1, 0.05, 0.12, 0\Big]^{\mathrm{T}}$ and $x^{max} = \Big[0.35, 0.3, 0.26, 0.24\Big]^{\mathrm{T}}$, respectively. To ensure a feasible solution, the individual limits are chosen such that the total generation demanded from the DGRs lies within the bounds of the collective constraints, that is, $\chi^{min} = 0.27 < \rho_d < \chi^{max} = 1.15$.

Given the initial outputs of each DGR and the individual constraints, the initial values for (2.8) and (2.9) are $\mu_1[0] = 0.4, \mu_2[0] = 0.45, \mu_3[0] = -0.12, \mu_4[0] = 0$ and $\sigma_1[0] = \sigma_2[0] = 0.25, \sigma_3[0] = 0.14, \sigma_4[0] = 0.24$ and the

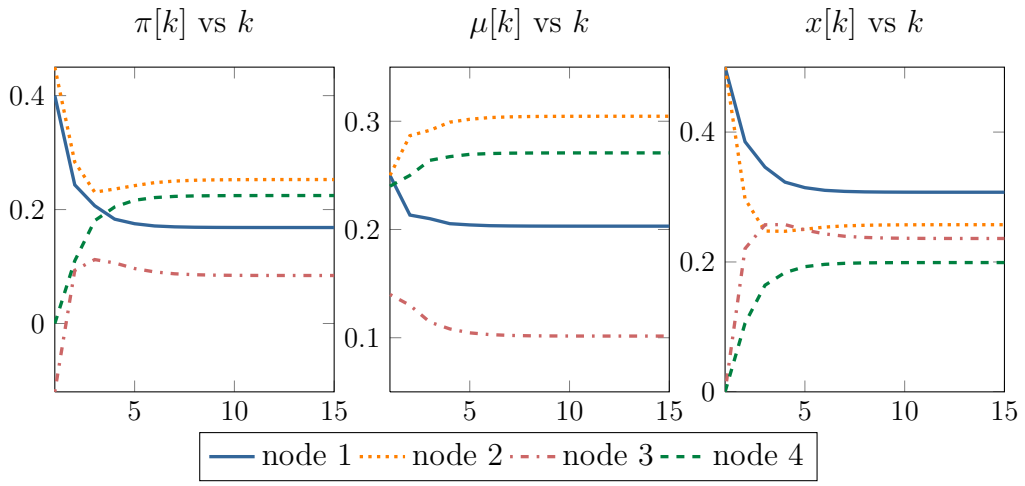


Figure 3.5: Evolution of the distributed algorithm for a network of 4 nodes with constraints

constrained algorithm written in matrix form is given as

$$\begin{aligned} \mu[k+1] &= P\mu[k] \\ \mu[0] &= \begin{bmatrix} 0.4, 0.45, -0.12, 0 \end{bmatrix}^{\mathrm{T}} \\ \sigma[k+1] &= P\sigma[k] \\ \sigma[0] &= \begin{bmatrix} 0.25, 0.25, 0.14, 0.24 \end{bmatrix}^{\mathrm{T}}, \end{aligned} \tag{3.4}$$

where P is the matrix given in (3.3).

The evolution of the constrained distributed algorithm is shown in Fig 3.5. Although each node i is not required to compute x_i until the iterative process is complete, it is useful to illustrate the evolution of the system. Thus the values of $\mu_i[k]$, $\sigma_i[k]$ and $x_i[k]$ for $i = 1, 2, 3, 4$ are shown in the figure. The plots show that the nodes reach their steady-state values in approximately 8 iterations, which given an iteration period of 500ms, requires around 4 seconds. As in the unconstrained case, the nodes are programmed to perform 14 iterations; thus, the nodes compute the amount of generation each DGR should provide according to (2.10) with $m = 14$ and we have that $x = \begin{bmatrix} 0.307, 0.257, 0.237, 0.199 \end{bmatrix}^{\mathrm{T}}$. If we sum the generation provided by all the DGRs in the system, we see that the collective output meets the overall demand, i.e., $\sum_{i=1}^{4} x_i = 1 = \rho_d$, while none of the individual constraints are exceeded.

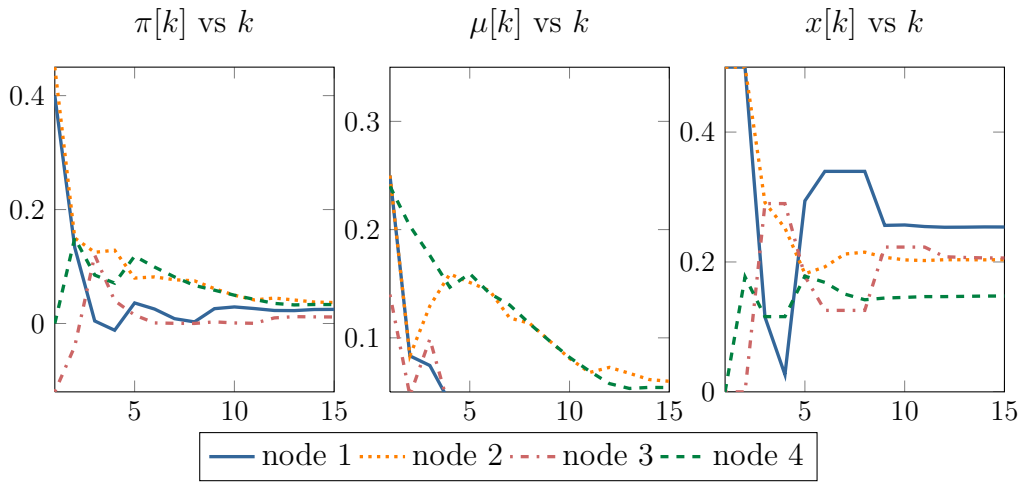


Figure 3.6: Incorrect evolution of the distributed algorithm for a network of 4 nodes with constraints

Incorrect Convergence

The iteration period was chosen conservatively in the previous example to reduce the probability of packet collisions resulting from nodes broadcasting their values concurrently. Moreover, nodes were restricted from transmitting information during the first and last 50 ms of each iteration to ensure that the algorithm would converge correctly despite synchronization error. To illustrate the sensitivity to these parameters, the 4-node network is tested again using a significantly smaller iteration period of 50 ms with no restrictions on broadcast time.

Using the same initial conditions as in the previous example, the evolution of the values maintained by the nodes is plotted in Fig. 3.6. From these plots it is evident that the loss of packets induced by reducing the iteration period effectively removes the ability of the algorithm to preserve the sum of the values maintained by the nodes, causing $\mu[k]$ and $\sigma[k]$ to quickly converge to zero. Although the values exchanged by the nodes approach zero, the figure illustrates that the value of $x[k]$, computed as a function of the ratio of $\mu[k]$ and $\sigma[k]$, tends toward a nonzero steady-state solution. Running the algorithm for 99 iterations (only the first 15 are shown in the figure) results in a steady-state solution of $x = \left[0.254, 0.204, 0.206, 0.147\right]^{\mathrm{T}}$. The total generation provided by the DGRs is given as $\sum_{i=1}^{4} x_i = 0.811 \neq \rho_d = 1$. Thus there is a mismatch between the collective amount of generation supplied and the total generation demand, and the algorithm is ineffective.

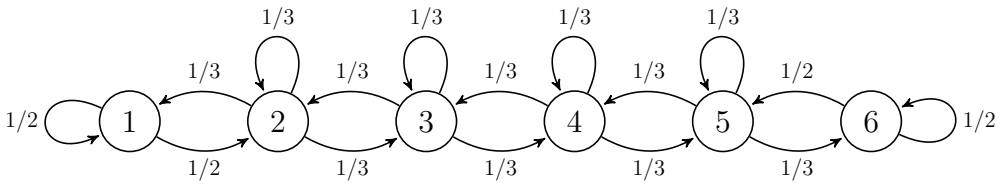


Figure 3.7: Graph of 6-node network

3.3.3 Robust Algorithm with Constraints Results

The 6-node network represented by the graph in Fig. 3.7 is created using the hardware testbed to evaluate the robust algorithm with constraints. In order to induce dropped packets, the iteration period is reduced to 40 ms and no restrictions are placed on broadcast time.

For this experiment, node 1 is selected to be the leader. The generation mismatch is chosen to be $\rho_e = \frac{1}{2}$ and initially, $x_1 = \frac{1}{2}$ and $x_2, x_3, x_4, x_5, x_6 = 0$. The minimum and maximum amount of generation each DGR can provide are given, respectively, by

$$x^{min} = \Big[0.02, 0.1, 0.05, 0.08, 0.12, 0\Big]^{\mathrm{T}},$$
$$x^{max} = \Big[0.146, 0.208, 0.193, 0.167, 0.229, 0.159\Big]^{\mathrm{T}}.$$

The collective lower and upper bounds are chosen to ensure the system is capable of meeting the overall generation demand, i.e., $\chi^{min} = 0.37 < \rho_d < \chi^{max} = 1.102$.

Using the initial generation output and the constraints of each DGR, the initial values of the internal states are given by the vectors

$$y[0] = \Big[0.98, -0.1, -0.05, -0.08, -0.12, 0\Big]^{\mathrm{T}},$$
$$z[0] = \Big[0.126, 0.108, 0.143, 0.087, 0.109, 0.159\Big]^{\mathrm{T}}.$$

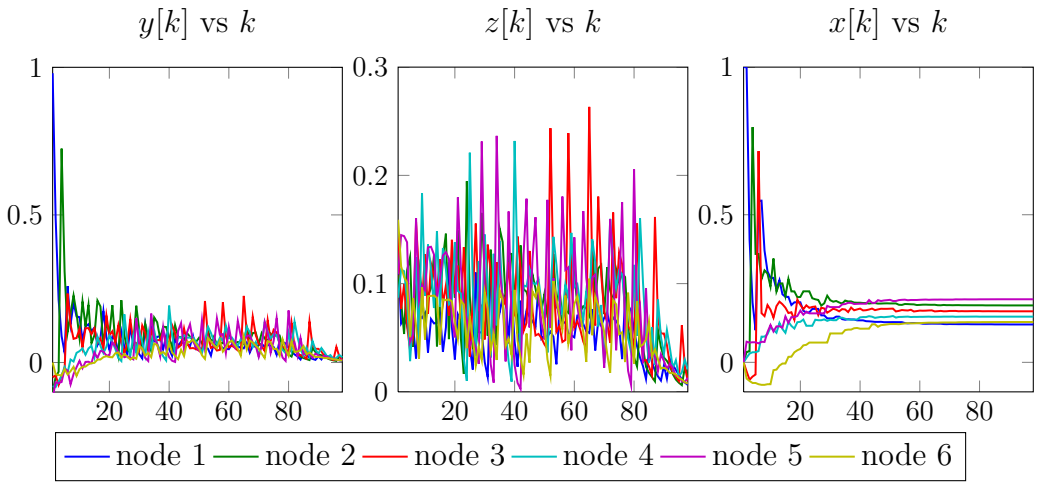


Figure 3.8: Evolution of robust constrained algorithm

Furthermore, the matrix of weights used by the nodes is given by

$$P = \begin{bmatrix} 1/2 & 1/3 & 0 & 0 & 0 & 0 \\ 1/2 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/2 \\ 0 & 0 & 0 & 0 & 1/3 & 1/2 \end{bmatrix}.$$

The evolution of the internal states and the output of each DGR computed at each node running the robust algorithm is shown in Fig. 3.8. The plots of the internal states $y[k]$ and $z[k]$ show erratic behavior that does not appear to reach steady-state. Despite this, $x[k]$ converges to a steady-state solution that meets the overall generation demand. After running 99 iterations, the generation output of each DGR is computed and given by the vector

$$x = \Big[0.128, 0.193, 0.173, 0.155, 0.214, 0.137\Big]^{\mathrm{T}}.$$

If we sum the total amount of generation provided by the DGRs, we see that $\rho = \rho_d$, while none of the individual resource constraints are violated.

3.3.4 Determining Feasibility

As mentioned in Section 2.4.2, each node can independently determine if the collective capacity of the available DGRs is sufficient to meet the overall demand for generation. Specifically, after performing the specified number of iterations and computing α_i according to (2.15), each node can determine if the demand for generation is outside the collective bounds of the DGRs if $\alpha_i > 1$ or $\alpha_i < 0$. By taking advantage of this property, it is possible, for instance, to designate a subset of DGRs as reserves which can participate in the distributed algorithm with artificially restricted limits until determining that the capacity of the remaining DGRs has been exceeded. To illustrate the ability of the individual nodes to determine feasibility, we show results for a case in which the resource demand is within the collective limit of the DGRs and one in which it is not. In both cases, the robust algorithm with constraints is used to implement the 4-node network depicted by the graph in Fig. 3.3 and the total demand for generation is chosen to be $\rho_d = 1$.

We first demonstrate the case in which the generation demand is within the collective bounds of the DGRs. For this experiment, the leader node is indexed by 1, the generation mismatch is chosen to be $\rho_e = \frac{1}{2}$, and, initially, $x_2 = \frac{1}{2}$, and $x_1, x_3, x_4 = 0$. Let the minimum and maximum capacities of the nodes be given respectively by $x^{min} = \Big[0.15, 0, 0.15, 0.1\Big]^{\mathrm{T}}$ and $x^{max} = \Big[0.3, 0.15, 0.4, 0.25\Big]^{\mathrm{T}}$, such that $\chi^{min} = 0.4 \leq \rho_d \leq \chi^{max} = 1.1$. Given the generation mismatch and the capacity of the DGRs, the vectors of initial states are given as $y[0] = \Big[0.35, 0.5, -0.15, -0.1\Big]^{\mathrm{T}}$, and $z[0] = \Big[0.15, 0.15, 0.25, 0.15\Big]^{\mathrm{T}}$.

The evolution of $\alpha_i[k]$ for $j = 1, 2, 3, 4$ over 25 iterations is shown in Fig. 3.9. From the figure, we see that after approximately 15 iterations, all nodes have converged to a solution in which $\alpha = 0.857$. Thus, the nodes determine the solution is feasible and compute the amount of generation each DGR should provide according to (2.20), and we have that $x = \Big[0.279, 0.129, 0.364, 0.228\Big]^{\mathrm{T}}$.

We now demonstrate a case in which the collective capacity of the DGRs is insufficient to meet the total demand for generation. Let the generation mismatch and the initial output of each generator be the same as in the previous case but adjust the minimum and maximum capacity of the nodes to be given

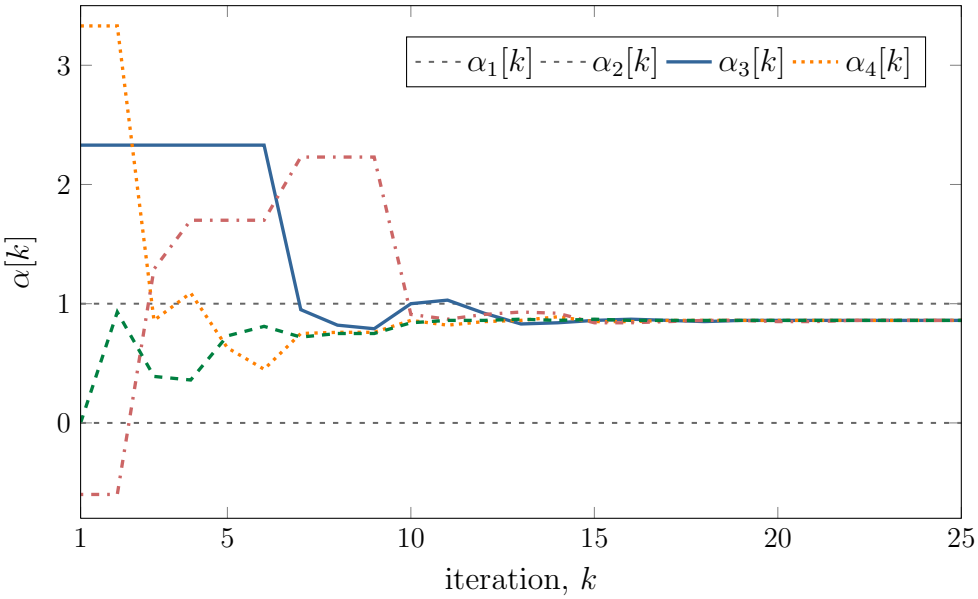


Figure 3.9: Evolution of $\alpha[k]$ for a 4-node system with feasible solution

respectively by $x^{min} = \begin{bmatrix}0.1, 0, 0.1, 0.1\end{bmatrix}^{\mathrm{T}}$ and $x^{max} = \begin{bmatrix}0.25, 0.15, 0.3, 0.25\end{bmatrix}^{\mathrm{T}}$, such that $\chi^{min} = 0.3$ and $\chi^{max} = 0.95 < \rho_d$. Thus, the vectors of initial states are given as $y[0] = \begin{bmatrix}0.4, 0.5, -0.1, -0.1\end{bmatrix}^{\mathrm{T}}$ and $z[0] = \begin{bmatrix}0.15, 0.15, 0.2, 0.1\end{bmatrix}^{\mathrm{T}}$.

The evolution of $\alpha_i[k]$ for the four nodes over 25 iterations is shown in Fig. 3.10. From this figure, we see that after approximately 15 iterations, all nodes have converged to a solution in which $\alpha = 1.167$. Thus, the nodes determine that the solution is infeasible and they cannot adjust their generation output beyond their maximum capacities.

3.3.5 Even Splitting Algorithm

In the previous examples demonstrating algorithms that account for constraints, the output of each DGR was computed such that the generation demand was distributed fairly among all DGRs in the system. Specifically, as illustrated by (2.15), after the algorithm has converged, each DGR i determines its generation output based upon its constraints and α_i. Since α_i is the ratio of the total demand to the collective capacity of the system, the output of each DGR is chosen to be proportional to the overall loading in the system relative to its constraints. In the absence of constraints, however, the algorithm can be adapted such that the total demand for generation is

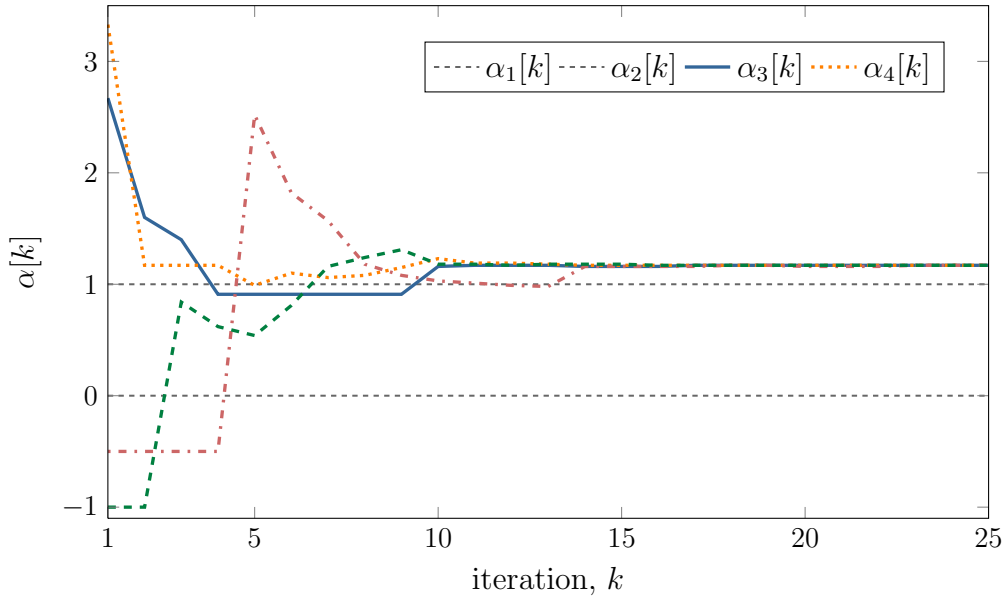


Figure 3.10: Evolution of $\alpha[k]$ for a 4-node system with infeasible solution

evenly divided among all DGRs in the system, i.e., $x_i = \rho_d/n,\ \forall i$.

To demonstrate a case when the nodes split the overall demand evenly, the 7-node network depicted by the graph in Fig. 3.11 is created using the hardware testbed and the robust algorithm with constraints with $x_i^{min} = 0$ and $x_i^{max} = 1,\ i = 1, \ldots, 7$. Similar to the previous examples, the leader is indexed by 1 and the generation mismatch is chosen to be $\rho_e = \frac{1}{2}$. Initially, $x = \Big[0.2, 0.1, 0.05, 0.15, 0.25, 0.35, 0.5\Big]^{\mathrm{T}}$; thus, the total generation demand is $\rho_d = 2.1$ and the vectors of initial states are given as

$$y[0] = \Big[0.7, 0.1, 0.05, 0.15, 0.25, 0.35, 0.5\Big]^{\mathrm{T}},$$
$$z[0] = \Big[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0\Big]^{\mathrm{T}}.$$

Given the edges in the graph representing the communication network, the

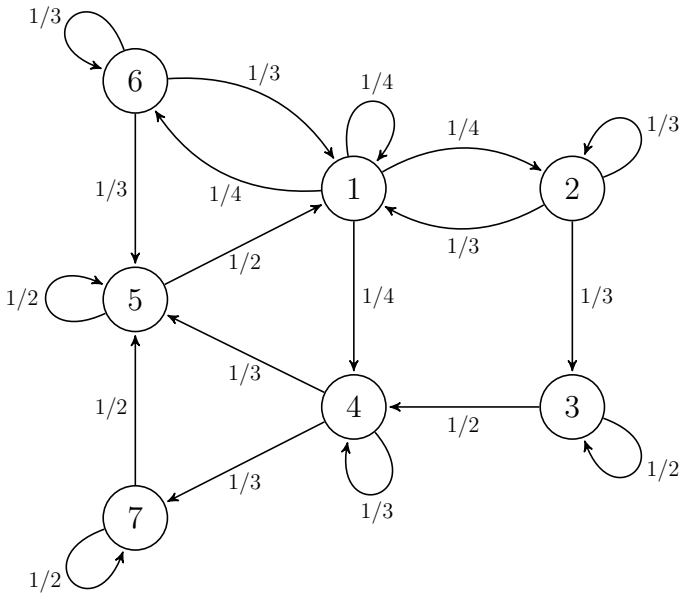


Figure 3.11: Graph of 7-node network

matrix of weights is

$$P = \begin{bmatrix} 1/4 & 1/3 & 0 & 0 & 1/2 & 1/3 & 0 \\ 1/4 & 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 1/2 & 0 & 0 & 0 & 0 \\ 1/4 & 0 & 1/2 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 1/2 & 1/3 & 1/2 \\ 1/4 & 0 & 0 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 0 & 0 & 1/2 \end{bmatrix}.$$

The evolution of $x[k]$ for the seven nodes over 35 iterations is shown in Fig. 3.12. From the figure, it can be seen that the nodes converge to a solution after approximately 30 iterations. As expected, all DGRs split the total demand evenly and thus $x_i = 0.3 = \rho_d/7,\ i = 1, \ldots, 7$.

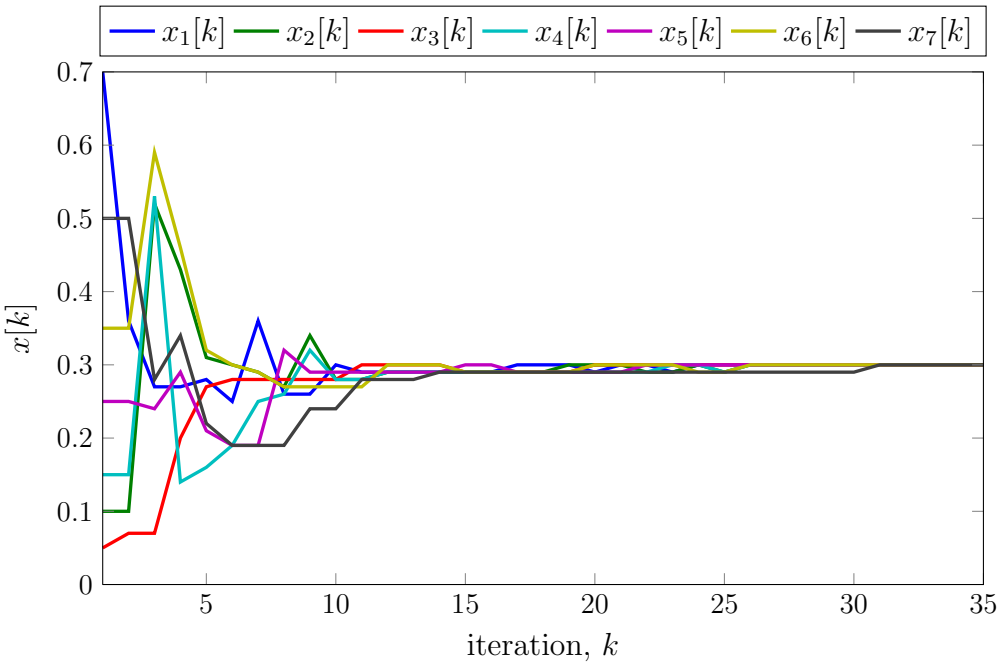


Figure 3.12: Evolution of $x[k]$ for 7-node system with even splitting

CHAPTER 4

APPLICATION TO DISTRIBUTED GENERATION CONTROL OF SMALL-FOOTPRINT POWER SYSTEMS

In this chapter, we utilize the distributed algorithms and the hardware testbed presented in Chapters 2 and 3, respectively, to control the generators in a small-footprint power system in order to regulate the electrical frequency. A brief overview of the model used to describe the synchronous generators in the system is given and the interconnection between the generators and loads is described. Results are provided for several cases to illustrate the efficacy of the algorithms and the testbed for distributed generation control.

4.1 Synchronous Generator Model

When implemented on the hardware testbed, the algorithms developed in Chapter 2 can enable a set of DGRs to be coordinated to meet generation demand without relying on a centralized controller. To be useful in a small-footprint power system, however, the algorithms must be part of an overall control strategy designed to account for the characteristics of the individual DGRs as well as the overall system. The DGRs in the experimental setup discussed in the next section are small synchronous generators; thus, we provide a state-space model that is used to develop a suitable control architecture.

Each synchronous generator can be modeled as a rotating mass connected at the shaft to a prime mover as shown in Fig. 4.1. As the figure illustrates, the generator imposes a torque of electrical origin, T_e, which opposes the mechanical torque, T_m, supplied by the prime mover. Thus the mechanical torque acts to increase the rotational speed while the electrical torque acts to slow it down. Given this relationship and assuming no losses, the rotational speed of the shaft, ω, will be constant only when the magnitude of T_m and T_e are equal. If the electrical load is increased so that T_e is larger than T_m, the entire rotating system will begin to slow down. Similarly, if the electrical

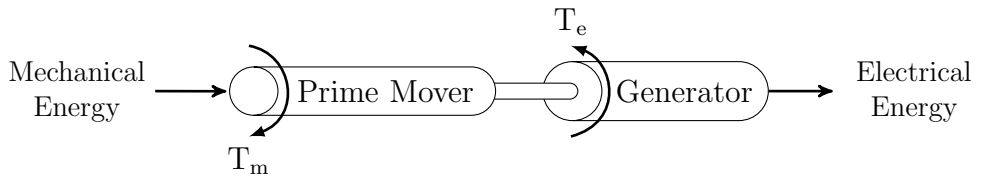


Figure 4.1: Mechanical and electrical torques in generator

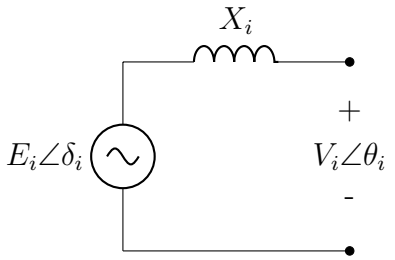


Figure 4.2: Equivalent circuit for synchronous machine model

load is decreased, the speed of the rotating system will increase [21].

Using the above-described relationship between the mechanical torque supplied by the prime mover and the electrical torque imposed by the generator, we use a two-state model—the so-called *classical model* (see, e.g., [22])—to describe the dynamics of the synchronous generators. For each synchronous machine, i, let δ_i denote the angle of the rotor (with respect to the synchronous reference rotating at ω_s [rad/s]) in electrical radians, ω_i denote the angular velocity of the rotor in electrical radians per second and P_i^m [pu] denote the power supplied by the prime mover. Furthermore, as shown in Fig. 4.2, let V_i [pu] and θ_i [rad] denote the magnitude and angle of the machine terminal voltage, respectively, X_i [pu] denote the internal machine reactance, and E_i [pu] denote the magnitude of the internal machine voltage. Then the machine dynamics are modeled as

$$\frac{d\delta_i}{dt} = \omega_i - \omega_s \tag{4.1}$$

$$\frac{d\omega_i}{dt} = \frac{1}{M_i}P_i^m - \frac{D_i}{M_i}(\omega_i - \omega_s) - \frac{E_i V_i}{X_i M_i}\sin(\delta_i - \theta_i) \tag{4.2}$$

where D_i [rad/s] is the damping coefficient of the spinning mass, M_i [s^2/rad] is the scaled inertia constant of the machine, and ω_s is the synchronous speed of the machine in electrical radians per second.

Upon inspecting (4.2), we see that each of the three terms can be attributed

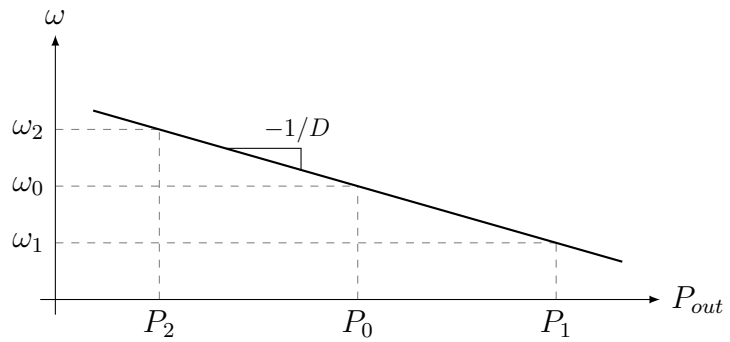


Figure 4.3: Droop characteristic of single synchronous machine

to distinct sources,

$$\frac{d\omega_i}{dt} = \underbrace{\frac{1}{M_i}P_i^m}_{P_m} - \underbrace{\frac{D_i}{M_i}(\omega_i - \omega_s)}_{P_{fw}} - \underbrace{\frac{E_i V_i}{X_i M_i}\sin(\delta_i - \theta_i)}_{P_e}. \tag{4.3}$$

In particular, the first term, P_m, is the power supplied by the prime mover while the second term, P_{fw}, is the power lost in the rotating mass due to friction and windage. The final term, P_e, is the electrical power extracted from the terminals of the generator.

We now use (4.3) to describe the steady-state behavior of the generator following a transient such as an increase or decrease of the electrical load. In steady-state, the derivative of (4.3) will be zero; thus, the speed is given as

$$\omega_i = \frac{1}{D_i}\left(P_i^m - \frac{E_i V_i}{X_i}\sin(\delta_i - \theta_i)\right) + \omega_s. \tag{4.4}$$

Immediately following a load change (or without the addition of a governor), the power supplied by the prime mover, P_m, remains constant. Thus we see that, given a change in load, in order for energy to be conserved, the speed of the rotating mass must change. Specifically, an increase in load will act to slow the system down while a decrease in load will act to increase the speed. The amount by which the speed changes is governed by the inverse of the damping coefficient, D_i. As illustrated in Fig. 4.3, the relationship between the power output and the speed can be represented by a line with negative slope, causing the synchronous machine to exhibit a natural drooping effect.

If two (or more) synchronous generators are connected to a power system, there will be a unique speed at which the load will be distributed between

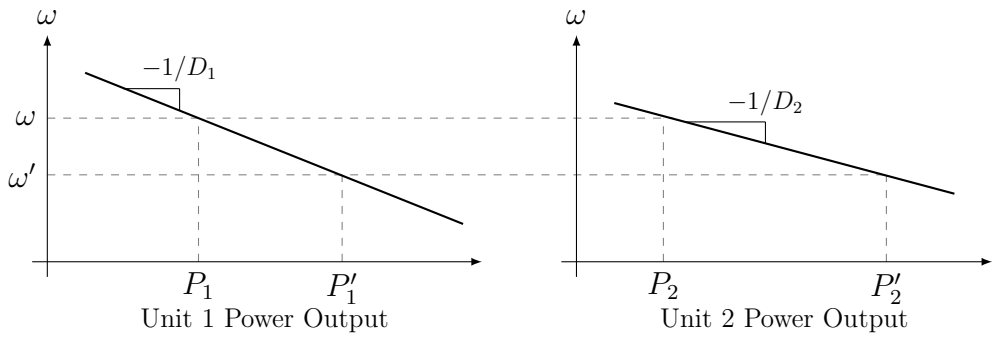


Figure 4.4: Droop characteristic of two connected synchronous machines

them. This behavior is illustrated in Fig. 4.4 which shows two synchronous generators connected to a common load. Suppose the generators are supplying power to an initial load $P_{\mathrm{L}} = P_1 + P_2$ at a speed of ω when the load increases to $\mathrm{P}'_{\mathrm{L}} = P_{\mathrm{L}} + \Delta P_{\mathrm{L}}$. For energy to be conserved, the machines will slow down in order to increase their power output until a new common speed, ω', is reached. The amount of load each generator will pick up is proportional to the slope of its droop characteristic. Thus if $D_1 \neq D_2$, then $P'_1 - P_1 \neq P'_2 - P_2$. Although the generators may not evenly divide the additional power, the increase in load is divided between the two generators such that $\Delta P_{\mathrm{L}} = P'_1 - P_1 + P'_2 - P_2$.

The inherent droop characteristic of the synchronous machines provides a natural way for determining the amount of generation needed to be added to or removed from the system in order to operate at a specified electrical frequency. Thus the leading DGR need only measure the frequency error in the system and multiply it by a gain to determine the generation mismatch as

$$\rho_e = k(\omega^{ref} - \omega), \tag{4.5}$$

where k has units [pu-s/rad]. In combination with the distributed algorithms, the computation of the generation mismatch by the leading DGR yields a two-stage closed loop controller as shown in Fig. 4.5. As the figure illustrates, the frequency error is used to determine the generation mismatch which is then dispersed among all DGRs by the distributed algorithms.

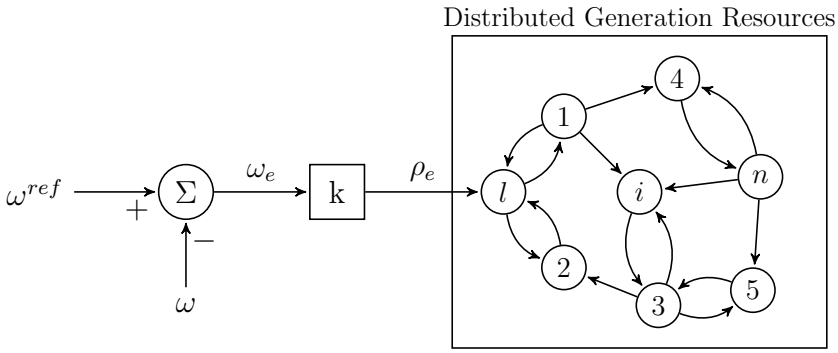


Figure 4.5: Two-stage control architecture

4.2 Small-Footprint Power System Setup

To demonstrate the ability of the distributed algorithms to control a set of DGRs, the six-bus, 240 V, 3-phase power system shown in Fig. 4.6 was constructed. The system is comprised of 3 Hampden Engineering (Table A.1) synchronous machines, G_1, G_2, G_3, and 3 wye-connected resistive loads, labeled as P_1, P_2 and P_3. Each synchronous machine is connected at the shaft to a permanent magnet synchronous servomotor (Table A.2) which serves as the prime mover. In order to regulate the frequency as the load in the system varies, the prime movers are operated in constant torque mode, with the torque command supplied by the nodes from the hardware testbed. Thus, the gain in equation (4.5) has units Nm/RPM and the generation mismatch has units Nm. Furthermore, the synchronous machines have 3 pole-pairs, therefore to maintain an electrical frequency of 60 Hz, the mechanical speed of the generators is regulated to 1200 RPM.

Table 4.1: Value of added inductances in power system

Parameter	Inductance [mH]
$L_{1,A}$	2.041
$L_{1,B}$	1.905
$L_{1,C}$	1.961
$L_{2,A}$	4.175
$L_{2,B}$	4.162
$L_{2,C}$	4.059

In the results presented throughout the remainder of this section, the generator at bus 1, G_1, is chosen to be the leader node. Furthermore, the per-phase resistance of each load is adjusted by adding 500 Ω resistors in

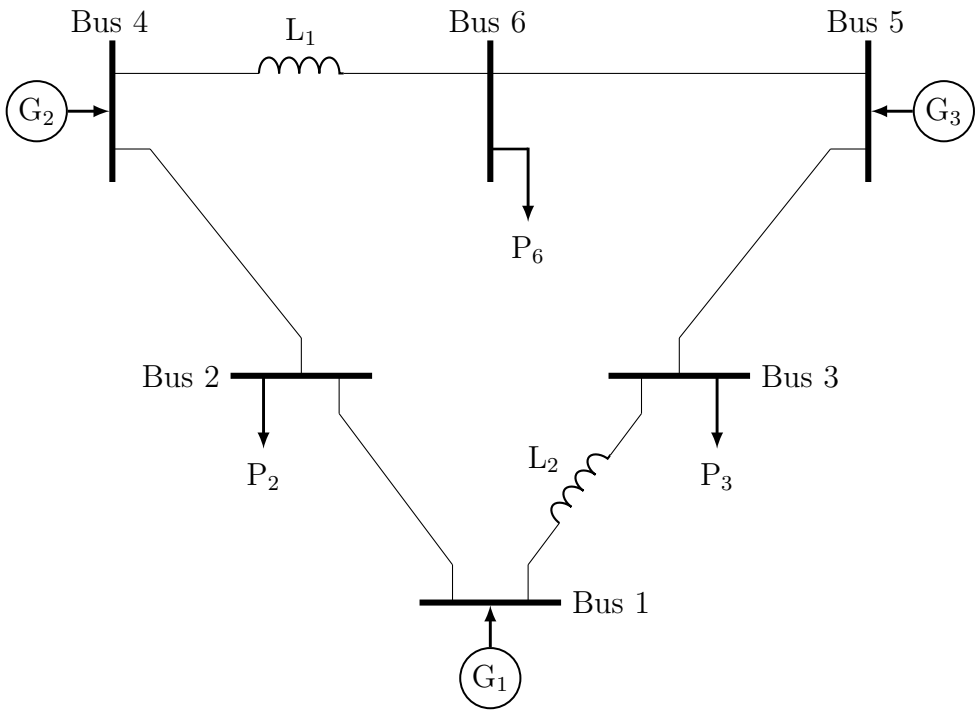


Figure 4.6: One line diagram of small-footprint power system

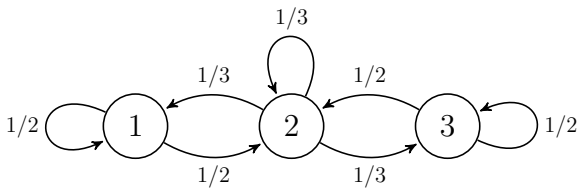


Figure 4.7: Graph of 3-node network

parallel. Each load can have up to 10 resistors in parallel per phase, yielding resistances in the range $500, 250, \ldots, 50\,\Omega$. Extra impedance is added via series inductors between bus 4 and bus 6 as well as between bus 1 and bus 3 as shown in Fig. 4.6. The per-phase inductances are given in Table 4.1.

4.3 Experimental Results

In this section, we present results from several experiments in which the hardware testbed is used to control the generators in the small-footprint power system described in the previous section. Cases demonstrating the generators fairly splitting the generation demand given a load increase and decrease are shown as well as a case in which the generation demand is split evenly

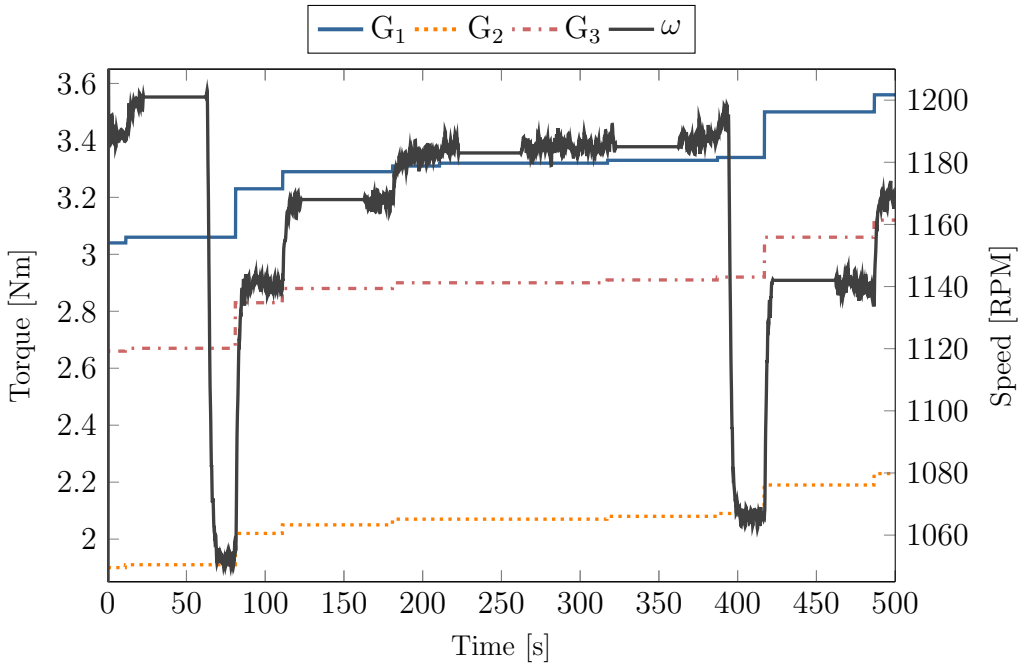


Figure 4.8: Prime mover torque and average speed during load increase with fair splitting

among the three generators. In the first three results, the communication between the nodes is represented by the graph shown in Fig. 4.7. In the final set of results, a fourth generator is added to the system to act as a spinning reserve. The extra generator participates in the distributed algorithms but does not adjust its output until the maximum capacities of the three original generators has been met. The robust algorithm with constraints is used in all cases.

We first discuss the results from increasing the load in the system using the fair splitting algorithm. In this case, the maximum constraints of the generators are chosen to be $x_1^{max} = 4.0\,\text{Nm}$, $x_2^{max} = 2.5\,\text{Nm}$, and $x_3^{max} = 3.5\,\text{Nm}$ while the minimum constraints are $x_i^{min} = 0\,\text{Nm}$, $i = 1, 2, 3$. Thus the collective system capacities are $\chi^{max} = 10\,\text{Nm}$ and $\chi^{min} = 0\,\text{Nm}$. The controller gain used by the leader is $k = 0.003\,\text{Nm/RPM}$.

The plot in Fig. 4.8 shows the torque supplied by the prime movers and the average mechanical speed of the generators for several load increases. Initially, the load in the system is $R_2 = 250\,\Omega$, $R_3 = 166.67\,\Omega$, and $R_6 = 166.67\,\Omega$. After 500 seconds, the load is increased to $R_2 = 166.67\,\Omega$, $R_3 = 125\,\Omega$, and $R_6 = 166.67\,\Omega$. As the figure shows, all the generators increase

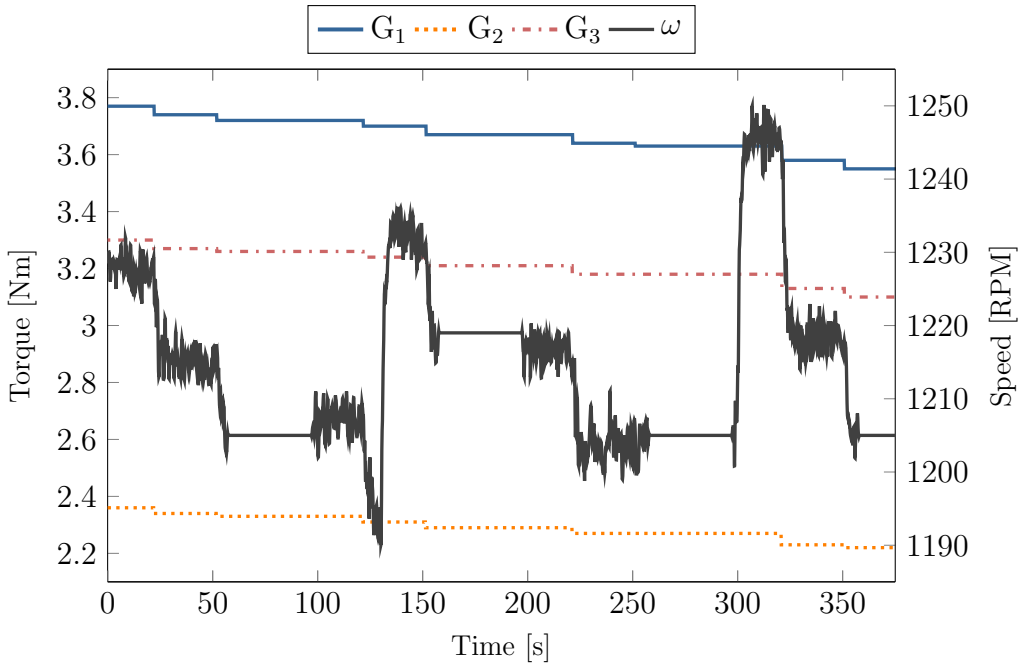


Figure 4.9: Prime mover torque and average speed during load decrease with fair splitting

their output in a way that is proportional to their limits. Furthermore, although the speed does not return exactly to 1200 RPM, the distributed algorithms serve to increase the output of each DGR given an increase in the system load and subsequent decrease in speed.

Using the same constraints as in the previous case, the plot in Fig. 4.9 shows the torque supplied by the prime movers and the average mechanical speed of the generators for several load decreases. The initial load in the system is $R_2 = 125\,\Omega$, $R_3 = 166.67\,\Omega$, and $R_6 = 83.33\,\Omega$. After approximately 350 seconds, the system load has been decreased to $R_2 = 125\,\Omega$, $R_3 = 166.67\,\Omega$, and $R_6 = 500\,\Omega$. As the figure illustrates, the output of each generator is decreased as the system load is shed and the speed increases. Although the speed does not reach 1200 RPM, the plot shows that the algorithm allows the DGRs to decrease their output relative to their individual constraints.

We now discuss a case in which the constraints on the DGRs are removed and the load is split evenly among the three generators. The plot in Fig. 4.10 illustrates the prime mover torque and the average speed of the generators. As expected, the output of each generator is exactly the same throughout the

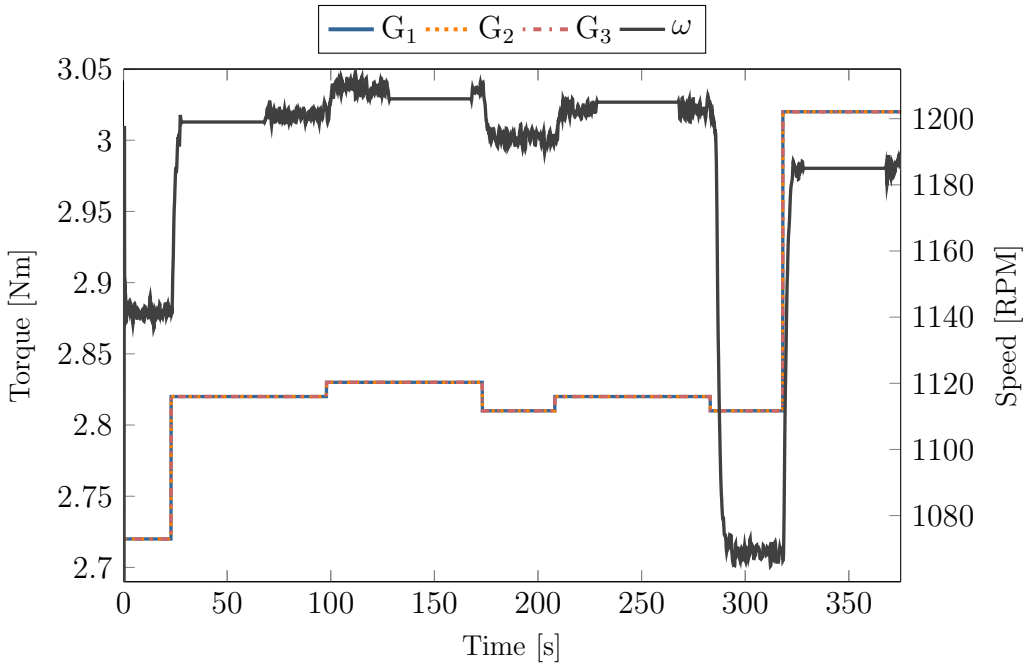


Figure 4.10: Prime mover torque and average speed during load increase with even splitting

experiment. At approximately 300 seconds, the load in the system increases such that the system speed drops to roughly 1050 RPM. After about 20 seconds, the generators all increase their output from 2.8 Nm to 3.01 Nm and the system speed increases to over 1180 RPM.

4.3.1 Spinning Reserve with Fair Splitting

In order to demonstrate the capability of each DGR to independently determine when the overall generation demand exceeds the collective constraints in the system, a fourth generator is added at bus 2 as shown in Fig. 4.11. The communication between DGRs is represented by the graph shown in Fig. 3.3.

Since all of the generators must operate in synchronism, the fourth generator is controlled in such a way that it only provides the necessary amount of torque required to be connected to the system. To facilitate this, the spinning reserve participates in the distributed algorithm with the other DGRs but sets its maximum capacity to the minimum amount of torque required for

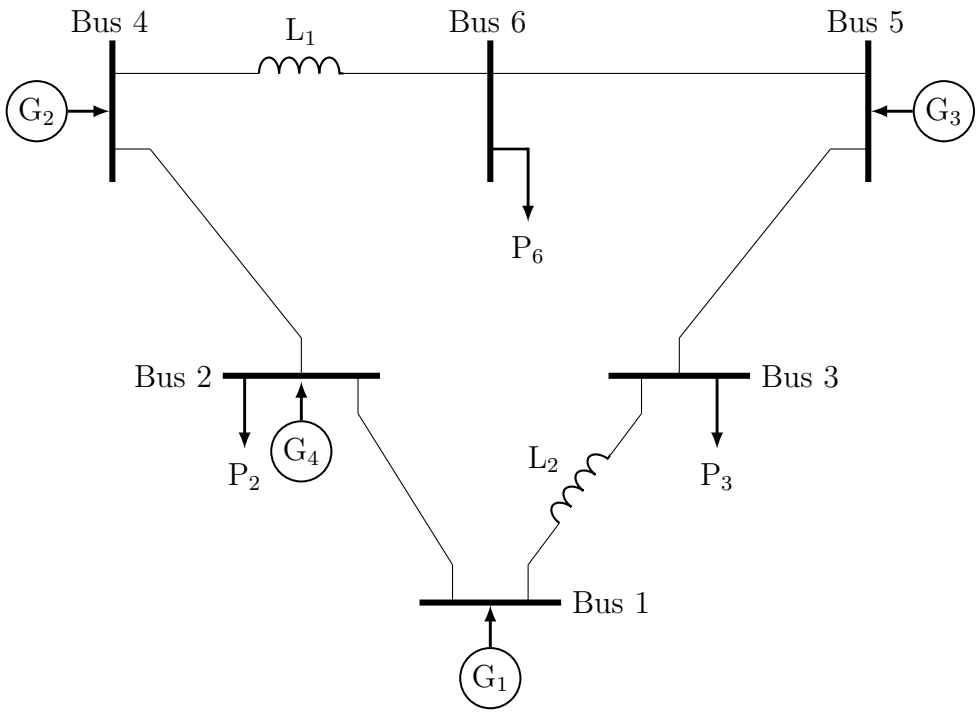


Figure 4.11: One line diagram of small-footprint power system with spinning reserve

synchronization. The DGRs have maximum constraints of $x_1^{max} = 3.5\,\text{Nm}$, $x_2^{max} = 2.5\,\text{Nm}$, $x_3^{max} = 3.0\,\text{Nm}$, and, initially, $x_4^{max} = 1.42\,\text{Nm}$, and minimum constraints $x_i^{min} = 0\,\text{Nm}$, $i = 1, 2, 3, 4$. Given the individual constraints, the collective capacity of the system is bounded by $\chi^{max} = 10.42\,\text{Nm}$ and $\chi^{min} = 0\,\text{Nm}$.

The plot in Fig. 4.12 shows the torque supplied by the prime movers connected to each DGR and the average speed over the course of several load changes. At approximately 70 seconds, the non-reserve generators have reached their maximum capacities and thus cannot increase their output in order to regulate the frequency. After this point, however, the spinning reserve determines that the total capacity in the system has been exceeded and adjusts its maximum output to its true value of $x_4^{max} = 2.5\,\text{Nm}$. This adjustment is realized at approximately 180 seconds at which point the other generators are able to reduce their output and the speed increases to approximately 1200 RPM.

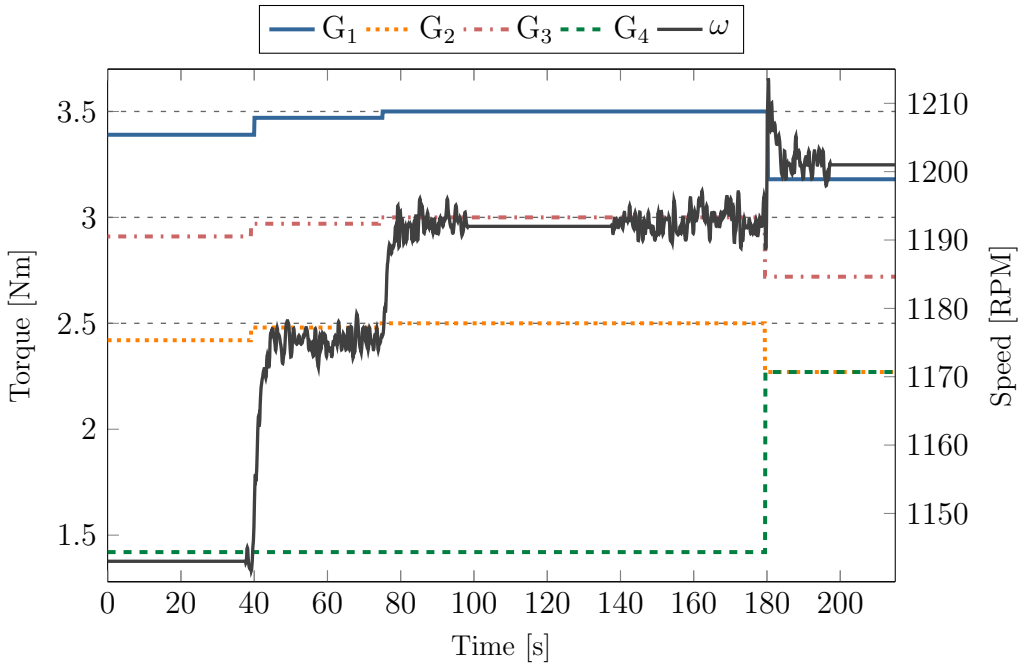


Figure 4.12: Prime mover torque and average speed during reserve switch-in with fair splitting

CHAPTER 5

CONCLUDING REMARKS AND FUTURE WORK

5.1 Concluding Remarks

In this thesis, several algorithms suitable for controlling distributed generation resources without the need for a centralized controller were proposed. We began by formulating an unconstrained algorithm that iteratively disperses the total generation demand among the DGRs and analyzed its convergence. We then extended this algorithm to account for individual DGR constraints and discussed how the result could be used by each DGR to ascertain the global state of the system. Finally we adapted the constrained algorithm to be more resilient to imperfect communication links. Each of the proposed algorithms was implemented using a hardware testbed comprised of low complexity devices capable of performing simple computations and exchanging information wirelessly with other nearby devices. Results were presented illustrating the capabilities of the hardware testbed as well as the evolution of the values computed at each iteration for the algorithms

Using the hardware testbed connected to synchronous generators in a small-footprint power system, we illustrated the efficacy of the algorithms as part of a two-stage control architecture for regulating the electrical frequency. Several results were discussed showing the change in system speed following an increase or decrease of the load and the subsequent adjustment of generation output of each of the DGRs. We concluded by showing a case in which one generator was treated as a spinning reserve by limiting its output until the remaining generators reached their capacity limits. This result demonstrates the ability for each DGR to independently determine when the overall demand for generation exceeds the collective constraints in the system.

All of the results presented herein were for systems comprised of relatively

few DGRs. Despite this, the algorithms are scalable, with only convergence speed being affected by the total number of nodes participating (and the connectivity of the communication network linking them). Furthermore, the small-footprint power system is just one of many applications that would be well-suited for a distributed control architecture similar to the ones proposed in this thesis. In fact, the algorithms discussed could be adapted for any class of applications in which one wishes to coordinate a set of distributed agents such that they collectively achieve a desired goal. Additionally, the algorithms could be used for applications in which resiliency and self-healing are important since the distributed nature obviates the need for a centralized controller with full knowledge of the network. One example application that would benefit from dynamic adaptation is to utilize the power electronics-based power supplies present in personal computers and/or uninterruptible power supplies to provide some control of the real and reactive power demand of a building.

5.2 Future Work

As mentioned above, the application to small-footprint power systems is just one example in which the proposed algorithms could be utilized. As part of our future work, we plan to use the hardware testbed for controlling other devices such as grid-tied inverters connected to photovoltaic arrays or uninterruptible power supplies. Furthermore, we would like to expand the application to controlling other resources such as reactive power for voltage support in power systems. We also envision other energy sources such as the aforementioned inverters and other power electronics-based devices supplementing the synchronous machines in our small-footprint power system setup.

Another aspect we would like to address is the costs associated with each DGR. While we demonstrated cases in which individual DGR constraints were accounted for, we neglected the incremental costs associated with increasing or decreasing the output of the DGRs. Given a quadratic cost function and upper and lower bounds on the output of each DGR, we would like to find a solution that minimizes the total cost while meeting the total demand for generation without violating DGR limits. That is, we would like

to use a distributed algorithm to find x_i for $i = 1, \ldots, n$, such that

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \frac{(x_i - \alpha_i)^2}{2\beta_i} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = \rho_d \\ & 0 < x_i^{min} \leq x_i \leq x_i^{max}, \ \forall i, \end{aligned} \tag{5.1}$$

where $\alpha_i \leq 0$ and $\beta_i > 0$ are real numbers. To achieve this, we plan to expand our work in [23] by implementing the proposed optimal solution utilizing the hardware testbed and using it to optimally dispatch the DGRs in our small-footprint power system setup.

APPENDIX A

SYNCHRONOUS MACHINE AND SERVOMOTOR NAMEPLATE SPECIFICATIONS

Table A.1: Hampden Engineering Corporation Synchronous Machine

Parameter	Value
Armature Voltage	133/230 RMS Volts
Armature Current	15.5/9 RMS Amps
Horsepower	2 Hp
Speed	1200 RPM
Frequency	60 Hz
Model	Syn-2

Table A.2: Kollmorgen Goldline Brushless Permanent Magnet Servomotor

Parameter	Value
Stall Current (Continuous)	10.3 RMS Amps
Stall Current (Peak)	33.0 RMS Amps
Torque (Continuous)	6.44 Nm
Torque (Peak)	19.5 Nm
Rated L/L Voltage	230 RMS Volts
Torque (Continuous)	6.44 Nm
Maximum Speed	4900 RPM
Frequency	164 Hz
Model	B-206-C-21

REFERENCES

[1] "Smart grid — Department of Energy," October 2011. [Online]. Available: http://energy.gov/oe/technology-development/smart-grid

[2] G. Joos, B. Ooi, D. McGillis, F. Galiana, and R. Marceau, "The potential of distributed generation to provide ancillary services," in *Proc. of IEEE Power Engineering Society Summer Meeting*, vol. 3, Seattle, WA, 2000, pp. 1762 – 1767.

[3] M. Baran and I. El-Markabi, "A multiagent-based dispatching scheme for distributed generators for voltage support on distribution feeders," *IEEE Transactions on Power Systems*, vol. 22, no. 1, pp. 52 –59, Feb. 2007.

[4] K. Turitsyn, P. Šandulc, S. Backhaus, and M. Chertkov, "Distributed control of reactive power flow in a radial distribution circuit with high photovoltaic penetration," in *Proc. IEEE Power and Energy Society General Meeting*, July 2010, pp. 1 – 6.

[5] A. D. Dominguez-Garcia, C. N. Hadjicostis, P. T. Krein, and S. T. Cady, "Small inverter-interfaced distributed energy resources for reactive power support," in *Proc. Applied Power Electronics Conference and Exposition*, March 2011, pp. 1616 –1621.

[6] C. Guille and G. Gross, "A conceptual framework for the vehicle-to-grid (v2g) implementation," *Energy Policy*, vol. 37, no. 11, pp. 4379 – 4390, 2009.

[7] A. D. Domínguez-García and C. N. Hadjicostis, "Coordination and control of distributed energy resources for provision of ancillary services," in *Proc. IEEE International Conference on Smart Grid Communications*, Gaithersburg, MD, October 2010, pp. 537 –542.

[8] A. D. Dominguez-Garcia and C. N. Hadjicostis, "Distributed algorithms for control of demand response and distributed energy resources," in *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), 2011* , Dec. 2011, pp. 27 –32.

[9] A. D. Domínguez-García, C. N. Hadjicostis, and N. H. Vaidya, "Distributed algorithms for consensus and coordination in the presence of packet-dropping communication links - part i: Statistical moments analysis approach," University of Illinois at Urbana-Champaign, Coordinated Science Laboratory technical report, September 2011.

[10] S. Cady, A. Domínguez-García, and C. Hadjicostis, "Robust implementation of distributed algorithms for control of distributed energy resources," in *North American Power Symposium (NAPS), 2011*, Aug. 2011, pp. 1 –5.

[11] D. B. West, *Introduction to Graph Theory*, 2nd ed. Prentice Hall, 2001.

[12] R. Horn and C. Johnson, *Matrix Analysis*. New York, NY: Cambridge University Press, 1985.

[13] "Arduino," October 2011. [Online]. Available: http://www.arduino.cc

[14] "Arduino mega 2560," Feb. 2012. [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardMega2560

[15] "Atmel atmega2560," Feb. 2012. [Online]. Available: http://www.atmel.com/devices/ATMEGA2560.aspx

[16] "Digi International Inc." October 2011. [Online]. Available: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-series1-module.jsp

[17] "Sparkfun electronics," October 2011. [Online]. Available: http://www.sparkfun.com/

[18] ZigBee Alliance, "ZigBee Specification," October 2011, San Ramon, CA. [Online]. Available: http://www.zigbee.org

[19] A. Rapp, "xbee-arduino," October 2011. [Online]. Available: http://code.google.com/p/xbee-arduino/

[20] H. Dai and R. Han, "Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, pp. 125–139, January 2004. [Online]. Available: http://doi.acm.org/10.1145/980159.980173

[21] A. Wood and B. Wollenberg, *Power Generation, Operation, and Control*. New York, NY: Wiley, 1996.

[22] P. Sauer and A. Pai, *Power System Dynamics and Stability*. Upper Saddle River, NJ: Prentice Hall, 1998, pp. 106–107.

[23] A. D. Dominguez-Garcia, S. T. Cady, and C. N. Hadjicostis, "Decentralized optimal dispatch of distributed energy resources," Under review for *51st IEEE Conference on Decision and Control (CDC), 2012* , Dec. 2012.