PRESERVING SAFETY IN DISTRIBUTED CYBER PHYSICAL SYSTEMS WITH UNRELIABLE COMMUNICATION CHANNELS

BY

FARDIN ABDI TAGHI ABAD

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Adviser:

Associate Professor Marco Caccamo

ABSTRACT

Cyber-physical systems (CPS) may interact and manipulate objects in the physical world with the aid of communication channels. Additionally, due to their nature, most CPS are safety-critical systems where there are safety invariant that need to be preserved. The big challenge is that communication channels are unreliable meaning that there may not be bounds on message delays. this will pose a threat to the safety of system. Guaranteeing safety for these systems can be even further complicated as physical components with which these systems interact may not have accurate physical models available.

In this Thesis we discuss two approaches to solve the safety problem. In the first part, we discuss a general methodology and architecture for distributed CPS design in order to increase the resiliency to communication faults. In this approach, each node exploits physical connections between nodes to estimate some of the state parameters of the remote nodes in order to detect the faults and also to maintain stability of system after fault occurrence. Finally, as a case study, a fault-resilient decentralized voltage control algorithm is presented and evaluated.

In the second part of the thesis, we address the challenge of proving safety and progress in distributed CPS communicating over an unreliable communication layer. This is done in two parts. First, we show that system safety can be verified by partially relying upon run-time checks, and that dropping messages if the run-time checks fail will maintain safety. Second, we use a notion of *compatible action chains* to guarantee system progress, despite unbounded message delays. We demonstrate the effectiveness of our approach on a multi-agent vehicle flocking system, and show that the overhead of the proposed run-time checks is not overbearing.

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to University of Illinois at Urbana-Champaign and the Computer Science Department for providing the opportunity for me to thrive and prosper in a vibrant environment. I would like to thank my adviser, Dr. Marco Caccamo for all his patient and generous guidance throughout past years. I am grateful to him for giving me the opportunity to be part of his team. I am thankful to Dr. Lui Sha for his support for my work on secure core project. I am thankful to Dr. Sibin Mohan that has provided his unlimited support for all the questions I have had. I am also thankful to my former teammate, Dr. Stanley Bak for being not only a collaborator but also a role model for my academic path. In addition, I have had the pleasure of collaborating with some of the brightest people in UIUC and I would like to thank all of them; Professor Carl Gunter, Man-Ki Yoon, Zhenqi Huang, Muhammad Naveed, Rohan Tabish, and Or Dantsker.

I also wish to thank all the friends that have been part of my life in Urbbana-Champaign for the past few years and for the years to come. My especial thanks to Jia, for all her support, encouragement and for her being my source of happiness during difficult times. Thanks to Adel Ahmadyan for having the biggest heart on the universe. To Renato Mancuso for being a colleague, a friend, and a brother. To Amin Ansari, for being my first go-to person for lots of my daily struggles and non-sense thoughts. To Siddhart Gupta, for being a friend who can always be relied on in any situation. I am indebted to my friends Jackie Alexander, Prakalp Srivastava, Nikita Spirin, Kuan-Yu Tseng, Neriman Tokcan, Andrea Llamu, David Dorimani, Paul Rausch, Hosseyn Tagharrobi Nia, Faraz Faghri, and Amir hossein Masnadi Shirazi for sharing so many unforgettable moments with me.

I would like to thank my family and dedicate this thesis to them. Especially I wish to thank my mother from whom I have learned patience, love,

confidence and strength. She has been the one who has supported my decisions, where no one else has and played the most important role in who I am today. I wish to thank my brother, Armin, who has been a younger brother, a father, a best friend, a competitor, a motivator, and an adviser for me. And finally, I wish to dedicate this thesis to memory of my father, whose influence on my thoughts and goals are getting more and more clear.

TABLE OF CONTENTS

CHAPTER 1

PRELIMINARY AND INTRODUCTION

1.1 Introduction

Cyber-physical systems are a class of systems in which physical systems are in tight combination with computation elements and they feature a high level of coordination with each other. Distributed cyber-physical systems, as an important subgroup of cyber-physical systems, can be defined as a set of interconnected computer controlled physical plants that physically affect each other. Which means, output of each distributed node is not only a function of its own control inputs and state variables, but also is a function of state in other nodes of the system depending on connection graph and physical structure. Examples of these types of systems include power grids, water and waste-water distribution system, and traffic control system. High criticality of this class of systems is the main reason that maintaining safety and stability is the first priority for every system architect. Any accidental or malicious fault in any of the above components can lead to huge costs and irreparable damages.

One of the biggest challenges in the guaranteeing safety of distributed CPS is the communication channel. Real-worlds communication channels are unreliable meaning that packets are not guaranteed to be delivered. A packet in a channel, can get delayed for unbounded amount of time, can get dropped in the routers or can get reordered relative to other packets. In addition, lots of software written for these systems assume the minimum reliability level provided by TCP/IP protocols. However, physical failures can violate TCP/IP guarantees, therefore, any control protocols for CPS must be tolerant to any type of communication failures.

An example of a distributed CPS is autonomous coordinated vehicle motion. A set of vehicles is moving through a shared physical space, and the user

would like to be able to make run-time changes to the routes of the vehicles, while guaranteeing that a formation is maintained and vehicles will not collide. Since messages may be lost over wireless, any new route command may arrive at some vehicles but not at others. Acknowledgments will not solve this problem, since acknowledgments may also sometimes be lost. Clearly, without appropriate solutions, using current communication channels would not be able to guarantee the safety predicate in this system.

Electric grid is another well-known distributed CPS that has two general control architectures; centralized and decentralized. In centralized methods, all the nodes of the system are connected through communication channels to a central supervisory control and data acquisition center. In decentralized algorithms, each node only communicates with only a subset of nodes without any central controller. In [1], based on examination of 162 disturbances reported by the North American Electric Reliability Council (NERC) authors indicate that, "information system failures contribute to a very high percentage of large failures". In both cases, the communication channels play an essential role and failures, faults or latency in them can endanger system safety.

As in distributed systems with lossy communication, it is impossible to achieve consensus in this system [2]. Despite this inherent limitation, we propose two approached to ensure the safety of the system. In the first approach which is described in chapter 2, we propose using physical signals as an alternative to the communication channels during communication faults. Our proposed architecture enables each node to independently manage its states such that the whole system remains stable until the communication resumes. After description of the general architecture, we investigate a distributed voltage control approach for power grid as a case study. We apply our architecture to this system, and show how voltage can remain in the safe range while some channels are lost. This approach is considered as most useful for highly interconnected CPS that do not have open loop stability.

In the second chapter if this thesis, we have introduced a communication/control protocol which can guarantee safety of the system and provide progress under certain conditions. In this chapter We consider a CPS scenario consisting of several embedded computing components each interacting and sensing the physical world and communicating with a central coordinator over an unreliable channel, such as wireless or the Internet. These low-level

controllers attempt to accomplish some task in a coordinated fashion. Since the physical world is being manipulated, it is essential that the supervisory control logic is carefully designed and satisfies strict safety requirements. This solution is designed for centralized systems where there is a central controller who issues command for the rest of the agents. By using the proposed approach we can ensure the safety invariant. If the communication channel eventually delivers packets, we can also provide the notion of progress that enables the system to safely change the commands at run-time.

Finally, in the last chapter we provide a look into the future directions and challenges that are still remaining.

1.2 References and Acknowledgments

The material presented in this work is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS-1302563, CNS-1219064 and CNS-1035736, and by John Deere under Award No. UIUC-CS-DEERE RPS #19. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF or John Deere.

Some of the material here has previously appeared in proceedings [3, 4] and is copyrighted by the Institute of the Electrical and Electronics Engineers (IEEE). Portions of this thesis, with permission, are reprinted from:

- "Using Run-Time Checking to Provide Safety and Progress for Distributed Cyber-Physical Systems", S. Bak, F. Abdi taghi abad, Z. Huang, M. Caccamo, Proceedings of the IEEE conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'13), Taipei, Taiwan, August 2013.

- "A Fault Resilient Architecture for Distributed Cyber-Physical Systems", F. Abdi Taghi Abad, B. Robbins, and M. Caccamo, Proceedings of the IEEE conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'12), Seoul, Korea, August 2012.

1.3 Related Work

Networked control systems have been employed in a variety of industrial automation applications. Recently, industrial wireless protocols and products have been developed as replacements for wired control systems [5, 6]. These were made not only to reduce costs due to materials (wiring), installation and wire maintenance, but also provide benefits in flexibility by allowing easy modification to the existing communication infrastructure. One benefit of using these solutions is that they strive to reduce (but can not eliminate) problems arising from communication delay and packetloss when wireless is used in industrial control systems.

The Simplex Architecture [7] was developed as an approach to increase system safety for individual Linear Time Invariant (LTI) control systems, by filtering commands from an untrusted controller and switching over to a safe backup mode. This approach can compliment the one presented in this work, by providing safety in the low-level controllers in a CPS architecture [8].

A network extension of Simplex has also recently been developed [9]. This work extended the Simplex approach to Linear Parameter Varying (LPV) systems, and incorporated network delays into the design. However, the analysis requires having a fixed upper bound on communication delay with no packetloss, which can not be guaranteed under wireless communication. Our guarantees of safety and progress hold without a fixed upper bound on communication delay, and, in the case of safety, we allow unrestricted packetloss to occur.

Our approach draws inspiration from the NASS framework developed to provide safety for medical systems communicating over wireless [10]. This system uses discrete dynamics with formal safety properties in a supervisory control system over wireless. Each command message includes a backup command vector, which is used if no further commands arrive. A safety filter provides protection from faults in the high-level control. This filter needs to reason about the worst-case packet delivery combinations, which in the case of the considered discrete system involves model-checking the possible combinations of packet reception and agent states. In our approach, we use a more control-oriented approach to providing safety and progress which allows for continuous state variables, and provide a method to help construct pair-wise compatible chains to guarantee progress.

Run-time approaches have been considered to create verified systems [11]. In this work, a time-bounded reachability computation is performed during system operation in order to determine if a controller should be disengaged. The advantage of this is approach is, since at runtime some of the variables are known, only a smaller state space needs to be considered. This is also the argument we make when advocating the design of the Runtime Command Monitor.

For partially synchronous systems, where messages get bounded nondeterministic delays or dropped, a sufficient condition for verifying convergence properties has been established [12]. The sufficient conditions require that (i) messages get delivered infinitely often and (ii) there exist some invariant neighborhood topology of the system satisfying a Lyapunov-type property.

For asynchronous distributed systems, where messages get nondeterministic but bounded delay, a static approach for reasoning about the convergence of an asynchronous system has been proposed [13]. The approach shows that under under some additional assumptions about the shape of the sublevel sets of the Lyapunov function, if convergence occurs in perfect communication, where messages get delivered instantly without dropping, convergence will also occur in the the corresponding synchronous system.

CHAPTER 2

A FAULT RESILIENT ARCHITECTURE

2.1 Introduction

As discussed in the introduction section, communication channels are one of the most critical components of the distributed cyber-physical systems. There has been efforts on the communication community to provide more robust and secure communication for specific CPS applications [14]. In [15], ubiquitous TCP/IP protocols were deployed to provide reliable data delivery for a cyber-physical system. These protocols have unpredictable latency which can cause major problems for time-critical control decisions [16, 17]. While most of these efforts in communication networks have focused on prevention, there is not much work for system control methods under malicious or accidental fault situations [18]. In case of any data streaming disconnection or delayed packet arrival, controllers would have to make decisions based only on their local state and unaware of rest of the system. This will highly increase the chance of taking system into a non-safe state.

Another issue with power grid infrastructure is that the technology currently being deployed by power grid communication infrastructure belongs to few decades ago during which many of the current advances in the distributed computing was not even made. Due to under-investment and heavy cost for transitioning to new communication solutions, deployment of new technologies in this infrastructure will not happen in a close future [19, 20].

Thus, a solution that can help overcome this problem without huge changes to the existing infrastructure is highly valuable. In this chapter, we are trying to exploit some unique features of cyber-physical systems and propose an architecture for this purpose.

We noticed that most of the previous works have not considered the dynamics of physical systems and how they can be used to detect compromised

nodes or a fault in a component [18]. We are exploiting physical connections between different nodes in order to enable each node to estimate state of the other nodes. Our architecture can be implemented with existing communication infrastructure and is resilient to the faults that might occur in the communication channels.

In section 2.2, first two unique features of cyber-physical systems are discussed. Based on this discussion, the general fault resilient architecture and design approach is described. In section 2.3, following this architecture, a fault resilient decentralized voltage control algorithm is suggested. In section 2.4, the effect of faults on unmodified decentralized voltage control algorithm and fault resilient decentralized voltage control algorithm is compared using two different communication fault scenarios.

2.2 Fault Resilient Controller Architecture

In this section, we first give a brief explanation about structure of physical connections and communication links between the nodes of distributed cyber-physical system followed by a discussion on the features of cyber-physical systems that we want to exploit for our architecture. The last part describes the architecture of a fault resilient controller.

2.2.1 Preliminary

The communications network and physical network initially share the same graph, i.e., their lines share the same transmission line. However, the structure of the graph can change due to failures or different operating modes.

The physical system can be represented by a graph $G^p = \{V, E^p\}$, where $V = \{1, 2, ..., m\}$ is the set of nodes and the edge set $E^p \subseteq V \times V$ represents the physical interconnections between each pair of nodes. By convention, we assume the graph is undirected, i.e., if $(i, j) \in E^p$, then $(j, i) \in E^p$. N_i^p is the set of physical neighbors of node i. G^p is considered to be strongly connected when there is a path of finite size from node i to node j for all $i, j \in V$.

Similarly, we define the graph $G^c = \{V, E^c\}$ that shares the same vertex and edges set as G^p to represent the communication network. We define N_i^c as the set of cyber neighbors of node i which can exchange data with node

i. Node j is said to be a disconnected neighbor of node i if $(i, j) \in E^p$ but $(i, j) \notin E^c$, i.e., node i and j are physical neighbors but the communication channel connecting them is not available. D_i represents set of disconnected neighbors of node i. Since physical faults are outside the scope of this work, we assume that E^p and N_i^p remain constant. However, E^c and N_i^c change when a fault occurs on a communication channel.

Each node can be classified as one of the following categories:

- Connected Nodes: Nodes that have an empty D_i set which means, $\|D_i\| = 0$. Initially, all the nodes are categorized under this category.
- Partially Connected Nodes: Nodes that have at least one connected neighbor and one disconnected neighbor which means, $\|D_i\| \geq 1$ and $\|N_i^c\| \geq 1$.
- Totally Disconnected Nodes: are the nodes that have an empty set of neighbors which means, $\|N_i^c\| = 0$.

Since system has a connected graph, there is no node with $\|N_i^c\| = 0$ and $\|D_i\| = 0$.

2.2.2 Features of Cyber-Physical Systems

While cyber-physical systems share many similarities with other computational systems such as data networks, cloud servers, and real-time systems, cyber-physical systems have some unique features among them. First, some of issued cyber messages are directly or indirectly related to a physical state. Second, the nodes in a distributed cyber-physical system are physically connected to one another, which enables them to estimate certain states of rest of the system.

Cyber messages are classified as either command or report messages. Command messages are sent to invoke specified physical changes for certain nodes in the system and the changes will be observed in the next cycle. Report messages are sent to check the current physical state of the system. For example, when a central controller in a water distribution system issues a stop command for a particular tap, the water flow in the pipe must drop down to zero after that point. If water is observed to be flowing, then this would be considered a fault in either message passing channels, controller in charge of

receiving and interpreting the command, or the physical plant in charge of performing command, i.e. mechanical tap. In this situation, a report command occurs when the message indicating amount of water flow being sent from one neighbor to another one does not match with the measured water flow. This can indicate a communication failure, reporter failure (failure in sensors or controller of reporter), or a physical failure such as water pipe leakage.

The physical connections between nodes enable them to potentially estimate certain state parameters of their neighbors within a specified tolerance based on local measurements. In these systems, nodes can use a combination of local measurements and estimated state variables to supplement the data received over communication channels. Relating back to the water distribution system example, based on measurement of flow that enters one node of the system which is a locally measurable parameter for each node, and also some prior information about physical structure of pipes one node can have an estimation of water pressure in its immediate physical neighbors.

As a result of these characteristics, our proposed architecture exploits the estimated states of remote nodes to detect communication faults and maintain the overall stability of the controlled cyber-physical system. The proposed architecture is described in the next subsection.

2.2.3 Architecture Description

Fig. 2.1 shows the various layers of our architecture and illustrates the communications between them. In the rest of this subsection, the descriptions of each component and design approach is provided.

Estimation Unit

A system designer needs to have a deep understanding of the physical structure of the system in order to extract all the relations between physical variables of distributed nodes. In addition, designer needs to analyze available locally measurable variables. Physical connections can provide a set of linear or non-linear equations, inequalities, or logical relations between state variables of different nodes. These relations can be utilized by a node when reliable data is not received in order to estimate state variables of remote

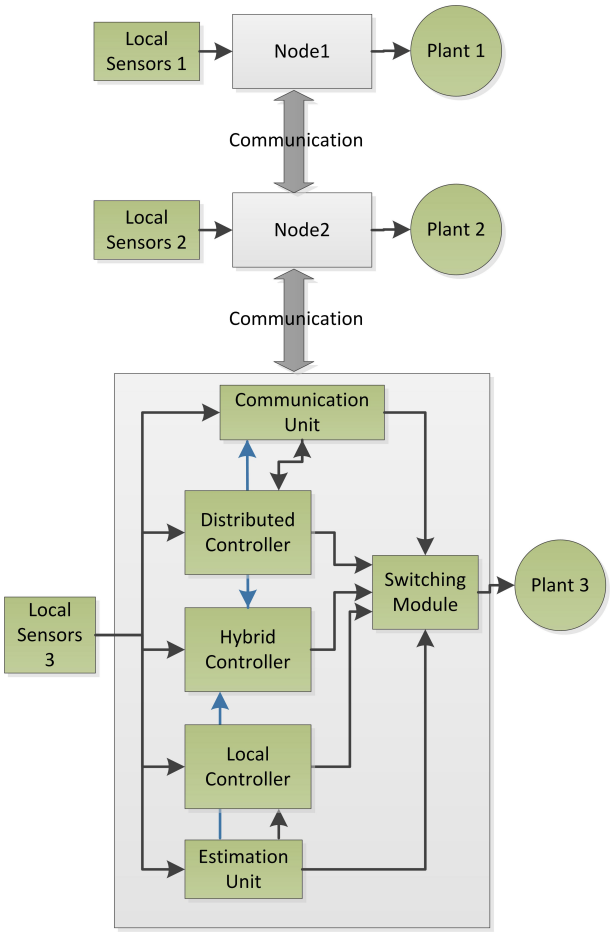


Figure 2.1: Fault Resilient Controller Architecture

nodes. Considering all estimation methods, physical connections, and local measurements designer can detect all the remote state variables that can be estimated by each node. Simply by reading data directly provided by local sensors, estimation unit estimates some of the state variables of remote nodes.

Accuracy of the estimations depends on the physical connections, accuracy of sensors and accuracy of estimation equations. Maximum error of estimation for remote variable X, which can be determined through experiment, is called e_x^{max}.

Communication Unit

At each control cycle, communication unit transmits two set of information to the neighbor nodes in following form.

$$TransmitPacket = \{Packet_{dist} \cup Packet_{meas}\}$$

$Packet_{dist}$ is the set of data that is required for distributed controller represented. Transmitting $Packet_{dist}$ is the purpose of communication. Basically all the communication channels are established in order to transmit this set.

$Packet_{meas}$ represents the measured local variables that are estimable for the neighbor nodes and is being transmitted in order to be used by the other neighbors to verify if the communication channel is working properly. Simply, if the communication unit and communication channel work properly, the estimated value and received amount would match.

In addition, communication unit has a buffer associated with each neighbor which is updated whenever a new value is received from that neighbor.

Switching Module

Switching module is in charge of detecting communication failures and switching to the appropriate controller depending on the situation of node.

When a new message is received from remote nodes by communication unit, it overwrite the previous message in the buffer. Switching module periodically reads the buffer and checks if the following inequality holds or not.

$$|x_{data} - x_{est}| \leq e_x^{max} \tag{2.1}$$

In the above inequality, x_{data} is read from communication unit buffer which is reported by the remote node through communication channel, x_{est} is the estimated value of x and e_x^{max} is the maximum estimation error for variable x.

Inequality (2.1) checks whether the estimated value and received value match with each other and it must hold for a correctly received remote state variable. In case of not receiving a new value or receiving a corrupted value, the inequality will be violated and a communication fault is detected.

Once a failure is detected for the communication channel of the neighbor $b_j \in N_i$, b_j is removed from set of cyber neighbors of b_i, N_i^c and is added to set of disconnected nodes of b_i, D_i. After this point, b_i does not send local measurements to b_j in order to make it add b_i into its disconnected nodes set, D_j.

Switching module keeps track of N_i^c and D_i and determines the category to which the node belongs based on number of members of $\|N_i^c\|$ and $\|D_i\|$. Every time that a new fault is detected, it triggers a switch to hybrid controller if node is categorized as a partially disconnected node and it triggers a switch to the local controller if the node is categorized as a totally disconnected node.

Notice that a switch is triggered only when a fault is detected in at least one of the communication channels. Thus this node cannot be categorized as a connected node anymore.

Distributed Controller

This controller is designed for the normal operation mode when communication is properly working and reliable data is received from all the neighbors. For most of the existing distributed cyber-physical systems, their existing controller can be used in this part of fault-resilient architecture without any modification.

Hybrid Controller

Node b_i is controlled by this controller if $\|D_i\| \geq 1$ and $\|N_i^c\| \geq 1$, i.e. it has both disconnected and connected neighbors. Hence, hybrid controller has access to both communication unit in order to receive information of the connected nodes and also has access to the estimation unit outputs for its disconnected nodes.

In design of this controller, the algorithm implemented in the distributed controller should be modified in the way that the estimated data replace the received data.

Local Controller

The reason this controller is called a local controller is that it does not require network communication. This controller is activated when $N_i^c = 0$. Control algorithm implemented in this controller can only exploit a combination of estimated state variables of remote nodes and locally measured variables. The main goal of this controller is to prevent the node and immediate neighbors from instability. Since some of the state variables of remote nodes might be non-estimable , or due to estimation errors, the outcome of this controller may not lead to optimal solution.

2.2.4 Notes

In this work we assume that probability of controller and sensor faults is negligible compared to communication faults. However, there are classic fault tolerant solutions such as system redundancy that can make these components more reliable. Another well-known technique for increasing reliability of controller is simplex architecture as explained in [21], [22], [23], and [24].

2.3 Fault Resilient Decentralized Voltage Control Algorithm

In this section we apply proposed fault resilient architecture to an electric power grid which follows a decentralized voltage control algorithm. Our goal is to detect occurrence of communication faults, and then maintain voltages in the permitted range of $V^{nominal} \pm 0.05V^{nominal}$.

First, physical and communication structure model for the power system is described. Then a brief introduction to the decentralized voltage control algorithm (DVC) is presented followed by a detailed description of fault resilient architecture and local controller algorithm. Finally, two communication fault scenarios are presented and results of fault resilient versus unmodified original algorithm are compared with each other.

2.3.1 Power System Model

Power systems are analyzed and assumed to be in quasi steady-state at 60 hz for power flow calculations and dispatch commands [25]. Given a $n+1$ bus power distribution system, we fix the voltage at the reference bus, otherwise known as the slack bus, and consider the remaining n buses to be PQ buses. On each PQ bus i, the active power P_i and the reactive power Q_i are fixed. Then the voltage magnitude V_i and voltage angle θ_i are computed by solving for the roots of mismatch equations, which are the difference of the specified power with the computed powers $P(V,\theta)$ and $Q(V,\theta)$. We use the power flow equations to compute the sensitivities of bus powers to changes in voltage

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} \partial P/\partial\theta & \partial P/\partial V \\ \partial Q/\partial\theta & \partial Q/\partial V \end{bmatrix} \begin{bmatrix} \Delta\theta \\ \Delta V \end{bmatrix}. \tag{2.2}$$

However, the weak coupling between ΔP and ΔV, as well as ΔQ and $\Delta\theta$, enables (2.2) to be reduced to

$$\Delta P = [\partial P/\partial\theta]\,\Delta\theta \tag{2.3}$$

$$\Delta Q = [\partial Q/\partial V]\,\Delta V. \tag{2.4}$$

This is a common simplification that is used in the *fast decouple power flow* method [25]. Finally, we can estimate the voltage sensitivity to changes in reactive power by $S = [\partial Q/\partial V]^{-1}$. Voltage sensitivity of each node to the non-neighbor nodes' reactive power is assumed to be zero therefore $\overline{S}$ is produced based on S with non-zero entities only for neighbor nodes.

The physical and communication networks can be represented by the graphs G^p and G^c, respectively. The system has a total of $m \leq n$ buses (nodes) with distributed energy resources (DERs) attached that are able to provide some amount of reactive power support π_i, which is limited by a maximum capacity $\pi_i^{max} > 0$ and a minimum capacity to $\pi_i^{min} < 0$. In this model, the active power provided by DERs is not controlled. Note that for the remainder of the chapter, each electric node i will be represented by b_i.

2.3.2 Decentralized Voltage Control Algorithm (DVC)

The control architecture shown in Fig. 2.1 is applied to improve the reliability of an existing decentralized voltage control (DVC) algorithm published in [26]. We will provide a brief overview of the algorithm, while the reader can refer to [26] for a more detailed explanation and discussion.

The purpose of this algorithm is to maintain system voltages within the specified range $V_{nominal} \pm 0.05V_{nominal}$. According to DVC, bus voltages are measured at steps $r = \{1, 2, \dots\}$ and the distributed algorithm is given k steps between r and $r + 1$ to reach a solution. We begin the algorithm by estimating the amount of reactive power that if injected (or absorbed) at b_i, will correct the measured voltage limit violation at that bus, i.e., the controllers estimate this value with (2.9). After the initialization procedure, the controllers run a recursive algorithm in which their current value is a linear combination of their previous and that of their neighbors until the system converges. The algorithm has the form

$$\pi_i[k+1] = p_{ii}[k]\pi_i[k] + \sum_{j \in N_i^c} p_{ij}[k]\pi_j[k], \tag{2.5}$$

where $p_{ij}[k] \in [0, 1]$ for all i and $j \in \{N_i^c \cup i\}$. $P[0]$ is constructed based on:

$$p_{ij}[0] = \frac{\overline{s_{ij}}}{\sum_{k=1}^{n} \overline{s_{kj}}} \tag{2.6}$$

where n is the number of all the nodes in the system.

The DVC relies on the construction of a *column stochastic* matrix that must be *primitive*, i.e., a nonnegative matrix that is both irreducible and aperiodic. Primitivity of $P \in \mathbb{R}^{n \times n}$ is established by ensuring that the underlying graph of the network is *strongly connected* and aperiodicity follows from the diagonal entries $p_{jj}[k] > 0 \ \forall j$ [27]. Given these properties, the Perron-Frobenius Theorem for nonnegative matrices ensures that P has a single eigenvalue of maximum modulus, e.g., $\lambda_1 = 1$ and $|\lambda_i| < 1$ for $i = 2, \dots, n$.

Thus, the system can determine in a distributed manner the invariant solution to

$$\pi[k+1] = P[k]\pi[k], \tag{2.7}$$

where $\pi[k] \in \mathbb{R}^n$ are the DER contributions previously mentioned and the weights of $P[k]$ are potentially time-varying. Through a local exchange of

information, the controllers at each node will asymptotically converge to a steady-state solution π^{ss} where $\sum_i \pi_i[k]$ remains constant for all k. Moreover, the solution π^{ss} has the form $\pi^{ss} = \alpha\mu$ where $\alpha = \sum_i \pi_i[0]$ and $\mu \in \mathbb{R}^n$ is the right eigenvector of P such that $\sum_i \mu_i = 1$.

The authors in [28, 26] extend these results to develop a constrained version of (2.7). The controllers are able to compute the ratio of demand to the available capacity. This result is then multiplied by the available capacity at each node to determine a solution that maintains the overall reactive power demand for the system with respect to individual device capacities.

2.3.3 Fault Resilient Decentralized Voltage Control

Following the approach discussed in the previous section, the architecture suggested in Fig. 2.1 is utilized in order to implement the controllers of each node. DVC algorithm which is described in the previous section is implemented exactly as distributed controller. Estimation methods and switching policy that is implemented in estimation unit and switching module are explained in the rest of the section and control algorithm implemented in local and hybrid controller is explained in the last part of this section.

Physical Connection and Estimation Unit

In power grids the most important physical connection between nodes is through power lines. Following the basic results of circuit theory if $b_j \in N_i^p$ then the following relation holds for V_i and V_j.

$$V_j = V_i + Z_{ij} I_{ij} \tag{2.8}$$

In this equation, V_i and V_j are voltages of b_i and b_j, Z_{ij} is impedance of the physical line between b_i and b_j, and I_{ij} is the current flow from node i to j. This equation indicates that if node i has a prior knowledge of the impedance of the line $l_{ij} \in E^p$ and it can measure the current flowing from i to j, I_{ij}, which is a local variable, above equation will estimate voltage of node j based on a local state measurement of b_i. Estimated voltage of b_j from b_i's point of view is called V_j^i.

In addition to voltage, another state parameter of remote nodes that is

estimated by estimation unit of b_i is the amount of reactive power needed to correct the voltage limit violation at bus $j \in D_i$, called $\pi_j^i[0]$. In distributed algorithm, at the beginning of every step r of the control algorithm, each bus has to measure its local voltage and set the initial reactive power request, $\pi_j[0]$, based on the following equation and then start sending out $\pi_j[0]$ to all its neighbors.

$$\pi_j[0] = \begin{cases} \frac{0.95V_j^{nom} - V_j}{\|N_j^p\| \overline{s_{jj}}} & \text{if } V_j \leq 0.95V_j^{nom} \\ \\ \frac{1.05V_j^{nom} - V_j}{\|N_j^p\| \overline{s_{jj}}} & \text{if } V_j \geq 1.05V_j^{nom} \end{cases} \tag{2.9}$$

$\overline{s_{jj}}$ is sensitivity of V_j to π_j. Using estimated voltage of b_j from following equation, b_i can also estimate $\pi_j[0]$:

$$V_j^i = V_i + Z_{ji} I_{ji} \tag{2.10}$$

By using V_j^i, equation (2.9) can be written as:

$$\pi_j^i[0] = \begin{cases} \frac{0.95V_j^{nom} - V_j^i}{\|N_j^p\| \overline{s_{jj}}} & \text{if } V_j^i \leq 0.95V_j^{nom} + |e_{V_j}^{max}| \\ \\ \frac{1.05V_j^{nom} - V_j^i}{\|N_j^p\| \overline{s_{jj}}} & \text{if } V_j^i \geq 1.05V_j^{nom} - |e_{V_j}^{max}| \end{cases} \tag{2.11}$$

In the equation above, $|e_{V_j}^{max}|$ represents maximum estimation error of voltage of b_j from b_i which can be measured through experiment on the real power line between two neighbors.

Note that, $\|N_j^p\|$ and Sjj are static variables that do not change over the operation. Thus, each node needs to have this prior knowledge before the system starts.

Communicating Unit

Communication unit transmits data only if either distributed controller or hybrid controller are activated. In either cases, two sets of data need to be transmitted to all $b_j \in N_i^c$. First, distributed controller output or hybrid controller output which is for iterative distributed algorithm and second, locally measured voltage, V_i, received directly from sensor unit. The purpose of sending periodic updates of V_i is that b_js can detect a failure in commu-

nication by comparing V_i to V_i^j.

Switching Module and Switching Policy

Switching module reads V_j^{data} from communication unit buffer to check if the following inequality holds or not.

$$\forall j \in N_i^c : |V_j^{data} - V_j^i| \leq e_{V_j}^{max}$$

V_j^{data} is the value read from the buffer of communication unit for neighbor j and V_j^i is estimated voltage of j. in other words, if a deviation more than $e_{V_j}^{max}$ occurs between estimated amount of V_j and the measurements received through communication channel, a communication fault is declared.

Once a failure is detected by b_i in the communication link with b_j, b_j is removed from set of cyber neighbors of b_i, N_i^c and is added to set of disconnected neighbors of b_i, D_i. Then, b_i does not send V_i to $b_j \in D_i$ in order to make them add b_i into their disconnected nodes set, D_j. Finally, based on the new N_i^c and D_i, a switch is triggered.

Hybrid Controller Algorithm

This controller takes the control of the system when b_i is categorized as a partially connected node.

A partially connected node will participate in the iterative DVC algorithm with its connected neighbors but it cannot exchange messages with the disconnected ones. Therefore, reactive power requests of disconnected nodes are taken into account when initializing the $\pi_i[0]$. If b_i is a partially connected node following equation is used to initialize $\pi_i[0]$

$$\pi_i[0] = \begin{cases} \max(\{\pi_j^i[0]\frac{\overline{s_{ij}}}{\sum_{k\in N_j^p \cup j}\overline{s_{kj}}} : b_j \in D_i\} \cup \pi_i[0]) \\ \qquad\qquad \text{if } \pi_i[0] \geq 0 \\ \min(\{\pi_j^i[0]\frac{\overline{s_{ij}}}{\sum_{k\in N_j^p \cup j}\overline{s_{kj}}} : b_j \in D_i\} \cup \pi_i[0]) \\ \qquad\qquad \text{if } \pi_i[0] < 0 \end{cases} \tag{2.12}$$

In this equation, $\pi_j^i[0]$ is the amount of reactive power that node i estimates to be needed by node j using equation (2.11). $\pi_j^i[0]$ is the reactive power

needed by $j \in D_i$ that i estimates using equation (2.9). $\frac{\overline{s_{ij}}}{\sum_{k \in N_j^p \cup j} \overline{s_{kj}}}$ splits $\pi_j^i[0]$ among neighbors of j according to their sensitivities. Thus $\frac{\overline{s_{ij}}}{\sum_{k \in N_j^p \cup j} \overline{s_{kj}}} \times \pi_j^i[0]$ is the fraction of $\pi_j^i[0]$ that i should produce.

In order to satisfy needs of all of $j \in D_i$, maximum of $\frac{\overline{s_{ij}}}{\sum_{k \in N_j^p \cup j} \overline{s_{kj}}} \times \pi_j^i[0]$ of all the $j \in D_i$ is assigned to $\pi_i[0]$. This would be able to regulate back voltages of the b_i and its disconnected neighbors to the permitted operational range. Now that $\pi_i[0]$ is initialized, b_i follows the DVC algorithm with initialized values and communicates with $b_j \in N_i^c$.

Local Controller Algorithm

This controller takes the control of the system when the node is a totally disconnected node. Thus, it cannot communicate with any of its neighbors which means that $\|N_i^c\| = 0$ and $\|N_i^p\| = \|D_i\|$. Following formula is the way to calculate the reactive power for b_i.

$$\pi_i[0] = \begin{cases} max(\{\pi_j^i[0] \frac{\overline{s_{ij}}}{\sum_{k \in N_j^p \cup j} \overline{s_{kj}}} : b_j \in D_i\} \cup \pi_i[0] \cup \pi_i^{max}) \\ \qquad\qquad \text{if } \pi_i[0] \geq 0 \\ min(\{\pi_j^i[0] \frac{\overline{s_{ij}}}{\sum_{k \in N_j^p \cup j} \overline{s_{kj}}} : b_j \in D_i\} \cup \pi_i[0] \cup \pi_i^{min}) \\ \qquad\qquad \text{if } \pi_i[0] < 0 \end{cases} \tag{2.13}$$

In this equation, $\pi_j^i[0]$ is the amount of reactive power that node i estimates to be needed by node j using (2.11). $\pi_j^i[0]$ is the reactive power needed by $j \in D_i$ that i estimates using (2.9). $\frac{\overline{s_{ij}}}{\sum_{k \in N_j^p \cup j} \overline{s_{kj}}}$ splits $\pi_j^i[0]$ among neighbors of j according to their sensitivities. Thus $\frac{\overline{s_{ij}}}{\sum_{k \in N_j^p \cup j} \overline{s_{kj}}} \times \pi_j^i[0]$ is the fraction of $\pi_j^i[0]$ that i should produce.

In order to satisfy needs of all of $j \in D_i$, maximum of $\frac{\overline{s_{ij}}}{\sum_{k \in N_j^p \cup j} \overline{s_{kj}}} \times \pi_j^i[0]$ of all the $j \in D_i$ is assigned to $\pi_i[0]$. This would be able to regulate back voltages of the b_i and its disconnected neighbors to the permitted operational range.

Note that in (2.13) this node does not participate in the iterative algorithm, and therefore above calculated $\pi_i[0]$ is actually the reactive power that this

node will produce at the end of cycle. Hence it cannot be more than π_i^{max}. However in (2.12), the $\pi_i[0]$ is not the final amount of reactive power that node i is going to produce. The node will use $\pi_i[0]$ calculated in (2.12) as initial value to start the iterative DCV algorithm and finally will converge to its final reactive power.

The algorithm suggested for local controller is based on the assumption that neighbor nodes have similar situations; i.e. two physical neighbors cannot be at the over voltage and under voltage situation at the same time. This is a valid assumption in power networks [26].

Proof of Effectiveness

When communication channels are working properly in the power system, if one of the nodes has extra reactive power request for voltage limit violation correction, all the other nodes in the network are aware of it and can provide the reactive power if they have extra capacity. In other words, it means that the capacity which can be exploited to regulate voltage limit violation is the sum of capacity of all the nodes in cyber-physical network to which violating node is connected. However when a node cannot communicate with network, it is only limited to its own reactive power production capacity. If the required power exceeds its capacity, the node will not be able to regulate its voltage.

Following theorem shows that proposed fault resilient architecture and algorithms presented for hybrid and local controllers can still use the capacity of immediate physical neighbors for voltage regulation, even though the node is disconnected from the network. In this theorem, $\pi_j^{net}[r]$ represents the amount of total reactive power that is produced by node j at step r of operation.

Theorem 1: If node b_i is a totally disconnected node suffering from undervoltage situation, its voltage can be increased to permitted range of above $0.95V^{nominal}$ iff $\Sigma_{j\in N_i\cup i}(\pi_j^{max} - \pi_j^{net}[r]) \times s_{ij} \geq (0.95V^{nominal} - V_i)$

Proof: In undervoltage mode, V_i is smaller than $0.95V^{nominal}$. In order to regulate the voltage to the permitted range, total injected reactive power by neighbors should increase the V_i at least by $0.95V^{nominal} - V_i$. At each step m after r, following linear equation between reactive power, sensitivities and

voltage changes holds:

$$\Delta V_i[r+m] = \sum_{\{j:j\in N_i^p \cup i \wedge \pi_j^{net}[r+m] \leq \pi_j^{max}[r+m]\}} s_{ij} \times \Delta\pi_j[r+m] \tag{2.14}$$

Based on the algorithm described in the previous sections, neighbors of i will increase their reactive power production if b_i is still in undervoltage situation. Not being able to regulate the voltage of b_i means that b_i is still overvoltaged while $\{j : j \in N_i^p \cup i \wedge \pi_j^{net}[r+m] \leq \pi_j^{max}[r+m]\}$ is empty. In other words, there is no neighbor left who still has the capacity to increase its reactive power production. However, this is impossible because

$$\sum_{p=r}^{n} \Delta V_i[k] = \sum_{p=r}^{n} \sum_{\{j:j\in N_i^p \cup i \wedge \pi_j^{net}[r+m] \leq \pi_j^{max}[r+m]\}} s_{ij} \times \Delta\pi_j[p]$$

If p_j is the step in which neighbor j reaches its maximum capacity, then we have $\Delta\pi_j[p] = 0$ for $p > p_j$ then we have rest of the above equation as follows:

$$\sum_{j\in\{N_i^p \cup i\}} \sum_{p=r}^{n} s_{ij} \times \Delta\pi_j[p] = \sum_{j\in\{N_i^p \cup i\}} s_{ij} \times \sum_{p=r}^{n} \Delta\pi_j[p]$$

since all the nodes have reached their maximum capacity, we have:

$$\sum_{j\in N_i^p \cup i} s_{ij} \times \sum_{p=r}^{n} \Delta\pi_j[p] = \sum_{j\in N_i^p \cup i} s_{ij} \times \ (\pi_j^{max} - \pi_j^{net}[r])$$

Based on the assumption in the theorem 1, we have :

$$\sum_{p=r}^{n} \Delta V_i[k] \geq (0.95V^{nominal} - V_i)$$

This states that the amount of voltage increase is more than what is needed by node i to resolve the undervoltage situation.

□

Theorem 2: If node b_i is a totally disconnected node suffering from overvoltage situation, its voltage can be decreased to permitted range of under $1.05V^{nominal}$ iff $\Sigma_{j\in N_i \cup i}(\pi_j^{min} - \pi_j^{net}[r]) \times s_{ij} \leq (1.05V^{nominal} - V_i)$

Proof: The proof is similar to theorem 1. □

2.4 Test Scenarios

In this section two different communication failure scenarios are presented. Effects of each scenario on the system with unmodified decentralized voltage control algorithm and the system with our fault resilient architecture are demonstrated and voltage and reactive power graphs are presented.

Power network used for these examples is a 8-bus distribution network shown in Fig. 2.2.

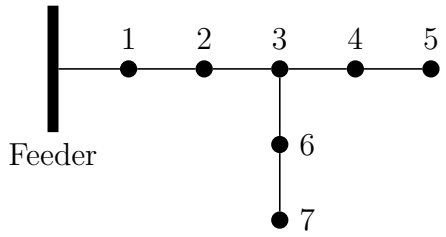


Figure 2.2: 8-Bus Distribuion Network

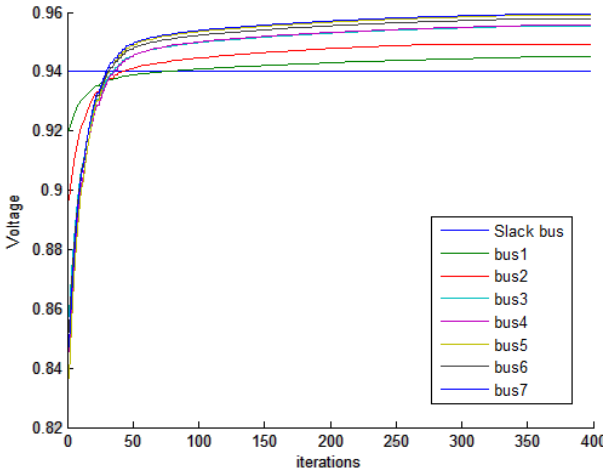
Voltage to reactive power sensitivity matrix for this network is:

$$S = 10^{-2} \times \begin{bmatrix} 0.5384 & 0.5388 & 0.5396 & 0.5398 & 0.5399 & 0.5397 & 0.5397 \\ 0.5388 & 1.3183 & 1.3201 & 1.3206 & 1.3209 & 1.3204 & 1.3205 \\ 0.5396 & 1.3201 & 2.8303 & 2.8313 & 2.8320 & 2.8308 & 2.8310 \\ 0.5398 & 1.3206 & 2.8313 & 3.6888 & 3.6897 & 2.8318 & 2.8320 \\ 0.5399 & 1.3209 & 2.8320 & 3.6897 & 5.3016 & 2.8325 & 2.8327 \\ 0.5397 & 1.3204 & 2.8308 & 2.8318 & 2.8325 & 3.5155 & 3.5158 \\ 0.5397 & 1.3205 & 2.8310 & 2.8320 & 2.8327 & 3.5158 & 4.4361 \end{bmatrix}$$

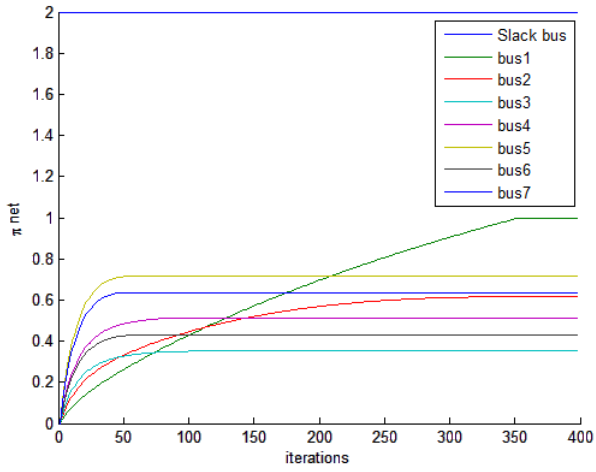
Assuming zero sensitivity for the nodes that are not immediate neighbors, we have:

$$\overline{S} = 10^{-2} \times \begin{bmatrix} 0.5384 & 0.5388 & 0 & 0 & 0 & 0 & 0 \\ 0.5388 & 1.3183 & 1.3201 & 0 & 0 & 0 & 0 \\ 0 & 1.3201 & 2.8303 & 2.8313 & 0 & 2.8308 & 0 \\ 0 & 0 & 2.8313 & 2.6888 & 3.6897 & 0 & 0 \\ 0 & 0 & 0 & 3.6897 & 5.3016 & 0 & 0 \\ 0 & 0 & 2.8308 & 0 & 0 & 3.5155 & 3.5158 \\ 0 & 0 & 0 & 0 & 0 & 3.5158 & 4.4361 \end{bmatrix}$$

For the following test scenarios, sensitivities extracted from $\overline{S}$ are used.



(a) Voltages of buses

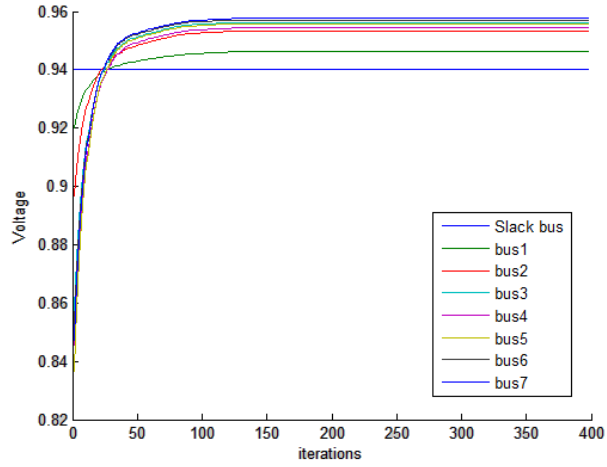


(b) Reactive power of buses

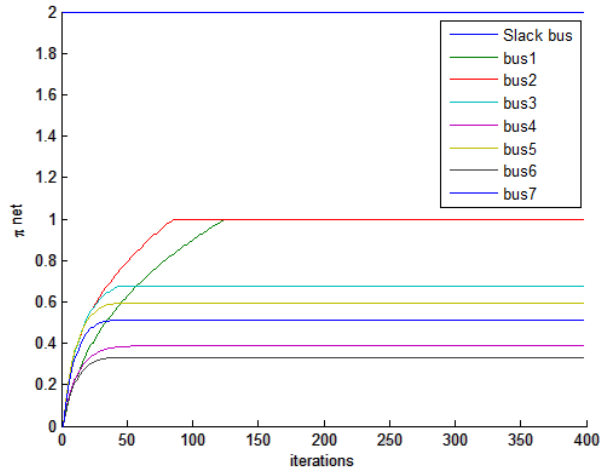
Figure 2.3: Scenario 1, b_1-b_2 and b_2-b_3 communication channel failure, DVC algorithm

2.4.1 Scenario 1, Single Communication Fault

In this scenario, two communication channels b_1-b_2 and b_2-b_3 have failed simultaneously. Once a disconnection is detected by b_1 and b_3, they remove it from N_1^c and N_3^c and add it to D_1 and D_3. At this point, b_2 removes b_1 and b_3 from N_2^c and adds them to D_2. At this point, $\|N_1^c\| = 0$ and we have $\|N_2^c\| = 0$ since both of its neighbors are disconnected. Therefore b_1 is categorized as totally disconnected nodes. For b_3, since it has $\|N_3^c\| = 2$ and $\|D_3\| = 1$ therefore it is categorized as a partially disconnected node. Rest of the nodes are still categorized as totally connected.



(a) Voltages of buses



(b) Reactive power of buses

Figure 2.4: Scenario 1, b_1-b_2 and b_2-b_3 communication channel failure, fault resilient DVC algorithm

This failure detection causes switching module in b_1 and b_2 to trigger a switch from distributed controller to the local controller and in b_3 to trigger a switch from distributed controller to the hybrid controller.

Having the sensor failures described above in the network, voltage in the slack bus drops down to $0.94V_{nom}$ which causes undervoltages in all of the nodes in the network. Fig. 2.3 and 2.4 show the voltages and reactive power productions of the nodes after this voltage drop occurs in slack bus.

Fig. 2.3 shows the voltage and reactive power production graphs when original decentralized voltage control algorithm is used. In this example voltage of slack bus is set to 0.94 which causes undervoltage in all the nodes

as seen in the initial point of the Fig. 2.4.a. As it is seen, after 350 iterations, b_1 reaches its maximum capacity while it is still in the undervoltage situation and b_2 has the capacity which can be used to increase V_1. Because there is no communication channel that b_2 and rest of network receive reactive power request of b_1.

Voltage and reactive power production graphs under fault resilient voltage control algorithm, are shown in fig. 2.4. As soon as the undervoltage occurs, nodes start to detect it and increase their reactive power production until their voltages are increased up to the permitted range. After about 80 iterations b_2 has reached its maximum reactive power production while it is still in undervoltage situation and after 120 iterations b_1 reaches its maximum capacity.

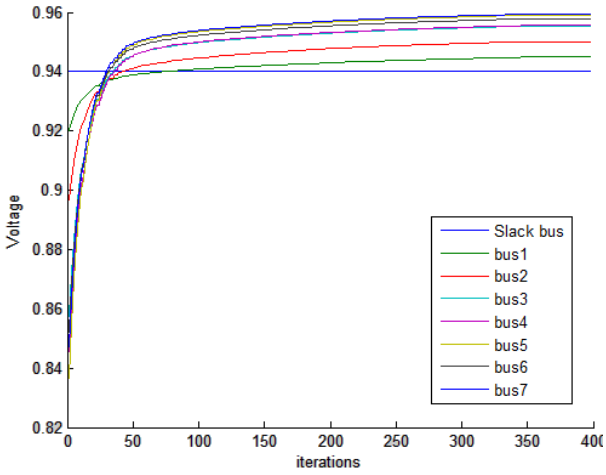
Based on *theorem*1, the capacity that can be used for voltage regulation in a totally disconnected node, is limited to its immediate physical neighbors capacity. At this point, the only neighbor of b_1 which b_2 and it has zero capacity left. Hence $\sum_{j \in N_i} s_{ij} \times \Delta\pi_j[r] = 0$ which means that no further voltage increase can be expected for b_1. However, voltage of b_2, although it has reached its maximum capacity, is still being increased. That is because, its partially connected neighbor, b_3, using estimation detects the undervoltage situation of b_2 and increases its reactive power production request until V_2 reaches the permitted voltage range of above $0.95V_{Nominal}$.

2.4.2 Scenario 2, Communication Fault

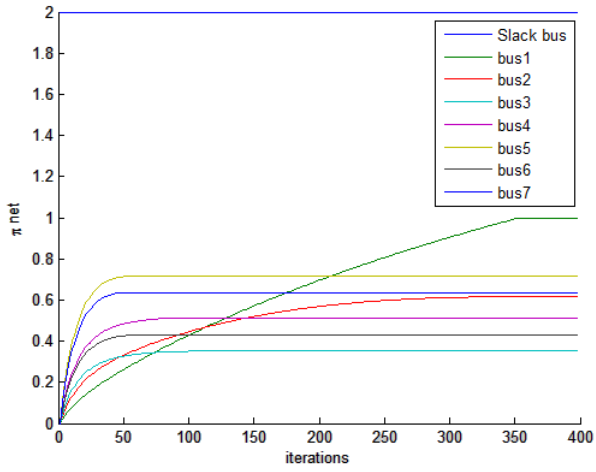
In this example, communication channel connecting b_1 to b_2 has failed which results in b_1 being disconnected from rest of the network.

Fig. 2.5, shows voltage and reactive power of the nodes under the original decentralized voltage control algorithm. Due to $0.94V^{nominal}$ voltage in slack bus, all the nodes are initially in undervoltage situation and therefore increase their reactive power production. b_1 is the only disconnected node in this network and can only exploit its own reactive production capacity in order to regulate its voltage. After 350 iterations, it reaches its maximum capacity while it is still in undervoltage situation and no further voltage increase occurs due to lack of communication.

Under the fault resilient voltage control algorithm, b_1 does not receive



(a) Voltages of buses

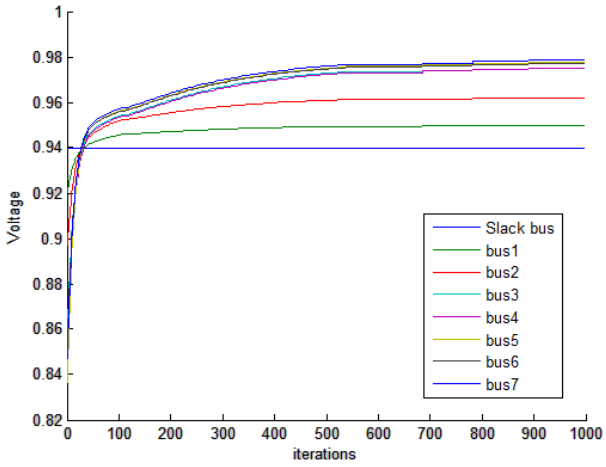


(b) Reactive power of buses

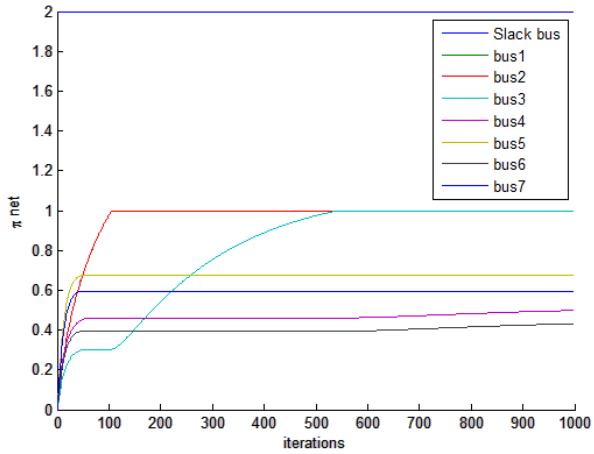
Figure 2.5: Scenario 2, b_1-b_2 communication channel failure, DVC algorithm

any communication from b_2 and therefore removes b_2 from N_1^c and adds it to D_1. Since b_2 is the only neighbor, node is categorized as a totally disconnected node after this point and switching module triggers the switch from distributed controller to local controller. Similarly, b_2 does not receive any packet from b_1 and removes b_1 from N_2 and adds it to D_2. Because N_2^c is not empty, b_2 is categorized as a partially disconnected node. Therefore, switching module switches from complex to safety controller.

As seen in the Fig. 2.6, all the nodes increase their reactive power in order to increase their voltages. After 100 iterations, b_1 and b_2 reach their maximum reactive production capacity both at the same time (reactive power production graphs of b_1 and b_2 are very close to each other) however, because



(a) Voltages of buses



(b) Reactive power of buses

Figure 2.6: Scenario 2, b_1-b_2 communication channel failure, fault resilient DVC algorithm

b_2 is a partially connected node and can estimate the undervoltage situation in b_1, following the algorithm, reports the need to the connected nodes in the network. As seen in the graph, b_3 starts to increase its reactive power production after this point in order to increase the voltage of b_1 and b_2. After b_3 reaches the maximum capacity, b_4 and b_6 start to increase reactive power production until b_1 comes back to the permitted voltage range.

2.5 Conclusion

Communication faults can cause irreparable damages to infrastructures of cyber-physical system and especially power grids. In this chapter, we tried to introduce the concept of using a combination of physical measurements and estimation methods in order to detect and handle these faults. In order to detect communication faults, the idea of estimation and comparison is applicable in all the systems that designer can find estimation methods using physical connections. However, handling these faults depends on number of the state variables of remote nodes that can be estimated.

CHAPTER 3

RUN-TIME CHECKING TO PROVIDE SAFETY AND PROGRESS

3.1 Introduction

Cyber-physical systems (CPS) combine networked communication along with interactions with the physical world. We consider a CPS scenario consisting of several embedded computing components each interacting and sensing the physical world and communicating with a central coordinator over an unreliable channel, such as wireless or the Internet. These low-level controllers attempt to accomplish some task in a coordinated fashion. Since the physical world is being manipulated, it is essential that the supervisory control logic is carefully designed and satisfies strict safety requirements. For example, autonomous vehicles may use wireless to communicate their positions and alter their future routes, but vehicles should never collide despite the potential for an unbounded number of message drops. This system is difficult to reason about because both (1) the communication layer can experience unbounded message delays and drops, and (2) the dynamics of the physical world are represented by interacting relationships in a continuous space.

An example of a distributed CPS is autonomous coordinated vehicle motion. A set of vehicles is moving through a shared physical space, and the user would like to be able to make run-time changes to the routes of the vehicles, while guaranteeing that a formation is maintained and vehicles will not collide. Since messages may be lost over wireless, any new route command may arrive at some vehicles but not at others. Acknowledgments will not solve this problem, since acknowledgments may also sometimes be lost. As in distributed systems with lossy communication, it is impossible to achieve consensus in this system [2]. Despite this inherent limitation, by using the proposed approach we can ensure the safety invariant that the vehicle flock is always maintained and collisions are avoided. If the communication channel

eventually delivers packets, we can also provide the notion of progress that gives the user the ability to safely change the routes at run-time.

In the context of a distributed CPS, a designer is typically interested in two properties: safety and progress. A proof of safety will guarantee that the system will never enter an undesirable state. We formally specify safety as a predicate on the variables of the agents of the distributed CPS which is true at all times (a safety invariant). The notion of progress that we consider is that, roughly speaking, all the agents will receive and follow a desired goal command in finite time. The ultimate guarantee that we provide is that the system will remain safe at all times (even if the network fails), while being able to meet the progress property as long as the communication network is functioning.

The scope of this chapter will be the verification of the high-level control logic of the distributed CPS, and not the verification of the individual controllers. We will therefore assume that the implementation of the individual low-level controllers is correct and bug-free. For example, upon receiving a command message, a low-level controller will follow that command as intended. Ensuring this is also non-trivial, but it is likely a more tractable problem for formal design approaches since each low-level system contains less variables than the composed system. Additionally, techniques such as the Simplex architecture [7, 29] may be used to guarantee certain behavior properties for low-level controllers, even if the complete controller is directly verifiable.

An overview of the type of distributed CPS we consider is shown in Figure 3.1. The key enabler of our safety result is the realization that, if the network is assumed to be unreliable, individual low-level controllers need be able to maintain global safety even in that case that packets do not arrive. Safety here means that a given predicate on the state space will evaluate to true over all time. We propose a Runtime Command Monitor which is interposed between the supervisory control logic and the network, as shown in the figure. If the supervisory control logic attempts to send control commands which, for any amount of message delay, can lead to a system state that violates the safety predicate, the Runtime Command Monitor will prevent the command from being sent. We show that this design results in a fail-safe system. The main technical challenge that we will elaborate on is to determine the exact behavior for the Runtime Command Monitor for a particular distributed

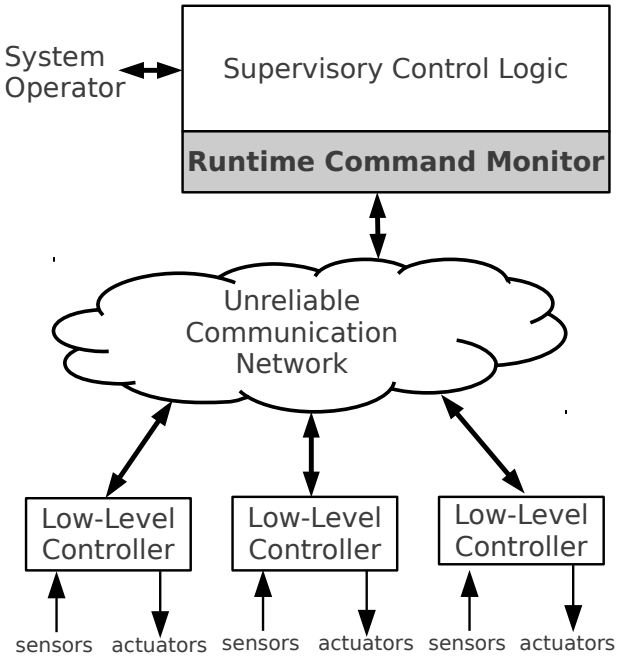


Figure 3.1: A Runtime Command Monitor ensures safety for the distributed cyber-physical system.

CPS system.

Since the network is unreliable, control commands which are sent from the supervisory logic may never arrive at a low-level controller. In order to be able to update the system behavior based on run-time information (progress), therefore, a stronger requirement must be imposed upon the network. As long as messages eventually arrive, we also provide a means to guarantee system progress while maintaining safety. To do this, we must ensure that commands that are sent out maintain the safety invariant both in the case where the command arrives and the new control strategy is used, and in the case where the command is indefinitely delayed. This notion of safe potential divergence is captured as *compatible actions*. We show that time-insensitive system progress properties can be guaranteed by constructing finite chains of compatible actions which end with the final desired system action.

The main contributions of this chapter are as follows:

- We prove that run-time properties provide necessary and sufficient conditions for safety in a distributed CPS system. By encoding these checks into a Runtime Command Monitor, a fail-safe system can be developed (Section 3.2).

- We provide sufficient conditions for providing time-insensitive progress guarantees. This requires constructing a chain of compatible actions, as well as a network which eventually delivers packets that are sent. (Section 3.3)

- We apply both of these approaches to a simulation of a coordinated vehicle flocking system. In addition to demonstrating the guarantees of safety and progress, we evaluate the overhead of the Runtime Command Monitor. (Section 3.4)

3.2 Providing Safety

In this section, we use hybrid input/output automata to formalize the notion of a distributed networked control system with arbitrary delays and packetloss. We then prove a general theorem which is both a necessary and sufficient condition for the safety of such systems. We then apply the theorem by stating the run-time checks in order to maintain system invariants, which will be encoded into the Runtime Command Monitor in the proposed architecture.

3.2.1 Hybrid I/O Automata

Hybrid input/output automata are general models for systems consisting of discrete and continuous states, where the discrete states are governed by transition rules, and the continuous states evolve according to differential equations. There is also input and output in these systems, which allows easy composition of different components into a larger system.

Rather than explaining the full semantics for hybrid I/O automata, we provide a brief overview of only the most important aspects here, and refer an interested reader to a more comprehensive review [30, 31].

A hybrid I/O automaton consists of four parts: variables, transitions, trajectories, and actions. **Variables** are the discrete or continuous entities of an automaton, for example velocity or mode. A state of an automaton is a specific valuation of the variables. **Transitions** provide the behavior of the discrete variables in the system. These have an enabling precondition and

an effect. The state after the effect is applied is called the post state of the transition. Preconditions specify when transitions can occur, but generally automata are not forced to take a transition, which can create nondeterminism. **Trajectories** give the behavior of the continuous variables in the system as time passes, typically using differential equations, and systems can also have nondeterministic dynamics described by nondeterministic differential equations. The conditions under which time can not advance are given as stop conditions, which can be used to force an enabled transition to occur. Finally, **actions** indicate the interaction points for external communication with other automata. An action will always have a corresponding transition in the automaton. An action can occur when both automata that have the action satisfy the corresponding transitions' preconditions.

Time passes for a hybrid automata when a trajectory is acting upon the continuous variables. During the execution, there can be discrete jumps in state caused by the transitions. For two hybrid I/O automata with compatible actions, say A and B, we denote their composition using $A||B$.

3.2.2 System Definition

We model our supervisory control system as a network of communicating hybrid I/O automata. In this network, there is an automaton describing the behavior of each of the N agents in the system, $A_1, A_2, \ldots, A_N$, and an automaton which models the communication channel. This model is slightly more general than the one discussed earlier with an explicit supervisory controller. Here, we could arbitrary choose one of the agents to be the supervisor.

In this section, we are concerned with verifying that a predicate is a safety invariant for a system. That is, we are provided with a safety predicate on the states of the agent automata. The predicate is an invariant if it evaluates to true for all reachable states of the system from a given initial state (an unsafe state can not be reached). A system is a composition of the agent hybrid I/O automata and the communication automaton.

For our unreliable network, we consider a communication automaton with weak guarantees about message delivery, named C_{weak}, which can delay each message arbitrarily long, or drop it. Such an automaton matches the communication properties of many networked or wireless communication systems.

```
automaton CommWeak(M : Type)
type Packet = tuple of message: M, delay: Real, dest : Nat
  variables
    bag : Bag[Packet] := [],
    now : Real := 0
  actions
    send(m: M, dest: Nat),
    receive(m: M, dest: Nat)
  transitions
    send(m, dest) // not in CommDrop
      effect
        bag := insert([m,now+rand(),dest]
    send(m, dest) // not in CommStrong
      effect
        /* dropped */
    receive(m, dest)
      precondition
        contains(bag, [m,0,dest])
      effect
        remove(bag, [m,0,dest])
  trajectories
    stop when
      ∃p: Packet p ∈ bag ∧ (now = p.deadline)
    evolve
      d(now) = 1
```

Figure 3.2: The C_{weak} communication automaton assigns messages arbitrary delays and can drop messages. Here, rand() returns a nonnegative real number.

The automaton description for C_{weak} is given in Figure 3.2. Here, there are two possible send transitions, either of which can be applied when a message is sent out. The first one assigns a real-valued arrival time greater than the current time. The second one silently drops the packet. We also will consider two other communication scenarios, C_{drop} and C_{strong}. In C_{drop}, the first send transition of C_{weak} is omitted so all messages get dropped. In C_{strong}, the second send transition is omitted, so that all messages can only be arbitrarily delayed, but never dropped. A communication automaton would be composed with each of the agent automata by connecting the receive transition with destination i to Agent A_i. All the agents would invoke the same send transition.

3.2.3 Safety Theorem

In order to prove a predicate P is an invariant for a system given a definition for each agent automaton and the communication automaton, a standard approach is to check that the invariant is satisfied for every transition and every trajectory. During this process, the invariant may need to strengthened in order for the proof to follow.

The standard approach for proving invariants, however, can be difficult to apply. Since reasoning is done ahead of time, the analysis must be applicable

to all states which can be encountered for each rule.

In this chapter, we present an alternative approach for creating invariant-satisfying systems. Here, we will use a combination of static reasoning done ahead of time along with run-time checks. With this approach, we can sometimes guarantee an invariant in an easier manner than by using the normal, static-only approach. Rather than reasoning over sets of possible values, we instead move part of the checking to run-time, and can therefore use a specific value in a specific message. In order to do this, however, we need to prove a theorem which provides an equivalent condition for verifying invariants.

A system is described by a composition of the automaton for each of the agents ($A^N = A_1||A_2||\ldots||A_N$) and the automaton for the communication channel. A property P is the predicate we are trying to show is an invariant, and is a predicate on the states of the agents, P: $A^N \rightarrow$\{true, false\}.

Theorem. A predicate P is an invariant for a system $S = A^N||C_{weak}$ if and only if (1) P is an invariant for the system $S' = A^N||C_{drop}$, and (2) from any post state of a `receive` transition in S, P is preserved by the system $A^N_{post}||C_{drop}$, where A^N_{post} is the composed agent automata A^N starting in the post state of the `receive` transition.

Proof. First we show the direction that if conditions (1) and (2) hold, the invariant is satisfied by the original system.

The proof of this statement is based on the observation that at every point in time, either no messages have been received, or there is a most-recently received message by one of the agents. As shown in Figure 3.3, for every possible trace there will be some amount of time where no messages have been received by any of the agents in the system, followed by a intervals of time where there is a most-recently received message.

Our proof proceeds by contradiction. Assume t_i is the first time at which P is evaluates to false in S. If t_i occurs before the first message is received, this means that P would also evaluate to false in S' at time t_i, since up to this point the behavior of S and S' is identical. This violates condition (1).

Therefore t_i occurs at or after a message has been received and processed. Let t_m be the time of the most-recently processed message before time t_i (the time at which the `receive` transition was invoked in C_{weak}). We apply condition (2) of the theorem at time t_m and take A^N_{post} as the composed agent automata in the post state of the `receive` transition in S. Since in S, P evaluates to false before any further messages are received after t_m, this

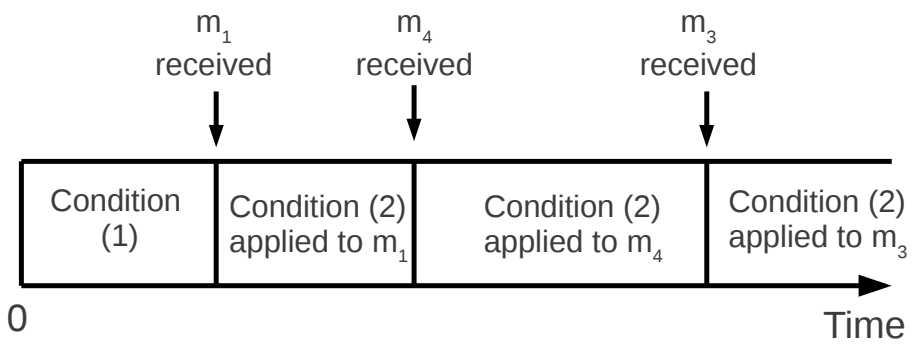


Figure 3.3: For every trace, at each time instant, either no message has been received in the system, or there is a most-recently received message.

would mean it also evaluates to false for the system with agent automata A^N_{post} and a communication automaton which does not receive any messages. This is exactly the case checked by condition (2).

Next we show the other direction, that if a predicate P is an invariant for S, conditions (1) and (2) will hold. Again, we proceed by contradiction.

Assume condition (1) does not hold but P is an invariant of S. The behaviors of C_{drop} can be exactly simulated by C_{weak}, which means that P can not be an invariant for S.

Next, assume the second case that condition (2) does not hold but P is an invariant for S. In the context of the false case of condition (2), let time t_m be the time at which the `receive` transition is invoked. Now consider a communication automaton which produces an identical behavior as S until t_m and then no longer receives messages. This behavior can also be exactly simulated by C_{weak} (by taking the dropping `send` transition for messages which would originally have an arrival time after t_m), which means that P can not be an invariant for S.

Since both cases yield contradictions, if an invariant is satisfied in the original system, conditions (1) and (2) must also hold.

The two conditions of the theorem are therefore both necessary and sufficient for proving an invariant is satisfied for a system with unreliable communication over all time.

3.2.4 Application of Theorem to Runtime Monitoring

From a static-time analysis perspective, the theorem does not gain us very much since condition (2) needs to be evaluated every time any message can be received, which is difficult to reason about. However, at run-time, condition (2) may be easier to verify. This is the approach advocated, to check condition (1) at system design time and condition (2) at run-time, which by the theorem will guarantee that P is an invariant of the system.

One challenge of this approach is that the necessary run-time analysis needs to be automated in software, which is done in our architecture in the Command Filter Safeguard. Since there may be nondeterminism from the dynamics of the agents, and since in general this may involve an infinite-time reachability computation, this may be easy or hard depending on the specific system.

In terms of applicability, one main concern that we will evaluate further in our case study in Section 3.4is the run-time overhead of the approach, which is application-specific. If we consider a typical case of time-invariant systems where low-level controllers are stable from a control-theoretic sense, and the commands are new set points, the potential area the agent may enter given some unknown delay consists of the states it will encounter while transitioning from the old set point to the new one, projected over all future time (since delay is unknown). To check condition (2), this would be computed and checked with the future states the other agents may enter against the safety predicate.

Another consideration is to specify the action to take if the analysis for the specific message indicates condition (2) is *not* satisfied at run-time. The system can not be allowed to take action based on the message, since it may lead to a state which violates the invariant. In our proposed design, these messages are filtered (never sent out) by the Runtime Command Monitor. This preserves condition (2) for the system (since no messages will be sent out unless (2) is satisfied) which guarantees that P will continue to be an invariant for the system. Of course dropping messages can adversely affect system progress, but it will only be done to maintain safety (if the predicate captures a notion of safety). In Section 3.3, we present sufficient conditions to guarantee progress which require, among other things, a stronger communication automaton, where messages can be delayed arbitrarily but not

dropped.

Since the Runtime Command Monitor drops messages at send time, it needs to reason about possible system states when the packet will be received (since condition (2) deals with the system state upon message reception, not sending). This also may be challenging because, for unrestricted systems, it involves reasoning about which messages may be sent out in the future before the arrival time of the message, and possible message reordering. For example, in Figure 3.3, message m_4 arrives before message m_3. The run-time analysis at the send time of message m_3 needs to take this possible reordering into account. Also, in an unrestricted system, these messages can be sent from and arrive at different agents (for example m_3 may be from Agent 1 to Agent 2, while m_4 is from Agent 3 to Agent 4). For specific systems, however, this analysis may be simpler. For example, systems which maintain sequence numbers in messages and only take actions on the most-recent messages received, do not have to consider reordering. Systems like the supervisory control system we are considering have a single entity which sends command messages, and therefore we do not need to reason about command messages exchanged between other agents. As matches our intuition, having guaranteed orders of packet delivery produces systems that are easier to predict and prove correct, whether using the standard static-time approach or our run-time technique. Condition (2) of the theorem demonstrates this, while, at the same time, tells us what would need to be checked for the more general case.

3.3 Guaranteeing Progress

We will now describe a manner in which we can guarantee a time-insensitive notion of safe system progress. We assume a more specific CPS model here where each agent is running a stable closed-loop controller.

First, we discuss the distributed control system architecture that we consider more specifically in Section 3.3.1. Section 3.3.2 defines the notion of compatible actions in the context of the distributed control system and proposes methods of checking compatibility. In Section 3.3.3, we then show scheme of coordinated control that guarantees safety according to our earlier result from Section 3.2. Finally, Section 3.3.4 proves progress of the system

under a stronger assumption of the communication layer.

3.3.1 Controller Architecture

As before, we consider a distributed control system consisting of a collection of N agents with a central coordinator. We assume that each agent receives commands only from the central coordinator. Each Agent A_i has a *local controller* and a variable *set point* $\mathbb{S}_i$. The set point of Agent A_i can be changed through communication with the central coordinator. In general, a set point indicates a single or a sequence of (i) actions A_i will take, or (ii) goal states A_i moving towards. For simplicity, in this section we will assume $\mathbb{S}_i$ is a single goal position of A_i. That is, the local controller of Agent A_i drives the agent's continuous variables to move towards the set point $\mathbb{S}_i$. When agent A_i reaches an ϵ-ball around the set point (for some fixed ϵ), agent A_i will report its arrival to the central coordinator be sending a progress update message. The central coordinator will then, upon receiving arrival messages from all the agents, send each agent its next set point. An execution of Agent A_i can therefore be viewed as a hybrid sequence $\eta_i = wait_i[0] \frown receive[1] \frown \tau_i[1] \frown send \frown wait_i[1] \frown receive[2] \frown \tau_i[2] \frown send \frown wait_i[2] \ldots$, where (i) each $\tau_i[k]$ is a trajectory moving to a particular set point $\mathbb{S}_i[k]$, (ii) *send* is the Agent sending the progress update message, (iii) $wait_i[k]$ is a trajectory when waiting for next set point, where agent A_i stays within the ϵ-ball of $\mathbb{S}_i[k]$, and (iv) $receive[k]$ is an action invoked by the central coordinator's send action, during which the set point of agent A_i is changed from $\mathbb{S}_i[k-1]$ to $\mathbb{S}_i[k]$. In each trajectory $\tau_i[k]$, the initial state and the final state of the trajectory are within ϵ-balls of successive set points of A_i. A global set point is defined as a collection of the local set points for each of the N agents, and is denoted as $\mathbb{S}^N$.

In this section we are concerned with progress, but the progress must be made cognisant of safety. As in Section 3.2, safety is defined in terms of a predicate P_S. The progress property is defined using a global set point $\mathbb{S}^N_{final}$. The formal notion of progress we prove is that each agent will, in finite time, reach within an ϵ-ball around its set point in $\mathbb{S}^N_{final}$, while always having P_S evaluate to true.

3.3.2 Compatibility and Stability

Section 3.2 showed that in order to ensure safety, the central coordinator needs to reason about future states of A_i, and will therefore issue set points according to the states A_i can reach. Reasoning about future states of A_i can be done using reachability analysis. We denote $Reach_i[k]$ as the set of reachable state of A_i under trajectory $\tau_i[k]$. The reachable set of the global system (the composed behavior of all the agents) is denoted as $Reach^N$. For safety of the system, we need to verify that $Reach^N$ satisfies the safety predicate P_S. Recall that a trajectory $\tau_i[k]$ of A_i depends on two set points, $\mathbb{S}_i[k-1]$ and $\mathbb{S}_i[k]$, of A_i. For a specific set point $\mathbb{S}_i[k]$, we check whether P_S remains true over the composed $Reach^N[k]$ by computing the reachable set of states for each of the other agents. This property of safety for a new global set point captures the a notion of *compatible actions*.

Definition $\mathbb{S}^N[k]$ and $\mathbb{S}^N[k+1]$ are said to be pairwise **compatible actions** if the global state $x^N \in Reach^N[k]$ always satisfies P_S when every A_i moves along a trajectory defined by $\mathbb{S}_i[k]$ and $\mathbb{S}_i[k+1]$.

The notion of compatible actions can also be generalized to n-way compatible actions. That is, given n collections of set points, we can say they are n-way compatible if the global state always satisfies P_S when every agent moves along a trajectory defined by any pair of the set points. Due the extra requirements, however, it is generally easier to construct chains of pairwise compatible actions. For this chapter we will use pairwise compatibility, and perhaps investigate applications of n-way compatible action chains in future research.

3.3.3 Safety Guaranteed Run-Time Checking

We assume low-level controllers which are *locally exponentially stable*, and start from a safe global set point.

Definition. A controller is said to be **locally exponentially stable** with respect to a set point, if there exist a neighborhood of the set point such that any trajectories starting from any state in a neighborhood of the set point, eventually converge to the set point. In addition, the distance between the trajectory and the set point decays exponentially over time.

We will now formally state the behavior of the supervisory control logic:

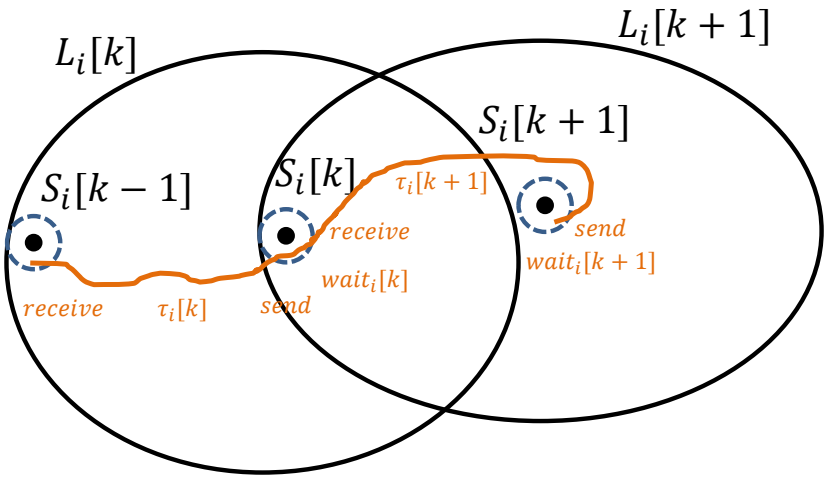


Figure 3.4: An execution trace in which $A_i$receives set points $\mathbb{S}_i[k-1]$, $\mathbb{S}_i[k]$, and $\mathbb{S}_i[k+1]$ in sequence.

(1) Until receiving progress report updates from all the agents, indicating that each agent is within an ϵ-ball of the current way point $\mathbb{S}^N[k]$, the central coordinator will not send any new set points.

(2) The server computes a new set of set point $\mathbb{S}^N[k+1]$ following conditions below, and issues them to the corresponding agents.

(2a) The global set point $\mathbb{S}^N[k+1]$ should be compatible with the global set point $\mathbb{S}^N[k]$. That is, the reach set or its overapproximation does not violate the predicate P_S.

(2b) For each agent, the ϵ-ball of its way point in $\mathbb{S}_i[k]$ should be contained by the region of attraction of its way point in $\mathbb{S}_i[k+1]$, to guarantee that the next set point will be reached by the low-level controller.

We now prove that safety predicate P_S is an invariant of the system, using the theorem from Section 3.2.

(1) If all packets get dropped at the beginning, the way points never change and P_S remains true.

(2) Suppose that after a packet get delivered, all follow-up packets get dropped. The server will stop sending new set points since not all reports are received. No agent will further update its set point since the coordinator will not send any new set points. Agent i's states will be remain in the pair-wise compatible reach set, for the current way point, forever. By pair-wise compatibility, P_S will remain true.

By (1) and (2), we conclude that P_S is an invariant of the system.

3.3.4 Progress Guarantee

We will now discuss a sufficient condition to guarantee system progress. Formally, we want the system reach a target global set point $\mathbb{S}^N_{final}$ in some finite amount of time.

To guarantee progress, we require three requirements. First, messages in the network can only get delayed arbitrarily long, but can not be dropped. For this assumption we will use automaton C_{strong}, as described in Section 3.2.2. In practice, this can be done by having a low-level network layer which keeps resending packets until an acknowledgment is received, assuming the connection will eventually get reestablished. Second, there is a finite chain of pairwise compatible actions (which we call a *compatible action chain*) from the current state to the target global set point $\mathbb{S}^N_{final}$. Third, the local controllers for each agent are exponentially stable for each set point in the compatible action chain.

We will now prove that the system $A^N||C_{strong}$ meets our progress requirement. Recall that agent A_i's execution is a hybrid trace $\eta_i = wait_i[0] \frown receive[1] \frown \tau_i[1] \frown send \frown wait_i[1] \frown receive[2] \frown \tau_i[2] \frown send \frown wait_i[2] \ldots$. First, $\tau_i[k]$ is a trajectory starting from an ϵ-ball of $\mathbb{S}_i[k-1]$ to an ϵ-ball of the $\mathbb{S}_i[k]$. Since we assumed the local controller is exponentially stable, the distance between the continuous state of A_i and the set point is exponentially decaying. Thus, any ϵ-ball of the set point will be reached in a finite time (depending on the constant in the exponential). Second, a send action through C_{strong} takes finite delivery time to invoke a receive action of the coordinator. Since this is true for all agents, the coordinator will receive all the reports of progress in a finite time. At this point the next set point will be sent back to A_i. This sending also takes a finite time since it is done by C_{strong}. Due to this, the $wait_i[k]$ trajectory where A_i is waiting for a new way point has a finite duration. Finally, since by the second requirement the chain of pair-wise compatible actions is finite, the target $\mathbb{S}^N_{final}$ is reachable through finitely many of these steps. By this reasoning, we conclude that the execution of η_i will reach $\mathbb{S}^N_{final}$ in a finite amount of time.

A system designer may want a stronger guarantee of progress that the final set point will be reached by all agents some exact amount of time (rather than just finite). In order to prove these stronger progress properties, we can adapt the same proof as above, while imposing limits on each of the

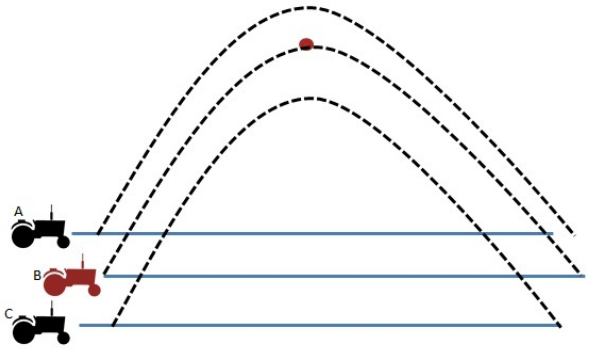
steps which were previously only required to take finite time. If the network guarantees packet delivery with a worst-case transmission time, and we know the exponential constants of our locally exponentially stable controllers, we can compute the maximum amount of time it can take for the system to go from one known way point $\mathbb{S}_i[k-1]$ to the next known waypoint $\mathbb{S}_i[k]$. Given this, we can compute the maximum amount of time for the system to complete the entire compatible action chain, by summing up the maximums for each pairs of compatible actions. In this way, the maximum amount of time that can elapse before reaching the final set point can also be calculated.

3.4 Coordinated Vehicle Flocking

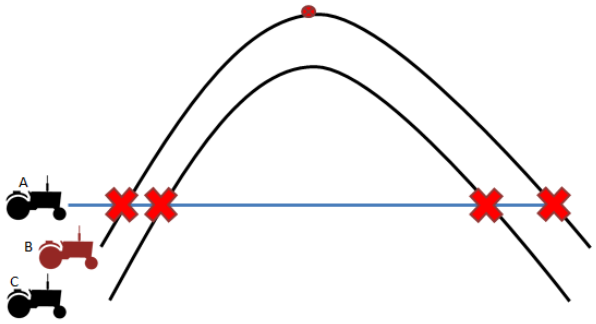
In this section, we describe a vehicle coordination case study, where a single operator controls multiple vehicles over wireless. This is inspired by experimental off-road agriculture vehicle systems currently being investigated [32]. Many agriculture tasks, such as plowing, seeding, and harvesting, require a vehicle, or a fleet of vehicles, to perform a covering of the field. By using automation, the operating cost of such a system can be reduced since less people are required to run the equipment. Additionally, productivity and efficiency may also be improved since GPS-provided actuation may be more precise than what humans would achieve on their own, and since a large number of vehicles can be used at a time.

However, since vehicles need to use wireless in order to exchange control commands, care needs to be taken to show that this unreliable component can not cause, for example, collisions. This imposes a challenge since, with unreliable communication, coming to a consensus amongst all the vehicles is impossible [2].

We consider the following system. A group of autonomous vehicles travel in formation along a path. There is a single operator in the center vehicle who can, at run time, attempt to modify the flock's route by entering a *detour point*. The new, desired path takes the flock to the detour point, and then back again to the original path. Multiple detour points can be entered during operation. A simple case showing the potential danger in such a system is shown in Figure 3.5. Here, the detour point is shown as a red circle. If packets are lost, there are four potential locations where a



(a) Dotted lines indicate desired paths.



(b) The top vehicle has not received the updated path.

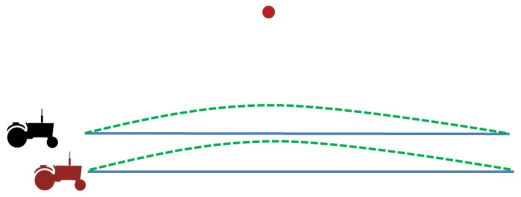
Figure 3.5: When the desired path is not received by a vehicle, collisions can result

collision can occur.

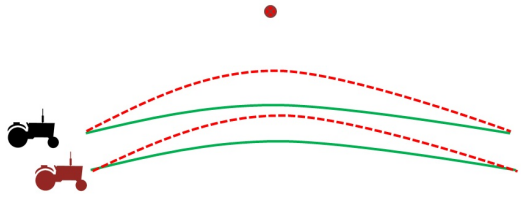
In the rest of this section, we will exploit the approaches described in the previous sections in order to create the supervisory control logic for such a system with the following guarantees:

- Vehicles do not collide with each other under packet loss or arbitrary packet delays.
- Despite packet losses or delays, all the vehicles end up in a pre-agreed location called P_{final}.
- The flock formation of the vehicles is maintained.

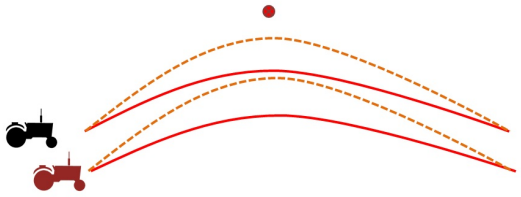
The central coordinator logic is physically on one of the vehicles which we call the leader. The leader is in charge of interacting with the operator, and generates control commands for each of the followers to be sent over wireless.



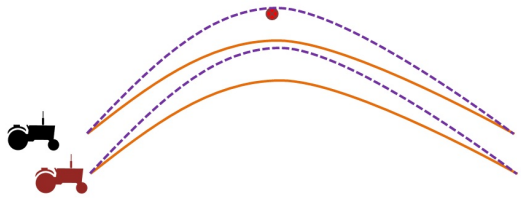
(a) interPath 1



(b) interPath 2



(c) interPath 3



(d) interPath 4

Figure 3.6: In this image, a set of intermediate paths are shown for a pair of vehicles. Dotted paths are the new paths that are generated and solid paths are the ones whose progress update has already been sent.

As described in Section 3.3, rather than immediately sending the final path to the followers, the leader will generate intermediate paths that are pairwise compatible and incrementally get closer to the desired path, as shown in Figure 3.6. In this way, wireless can be lost at any time and the system will remain safe, whereas if the wireless network works, the desired path will eventually be reached.

Every time new paths are generated by the leader, the Runtime Command Monitor will check the reachable region of each vehicle. Here, the reachable states for a vehicle are all the points that the vehicle can reach after it receives this new path. It therefore not only includes the new path of the vehicle, but also all the area in between the old path and new path that the vehicle might enter during the transition from the old path to new path. Once this region is calculated for each vehicle, if there is no intersection between the reachable regions, all the new paths can be sent out. If this check is false we generate a new intermediate path that is closer to the original path of the vehicles, and rerun the checks.

This method provides safety because no matter if the new path is received or dropped for any of the followers, the flock not collide because all the possible regions that the vehicle can reach were included in the reachable set and already checked for safety. At every step, the intermediate paths that are generated for the followers will be sent out only if all the progress update reports from all the other followers for all the previous paths have been received. As long as there is a vehicle that has not sent the progress update, the leader will keep sending the same path to the vehicle which has yet to report it is on the new path.

We will now give the technical details for the generation of a compatible action chain which can guarantee progress and safety. We first describe the generation of the desired path (which goes through the detour point), and then the computation of intermediate paths that form a pairwise compatible action chain to reach the desired path.

3.4.1 Desired Path Generation

Upon receiving a new detour point from operator, the leader will generate $desiredPath_i$ which is a path from the current position of the vehicle i to

the detour point and back from the detour point to the original path, which ends at the pre-agreed endpoint P_{final}. For instance, the last path in the Figure 3.6 generated for the leader and follower is their desired path. In our implementation, we generate Bezier curves [33] to give a smooth path that transitions from the original path, touches the detour point, and then smoothly transitions back to the original path.

3.4.2 Intermediate Path Generation

After generating the desired path for each vehicle, the leader can not generally send this path directly to followers because, similar to the situation in Figure 3.5, the current paths and the desired paths are not compatible (with respect to the safety predicate). Therefore, we iteratively use smaller substeps until we can generate a chain of compatible intermediate paths.

The path $desiredPath_i$ consists of n segments with $n+1$ way points in which $desiredPath_i[0]$ is the current position of follower f_i, $desiredPath_i[n]$ is the pre-agreed endpoint P_{final}, and one of the way points of this path equals to newly-entered detour point (or more strictly the point in the formation where follower i should be when the leader reaches the detour point). When the leader is to send out a new path, each follower will be following its current path, denoted by $currentPath_i$ which also consists of n segments and $n+1$ way points. Given this, a set of lines can connect each way point on the current path with the corresponding way point on the desired path. We define these lines as $L_i[0]$ to $L_i[n]$ such that for $0 \leq p \leq n$, $L_i[p].startPoint = currentPath_i[p]$ and $L_i[p].endPoint = desiredPath_i[p]$.

With these definitions, in order to generate the the kth way point on the mth intermediate path for follower i, called $interPath_i[m][k]$, we take the following incremental approach: $interPath_i[m+1][k] = L_i[k].start + weight * (L_i[k].end - L_i[k].start)$ where $weight \in [0,1]$ is the size of incremental step. By making the weight closer to 0, we can make the intermediate path is closer to the original path. For maintenance of the formation, we could start with a lower $weight$ value to make sure the formation is not affected too severely if some of the vehicles receive the new path and some do not.

Once $interPath_i[m+1]$ is calculated, we compute $reach_i[m]$, which is the reach set for follower i upon receiving the path update. The reach set, assum-

ing a controller which moves between way points exactly, is defined as the area between the current follower path and the potential new path (which includes all the transitions from the old paths to the new paths), bloated by the size of the vehicle. If $\forall i, j \in [0, n]$ and $i \neq j$: $Reach_i[m+1]$ does not intersect $Reach_j[m]$, then $interPath_i[m+1]$ can be sent to follower i as the new path. Otherwise we use a smaller incremental substep by taking $weight := 0.9 * weight$ and $interPath_i[m+1]$ is recalculated. This iterative recalculation will happen until an $interPath_i[m+1]$ with a compatible $Reach_i[m+1]$ is found, or a maximum number of trials is reached which indicates a chain of compatible actions could not be found.

The $Reach_i[m+1]$ sets that are computed after calculations of $interPath_i[m+1]$ are only valid if all the followers have already received $interPath_i[m]$. This is why the leader will only send $interPath_i[m+1]$ to the followers if a progress update message for $interPath_i[m]$ has already been received from each vehicle. If this is not true, the leader will not generate any new paths and will keep retransmitting the current paths until a progress update is received from all vehicles.

3.4.3 Implementation and Measurements

We have implemented the described algorithm on the mobile robot simulator for the StarL platform [34]. StarL is a Java-based programming library for developing mobile robotics applications to control Roomba robots communicating over WiFi. It includes a simulator that runs identical robot logic code, but with simulated dynamics and network delays and drops. A video of the execution on the simulator is available online [35].

We performed measurements on the implemented platform in order to better understand the overhead incurred by the Runtime Command Monitor, and the expected convergence time when using the chained compatible action approach. In our setup, we simulate a flock of vehicles with a straight initial path ending in a final way point. Initial formation of the flock is shown in the Figure 3.7 where distance of each vehicle from its neighbor is 500 units, and the radius of each vehicle is 165 units. Measurements shown in the tables are the averages from five executions.

In the first experiment we measure the effect of the distance between the

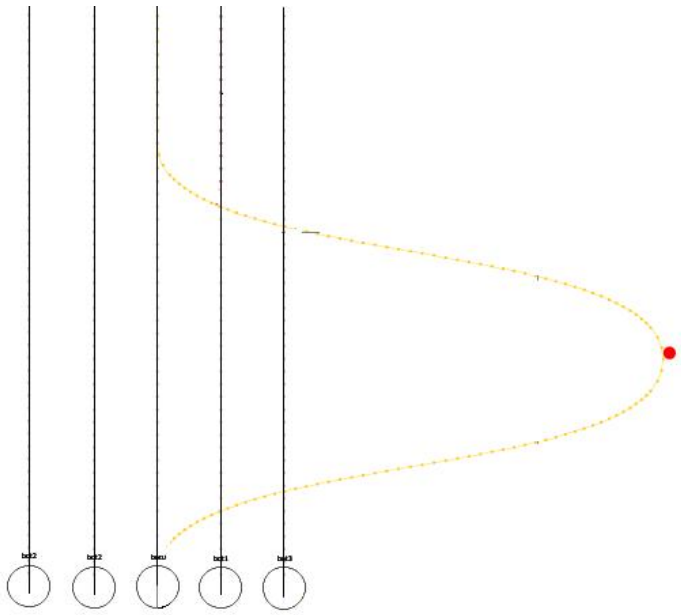


Figure 3.7: Initially the vehicles move in parallel straight lines (black). The red circle is the entered detour point, and the orange line is the desired final path for the leader vehicle.

new detour point and the initial straight path of system on the convergence time and network overhead assuming no packetloss, as well as the overhead of the check in the Runtime Command Monitor. Since a detour point further away requires a longer compatible action chain, more time is needed before the final paths are received by all the followers. This is confirmed in Table 3.1, where, as the distance of the detour point from the initial path increases, the number of messages sent and convergence time also increase. The overhead in terms of number of messages is fairly small, and there is a linear trend between the detour point deviation from the original path and the convergence time, which is expected since the length of the compatible action chains will follow a linear trend.

The overhead of the Runtime Command Monitor is related to the Intersection Checks column. During execution, paths are checked to be compatible by, for each vehicle, creating quadrilaterals from pairs of waypoints on the current path and the new path that have equal times. These quadrilaterals capture the reachable set of the vehicle transitioning from the current path to the new path. The distance between these quadrilaterals and those of neighboring vehicles is computed, and made sure to be larger than the

Detour Point	Convergence ms	Msgs Sent	Intersection Checks
(1000, 6000)	1942	21	10368
(3000, 6000)	4294	39	15488
(6000, 6000)	7464	60	22472
(7000, 6000)	9834	72	28800

Table 3.1: As the detour point distance increases from the initial path, compatible action chains contain more steps, which leads to both longer convergence time and more messages being exchanged. Here, an packetloss-free network was evaluated with three vehicles.

Packetloss Rate	Convergence ms	Messages Sent
0%	4850	45.0
20%	7312	57.6
40%	13000	94.2
60%	26818	176.8

Table 3.2: As packetloss increases, both convergence time and total number of message sent increases.

vehicle diameter. The Intersection Checks column records the number of these checks, each on a pair of quadrilaterals. This number of checks is done whenever a new intermediate path is sent to all the vehicles.

Network quality is another parameter that affects performance of the overall system. In this experiment we have measured the effect of network quality on convergence time and the number of messages that need to be sent by leader to each follower until all the vehicles safely converge to the desired path.

In this experiment, the flock consists of three vehicles (1 leader, 2 followers) moving on the straight offline path and a detour point is entered in (5000, 15000). Measurements are shown in Table 3.2. Since, in addition to command messages, acknowledgment messages are also being dropped, when packetloss rate is α, the probability of both a message and its acknowledgment being delivered is $(1-\alpha)(1-\alpha)$ which complies with the quadratic trend seen in the table.

In the last experiment we aim to evaluate the scalability of our approach as we increase the number of vehicles. For this purpose, we have performed the experiment with different numbers of vehicles and the results are presented in Table 3.3. Here, the detour point again was (5000, 15000).

Under ideal communication where packets are not dropped, we notice no

Vehicles	Conv (ms)	Msgs Sent	Conv (30%)	Msgs (30%)
3	4853.3	45	9023.3	69.3
4	4853.3	60	11808.3	105.0
6	4876.0	90	12968.0	159.0
8	4900.0	120	15036.0	228.6

Table 3.3: Under ideal communication (0% packetloss, left), the number of vehicles does not affect convergence time. However, with packetloss (30% packetloss, right), more robots leads to a larger convergence time as the algorithm needs to wait for packets to be retransmitted before continuing.

change in convergence time when the number of vehicles increases, and the number of messages sent scales linearly. This is because, when no messages are dropped, all communication is done in parallel so there is no delay experienced.

In the case where there is 30% packetloss, however, we do see an effect as we increase the number of vehicles. This is roughly because, with a fixed chance to miss a packet and a larger number of packets to send, there is a higher chance that one of the followers will miss the packet or the leader will miss the acknowledgment. When this occurs, the entire algorithm is stalled waiting until the message is resent and arrives successfully. This demonstrates one of the weaknesses of the current approach, namely that if any of the agents fails to communicate, the algorithm must wait until communication is reestablished. As future work we may investigate more robust approaches which can better handle the failure of an agent.

3.5 Conclusion

In this chapter, we have described an approach to increase resilience in a cyber-physical system from errors in the high-level control logic. Our approach, monitoring run-time commands in order to maintain a safety invariant, is general and powerful, but comes at the cost of performing part of the checking at run time. We have proven a theorem which states the exact condition that needs to be checked, in order to design the Runtime Command Monitor. Furthermore, we have used the notion of chains of pairwise compatible actions to provide time-insensitive progress guarantees under normal network conditions where messages eventually arrive at their

destination (which we have shown can be extended to provide time-aware guarantees given stronger requirements). The challenge with this approach is the application-specific task of creating a finite chain of actions, which take the system from its current state to the goal state. Both of these techniques were implemented in coordinated vehicle flocking case study, and the run-time overhead of our approach was shown to be tractable.

As future work, we may investigate extending the progress mechanism from pair-wise compatible action chains to N-way compatible action chains, which N consecutive actions of a chain can be sent out without requiring the supervisory controller to wait for a progress confirmation message. The challenge with this direction is that N-way compatible action chains may lead to more run-time overhead, and such an approach would need to be justified by an application which requires the extra flexibility. Additionally, we could consider more complicated notions of safety rather than invariants, for example temporal logic properties defined using LTL or CTL. Finally, we would like to relax the architectural requirement of a central coordinator, and instead allow distributed agents to send commands to one another as needed, while still maintaining safety and a notion of progress.

CHAPTER 4

CONCLUSION

In this thesis, we presented two sets of solutions in an attempt to solve the problem of maintaining safety while having an unreliable communication layer in CPS. However, each approach has taken a different direction to address this issue.

In the first solution, the communication channel is replaced by the data flow extracted from alternative physical sources. While this solution is not limited to any architecture (centralized or decentralized) and does not limit system progress, applying this solution to a CPS, requires expertise and familiarity with the physical aspects of the system.

In the second solution, global safety of the system was considered regardless of the physical model and behavior. in this approach we focused on defining a communication and control protocol for the system to create a balance between progress and safety. This solution is only applicable to distributed CPS with central coordinator.

Safety of distributed CPS is an ongoing topic for our research. Lots of challenges remains in this area that need to be addressed. Verification in real-time for the systems that do not have an accurate physical model is one of the interesting directions. Traditional verification solutions have a very high complexity and checking an invariant even for relatively simple systems, can take lots of computation power. Reducing this time or finding alternative solutions needs more study. Another direction that needs further investigation, is decentralized systems. Maintaining global safety in the CPS where there is no central leader, can pose lots of challenges. However, lots of real-world CPS are of this type.

REFERENCES

[1] Z. Xie, G. Manimaran, V. Vittal, A. Phadke, and V. Centeno, "An information architecture for future power systems and its reliability analysis," *Power Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 857 – 863, aug 2002.

[2] J. Turek and D. Shasha, "The many faces of consensus in distributed systems," *Computer*, vol. 25, no. 6, pp. 8–17, June 1992. [Online]. Available: http://dx.doi.org/10.1109/2.153253

[3] F. A. T. Abad, M. Caccamo, and B. Robbins, "A fault resilient architecture for distributed cyber-physical systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*. IEEE, 2012, pp. 222–231.

[4] S. Bak, F. A. T. Abad, Z. Huang, and M. Caccamo, "University of illinois at urbana-champaign, usa," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013 IEEE 19th International Conference on*. IEEE, 2013, pp. 287–296.

[5] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "Wirelesshart: Applying wireless technology in real-time industrial process control," in *RTAS '08: Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 377–386.

[6] Honeywell, "Onewireless network - isa100.11a-compliant wireless mesh network," https://www.honeywellprocess.com/en-US/explore/products/wireless/OneWireless-Network/pages/default.aspx, 2012.

[7] L. Sha, "Using simplicity to control complexity," *IEEE Softw.*, vol. 18, no. 4, pp. 20–28, 2001.

[8] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *RTSS '07*, 2007.

[9] J. Yao, X. Liu, G. Zhu, and L. Sha, "Netsimplex: Controller fault tolerance architecture in networked control systems," *Industrial Informatics, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2012.

[10] C. Kim, M. Sun, S. Mohan, H. Yun, L. Sha, and T. F. Abdelzaher, "A framework for the safe interoperability of medical devices in the presence of network failures," in *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, ser. ICCPS '10. New York, NY, USA: ACM, 2010, pp. 149–158.

[11] L. Bu, Q. Wang, X. Chen, L. Wang, T. Zhang, J. Zhao, and X. Li, "Toward online hybrid systems model checking of cyber-physical systems' time-bounded short-run behavior," *SIGBED Rev.*, vol. 8, no. 2, pp. 7–10, June 2011.

[12] J. N. Tsitsiklis, "On the stability of asynchronous iterative processes," *Theory of Computing Systems*, vol. 20, pp. 137–153, 1987.

[13] K. M. Chandy, S. Mitra, and C. Pilotto, "Convergence verification: From shared memory to partially synchronous systems," in *FORMATS*, 2008, pp. 218–232.

[14] W. Wang, Y. Xu, and M. Khanna, "A survey on the communication architectures in smart grid." *Computer Networks*, vol. 55, no. 15, pp. 3604–3629, 2011.

[15] J. McDonald, S. Rajagopalan, J. Waizenegger, and F. Pardo, "Realizing the power of data marts," *Power and Energy Magazine, IEEE*, vol. 5, no. 3, pp. 57 –66, may-june 2007.

[16] D. E. Bakken, C. H. Hauser, H. Gjermundrd, and A. Bose, "Towards more flexible and robust data delivery for monitoring and control of the electric power grid," 2007.

[17] K. P. Birman, J. Chen, K. Hopkinson, B. Thomas, J. Thorp, R. V. Renesse, and W. Vogels, "Overcoming communications challenges in software for monitoring and controlling power systems," in *Proc. of IEEE*, 2005, p. 1028.

[18] A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *Distributed Computing Systems Workshops, 2008. ICDCS '08. 28th International Conference on*, june 2008, pp. 495 –500.

[19] T. Honorable and S. Abraham, "National transmission grid study," *Energy*, no. May, pp. 9–22, 2002.

[20] "Grid 2030 - a national vision for electricity's second 100 years," U.S. Department of Energy, Office of Electric Transmission and Distribution, Tech. Rep. July, 2003.

[21] L. Sha, "Using simplicity to control complexity," *Software, IEEE*, vol. 18, no. 4, pp. 20 –28, jul/aug 2001.

[22] L. Sha, "Dependable system upgrade," in *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, dec 1998, pp. 440 –448.

[23] L. Sha, R. Rajkumar, and M. Gagliardi, "Evolving dependable real-time systems," in *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*, vol. 1, feb 1996, pp. 335 –346 vol.1.

[24] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. Kumar, "The simplex reference model: Limiting fault-propagation due to unreliable components in cyber-physical system architectures," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, ser. RTSS '07. Washington, DC, USA: IEEE Computer Society, 2007. [Online]. Available: http://dx.doi.org/10.1109/RTSS.2007.50 pp. 400–412.

[25] J. D. Glover, M. S. Sarma, and T. J. Overbye, *Power System Analysis and Design*, 4th ed. Stamford, CT: Cengage Learning, 2008.

[26] B. A. Robbins and A. D. Domínguez-García, "Part 4: Implications of the smart grid initiative on distribution engineering," Power Systems Engineering Research Center, Tech. Rep. T-41, 2011.

[27] R. A. Horn and C. R. Johnson, *Matrix Analysis*, 23rd ed. New York, NY: Cambridge University Press, 2010.

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

[29] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *RTAS '09: Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*.

[30] S. Mitra, "A verification framework for hybrid systems," Ph.D. dissertation, Massachusetts Institute of Technology, 2007.

[31] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, *The Theory of Timed I/O Automata (Synthesis Lectures in Computer Science)*. Morgan & Claypool Publishers, 2006.

[32] D. Kohanbash, M. Bergerman, K. M. Lewis, and S. J. Moorehead, "A safety architecture for autonomous agricultural vehicles," in *American Society of Agricultural and Biological Engineers Annual Meeting*, July 2012.

[33] F. Yamaguchi and F. Yamaguchi, *Curves and surfaces in computer aided geometric design*. Springer-Verlag Berlin, 1988.

[34] A. Zimmerman and S. Mitra, "Stabilizing robotics programming language (starl)," Tech. Rep., https://wiki.cites.uiuc.edu/wiki/ display/MitraResearch/StarL.

[35] F. Abdi, "Distributed safe flocking algorithm over the unreliable network," http://fardinabdi.com/node/13, 2012.