Fall 2011

CEEME: compensating events based execution monitoring enforcement for Cyber-Physical Systems

Thoshitha T. Gamage

Department: Computer Science

CEEME: COMPENSATING EVENTS BASED EXECUTION MONITORING ENFORCEMENT FOR CYBER-PHYSICAL SYSTEMS

by

THOSHITHA THANUSHKA GAMAGE

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

2011

Approved by

Bruce McMillin, Advisor
Ali Hurson
Sriram Chellappan
Wei Jiang
Mariesa Crow

ABSTRACT

Fundamentally, inherently observable events in Cyber-Physical Systems with tight coupling between cyber and physical components can result in a confidentiality violation. By observing how the physical elements react to cyber commands, adversaries can identify critical links in the system and force the cyber control algorithm to make erroneous decisions. Thus, there is a propensity for a breach in confidentiality leading to further attacks on availability or integrity. Due to the highly integrated nature of Cyber-Physical Systems, it is also extremely difficult to map the system semantics into a security framework under existing security models. The far-reaching objective of this research is to develop a science of self-obfuscating systems based on the composition of simple building blocks. A model of Nondeducibility composes the building blocks under Information Flow Security Properties. To this end, this work presents fundamental theories on external observability for basic regular networks and the novel concept of "*event compensation*" that can enforce Information Flow Security Properties at runtime.

ACKNOWLEDGEMENT

Looking back at all these years, it amazes me how fortunate I have been to be able to associate with so many outstanding and wonderful people. My sincere and eternal gratitude goes out to my advisor, Dr. Bruce McMillin, for giving me this rare opportunity to work with him and being extremely patient and supportive even during times when my productivity was low. His guidance made graduate school an incredibly rewarding and painless experience for me.

I am grateful to Dr. Ali Hurson, Dr. Mariesa Crow, Dr. Sriram Chellappan, and Dr. Wei Jiang for accepting to serve in my Ph.D. committee. I additionally want to thank Dr. Ali Hurson for the numerous opportunities he gave me to stand tall among other graduate students in the Computer Science department.

Thomas Roth (Tom) is simply a rare combination of smarts, talents, and friendship. His genuine support in converting all my raw ideas into working models helped me accelerate my research. I appreciate the support of my readers Tom, Stephen Jackson, and Ravi Akella. I owe thanks to my lab-mates for providing me with the best working environment and nurturing my thought process with all their ideas. Also acknowledged are Rhonda Grayson and Dawn Davis for all their help throughout my stay at the Missouri S&T.

This work was supported in part by the Future Renewable Electric Energy Distribution Management Center; a National Science Foundation supported Engineering Research Center, under grant NSF EEC-0812121 and NSF CSR award CCF-0614633, and in part by the Missouri S&T Intelligent Systems Center.

Last but not the least, I thank the most important people in my life, my family. My dear mother for raising me to be the person I am today, my brother and two sisters for sharing all the hardships we went through to get to this point in our lives, my ever supportive uncle for always having my back when I am down and low, my loving wife Lasanthi and son Evin for their enormous patience and countless sacrifices. All of this would be nothing without you all. I love you!

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

Cyber-Physical Systems (CPSs) are pure physical systems (physical processes) highly integrated with pure cyber components (computations) for the purpose of providing better resource utilization, control, fault tolerance and performance [1]. The embedded computers and communication networks in such systems govern both physical layer manifestations and the cyber layer computations, and affect how these two major components interact with each other and the outside world [2]. A CPS consists of a multitude of interacting distributed components, and, by definition, has a tight coupling between the computing components of the system and the physical components, underlying processes, and the governing policies [3]. The physical layer of a CPS, most often, is involved in moving a certain commodity (such as water, power, or energy) through a physical medium. The cyber components govern the movement of such elements using complex control algorithms, scheduling mechanisms, and protection systems. Critical infrastructures, as they advance, invariably become CPSs. These include commodity transportation and distribution systems (e.g. the national power distribution network, the national gas distribution network), factory automation, smart houses, emergency medical services, and intelligent highway systems.

A prominent feature of CPSs is that they have inherent external observability (power, gas flows, cars move, planes fly etc.) which can result in a confidentiality security violation. A direct consequence of this external observability is that sensitive cyber control settings emerge to adverse external parties as observable physical changes. These turn out to be more than mere observations; a group of external observers can deduce sensitive cyber information through a collaborative effort, ultimately leading to a cyber control event confidentiality violation [4]. Once the physical consequences of cyber actions are deduced, adversaries can cause coordinated physical actions to force the system to respond in an unwanted manner [5]. Out of the multitude of CPS research issues including discrete and continuous dynamics, models for specification, design, control, reliability, and security, this is a lesser-stressed yet emerging concern. The recently published *NISTIR 7628*

Guidelines for Smart Grid Cyber Security [6], for example, highlights the heightened potential for uncommon vulnerabilities in Smart Grid domain primarily due to its complex interconnected nature. More interconnections means increased opportunities for exploit, attacks (denial of service, malicious injections, intrusion, or hardware compromise), and confidentiality and privacy breaches. These systems draw national interest due to how dependent society and the economy are on their reliable functionality and operation.

Classically, security is organized into three principle categories. **Confidentiality** is preventing unauthorized users and parties from accessing and/or disclosing protected resources; only authorized users can access resources with accordance to their respective security clearances. **Integrity** is maintaining the authenticity and the accuracy of resources. **Availability** is the ability to use resources as desired in a usable form, in enough capacity and (for services) in a forward progress state [7].

The far-reaching objective of this research is to develop a science of self-obfuscating systems based on the composition of simple building blocks. A novel concept called "event compensation" is proposed as the runtime enforcement mechanism of self-obfuscation. In this regard, this work proposes the Compensating Events based Execution Monitoring Enforcement (CEEME) framework – a runtime enforcement mechanism capable of enforcing Information Flow Security Properties (IFPs) and providing event confidentiality for CPSs.

The notion of information flow violation describes ways in which unintended and unwanted information disclosures occur between two domain segregated user groups, namely, a High-Level Domain ($\mathcal{D}_H$) and a Low-Level Domain ($\mathcal{D}_L$). $\mathcal{D}_H$ usually has a secret to preserve that the $\mathcal{D}_L$ subjects try to acquire. Information flow is inexorably intertwined between and among cyber and physical components in CPSs and preserving them is a challenging task. The proposed work aims at integrating the information flow contained in physical commodity flows with the information flow contained in the cyber components for a unified CPS security model. The philosophy behind CEEME is to develop a security model that unifies the cyber and physical aspects of security, specifically focused on the

unique confidentiality vulnerabilities presented by CPSs, and a security mechanism to mitigate such vulnerabilities. Information flow security provides such a semantically-integrated model.

1.1. NEED AND CHALLENGES

The ability to uncover sensitive system internals has several ramifications. Not only does it allow adversaries to identify the critical components of the system, but coupled with the system semantics, it can be used to force the system into erroneous states. Exploiting how the cyber components react to sensory readings and how the physical network reacts to cyber commands can make the difference between the sound operation of a CPS and potential sabotage.

The Stuxnet worm (W32.Stuxnet), first discovered in July 2010, is a rootkit capable of modifying and injecting its own controller software into a specific range of Siemens programmable logic controllers. Once injected onto a vulnerable host controller, the worm is believed to be capable of manipulating the cyber and physical parts of the system. The worm reads and carefully changes particular process control sensor signals to prevent the infected system from shutting down due to abnormal behavior [8]. At the same time, it collects data on how the controller operates and subverts the infected controller system. Stuxnet's malicious operation was indistinguishable to the system operators from normal operation. From an operational point of view, the controllers were doing what they were supposed to be doing – receive and respond to process control sensor readings – only that these were fake signals generated by the worm. *The information flow violation due to Stuxnet went undetected under existing security models*. As of October 2010, an estimated 100,000 hosts in over 30,000 organizations from over 155 countries worldwide were infected by Stuxnet [9].

Emerging specific security risks are best exemplified by future power systems or "Smart Power Grid" systems [10,11] that result from using advanced technology to manage power and energy more intelligently. These CPSs are more susceptible to integrity violations than existing infrastructures [12]. Concern exists over even

early phases of Smart Grid deployment; Smart Metering systems [13, 14] are vulnerable to sniffing attacks and internal threats resulting in malicious command injection to home meters. More advanced Smart Grid systems, such as the FREEDM ERC [10], have much greater security risks due to their increased complexity [15]. Physical observation of smart meters, power transmission lines [16–18], power distribution lines, and usage patterns [19] can lead to overall CPS confidentiality breaches.

The proposed work *anticipates* future significant vulnerabilities in CPSs and proposes models and mitigation. By contrast, a great deal of existing critical infrastructure security work at the industrial and national lab level has been focussed on protecting cyber assets [20–26]. There is also recent work on intrusion detection within the real-time systems community [27]. However, to semantically integrate cyber security with physical security through information flow demands a look back to fundamental principles. Rather than developing piecemeal solutions by patching together different security aspects (cyber, network, physical), a more fundamental approach would be to build a new theory block by block from the bottom up.

1.2. CONTRIBUTIONS

This dissertation presents an event compensation based generalized framework to enforce Information Flow Security Properties (IFPs) in CPSs. At the core of this concept is a coordinated action-correction tuple called the "**Compensating Couple**". The idea is to compensate the observable effect of a system event (that violates information flow) by pairing it with appropriate reaction events. By executing a compensating couple in a timely and coordinated manner, the expected net observable change is either insignificant in terms of deducing sensitive information or equivalent to some other system characteristic. Thus, the objective is to obfuscate the observable effects of a system. This is not to be confused with obfuscating actual physical actions, as no amount of compensation can prevent physical observability. In this respect, the contributions of this work are to:

- Develop a semantic model to analyze confidentiality violation in CPSs. In particular, the ability to deduce sensitive system settings using inherently

observable changes is precisely characterized. A theory of information flow security specifically geared towards preserving event confidentiality in CPSs is also proposed

- Introduce a class of system properties called $\mathcal{P}$–compensate properties which are execution monitoring enforceable in CPSs. This extends the present understanding of enforceable security to the cyber-physical security domain and sets the basis for a CPS security model by composing simple building blocks into a more complex system

- Extend previous work on runtime enforceable policies by combining a predicate mechanism with the ability to inject events. This is used in formally developing an information flow based semantic enforcement mechanism for CPSs

- Extend existing enforcement mechanisms beyond the safety property requirement by proposing an event compensation based security framework. This is fundamentally a unification of cyber and physical security aspects through the shared semantics of information flow

- Formally demonstrate the unique external observation based confidentiality violations present in CPSs

1.3. THESIS ORGANIZATION

The rest of this document is organized as follows:

- Chapter 2 lists recent related work in computer security, information flow analysis and runtime enforcement

- Chapter 3 presents the theory of external observability based confidentiality violations as it applies for parallel and mix connected regular networks

- Chapter 4 presents the concept of "event compensation" including a formal definition of the compensating events based execution monitoring enforcement mechanism

- Chapter 5 is on obfuscation in mix connected regular networks and the time domain response of the compensating couple
- Chapter 6 is on System Behaviors based confidentiality model for CPSs
- Chapter 7 is on Replacement Solutions based confidentiality models for CPSs
- Chapter 8 presents the CEEME architecture including algorithms for the two proposed confidentiality models
- Chapter 9 presents preliminary results from extending the CEEME architecture to arbitrary networks with the standard IEEE 118-bus system taken as a model
- Chapter 10 is the conclusion of the work presented and future directions

2. RELATED WORK

System security has two main approaches: access control based methods and information flow based methods. A significant amount of recent literature suggests that access control is not going to be the direction for the next generation of security mechanisms. This section examines information flow based security policies and mechanisms, their feasibility in meeting modern system security needs, and enforceable security using information flow security policies.

2.1. ACCESS CONTROL VS. INFORMATION FLOW

The general security problem is to assure Confidentiality, Integrity and Availability. Traditional security models, such as HRU [28], Take-Grant [29], Bell-Lapadula [30] and Biba [31], take an access control based approach to address this problem. However, access control can only impose spatial restrictions on information and resources, and fails in preventing their propagation [32]. Moreover, there isn't a way to control how the data will be used after it has been read; access control cannot prevent *implicit* and *explicit* information flows[1].

The rationale behind information flow over access control as the preferred approach to system security is explainable using a simple example as follows. Consider a case where a $\mathcal{D}_H$ user sets a global flag to indicate that he has access to a certain sensitive piece of information. A $\mathcal{D}_L$ user can simply notice this global flag and determine that his accomplice in the $\mathcal{D}_H$ has just accessed sensitive information. This is an indirect violation of the *-property [33]; there is no direct "write-down" violation, yet the $\mathcal{D}_L$ user obtains sensitive information. Sound enforcement of confidentiality using traditional access control mechanisms, thus, requires that the access grants be given only to processes guaranteed not to leak or improperly transmit sensitive information after read [32]. Unfortunately, there is no provision in access control based mechanisms to impose such a strong flow restriction. Numerous authors have pointed out this fact, the inability of discretionary access control methods to properly safeguard security, in their work [32, 34–40].

[1] Implicit information flows are a result of indirect transfer of information (implied or derived knowledge) while the latter is a result of explicit transfers (variable assignment)

The primary focus of information flow security is to prevent unintended $\mathcal{D}_H$ (secure/private/confidential) information disclosures to the $\mathcal{D}_L$ (open/public) of systems. A violation, in this model, is the ability to deduce $\mathcal{D}_H$ information at relatively inferior security clearances. The sensitivity of information and the "communication" depend on how each process views other processes of the system. Fundamental IFPs introduced over the years can characterize and capture these flows. For this reason, information flow based security is better suited for environments with inherent observable changes, such as CPSs. Some of the most prominent IFPs are, Noninterference [41], Noninference [42] and Nondeducibility [43].

IFPs can be enforced using two main methods: static *compile time enforcement* and dynamic *runtime enforcement*. In **compile time enforcement**, security constraints and labels are attached to program objects using security annotations [44]. The compilers use these annotations to ensure secure information flow at compile time. Conceptually, **runtime enforcement** monitors a system during its execution and evaluates whether it deviates from the set of pre-identified desired behaviors or *properties*. This concept is also known as Execution Monitoring (EM) enforceability. Execution monitors work by monitoring the computational steps of an untrusted program and intervening whenever the execution is about to violate the security policy being enforced [45]. Each of the intended behaviors (or forbidden behaviors) of the system are security invariants. These are evaluated based upon a certain *security predicate*[2]. To facilitate this process, system executions are modeled as state sequences. Transitions between states capture the changes to the state variables.

While individual components of a system can preserve the desired security invariant, their composition can lead to security violations. In most systems, security is not considered a functional requirement. As a consequence, the final implementation often lacks the expected level of system security. Poor requirement analysis and design underspecification [46] are also main contributing factors. Covert channels can still exist even in well designed systems [47, 48]. Also,

[2] A predicate acts as a decision engine

the incorporation of untrusted and often malicious hosts and users makes it even harder to analyze confidentiality properties in modern systems [32].

2.2. PRELIMINARIES AND NOTATIONS

Some of the premier work on formally defining concurrent system executions was done by Alpern and Schneider [49]. They introduced the concept of *property* which, by definition, is a set of infinite state sequences known as *executions*. Lamport categorized system properties into two major classes: *Safety* and *Liveness* [50]. Informally, a safety property stipulates that "no bad thing" can happen during an execution, and liveness is a "good thing" eventually happening. Backed by this primary categorization, Schneider argued that only safety properties can be EM enforced [51]. Further, his definition of EM enforceable mechanisms terminates any execution which is about to violate the corresponding security predicate. The initial argument was that every system property is either a safety property, a liveness property or the intersection[49]. However, McLean argued that possibilistic properties, such as information flow security properties, are not preserved under refinement and they are neither safety nor liveness properties [52]. This is simply because these are sets of execution sets rather than execution sets [52, 53].

In an executing system[3], two questions are critical: (i) what system behaviors are runtime enforceable and (ii) what are the appropriate measures to take once a violation is detected. These are significant research questions given that most existing enforcement mechanisms are not designed with the interconnected, composite nature of modern systems in mind.

Let Σ^{∞} be the universe of all executions and $\Sigma^{*} \subseteq \Sigma^{\infty}$ denote the set of system executions. Each execution of this set $\sigma^{*} = < \sigma_0, \sigma_1, \ldots > \in \Sigma^{*}$ is composed of a sequence of global states $\sigma_i \in \sigma^{*}$. The subscript i denotes the i^{th} element of the sequence. A particular global state σ_i is a snapshot of the system at some particular point in execution, and consists of a set of system variables $\mathcal{V}$. The universe of system actions is the composition of input and output actions, i.e., $\Phi^{\infty} = \acute{\Phi} \cup \bar{\Phi}$. The subset $\acute{\Phi} \subseteq \Phi^{\infty}$ called input actions has the potential to influence one or more system variables $v_i \in \mathcal{V}$ and trigger state transitions. Upon a change to $\mathcal{V}$, the

[3]The system is already *"running/online/active"* when runtime enforcement is considered

system assumes a new state. Thus, an execution $\sigma^* \in \Sigma^*$ can be represented as,

$$\sigma^* \ : \ \sigma_0 \xrightarrow{\phi_1} \sigma_1 \ldots \sigma_{i-1} \xrightarrow{\phi_i} \sigma_i \ldots$$

Note that the first action ϕ_0 of each sequence is the null element which leads the execution to the initial state σ_0. Associated with each execution $\sigma^* = < \sigma_0, \sigma_1, \ldots >$ is a sequence of actions (defined as a *trace* subsequently) $\phi^* = < \phi_1, \phi_2 \ldots > \in \Phi^*$. Here, Φ^* represents the universe of action sequences. Elements of $\bar{\Phi}$ cannot change $\mathcal{V}$, but present a view of the system (effectively a view of $\mathcal{V}$) to a particular security domain. Thus, a particular action sequence ϕ is not (necessarily) exclusively comprised of input actions; $\phi_k \in \bar{\Phi} \implies \sigma_i \xrightarrow{\phi_k} \sigma_i$

By definition, a property $\Gamma \subseteq \Sigma^*$ is a finite subset of executions. As an example, consider the following definition of a safety property[49].

Definition 1 (**Safety Property**). *A property* Γ *is a safety property if and only if,*

$$\forall(\sigma^* \in \Sigma^*) \notin \Gamma \implies (\exists \beta \in \Sigma^* \ : \ \sigma[\ldots i]\beta \notin \Gamma)$$

In other words, no prefix of a valid execution can violate the safety property. For each violation there exists a distinctly identifiable point at which it occurs. This is known as the prefix-closed feature of safety properties. Once violated, the execution can no longer be extended and the violation cannot be undone[54]. $\sigma[\ldots i]$ *is the prefix of execution* σ^* *up to step* i*.*

Some classic examples for safety properties are starvation freedom, mutual exclusion, partial correctness and first-come-first-serve [49]. A closely related concept is the *security policy*. Technically, a security policy $\mathscr{P}$ is a generalized property, defined over a characteristic predicate $\hat{\wp}$. The membership of each element in $\mathscr{P}$ is determined by $\hat{\wp}$.

Definition 2 (**Security Policy**). *A set of executions* $\Pi \subseteq \Sigma^\infty$ *satisfies a security policy* $\mathscr{P}$ *if and only if,*

$$\mathscr{P}(\Pi) \implies \forall \sigma^* \in \Pi \ : \ \hat{\wp}(\sigma^*)$$

In other words, a security policy is a property when each element of the property satisfies the characteristic predicate of the policy[54].

As an example, a security policy may require that each execution not disclose a certain action before the i^{th} step.

Definition 3 (**Liveness Property**). *A property* Γ *is a liveness property if and only if,*

$$\forall \sigma^* \in \Sigma^*, \exists \beta \in \Sigma^\infty : \sigma^* \beta \in \Gamma$$

In other words, any finite system execution, irrespective of whether it violates the property or not, can be extended by appending an infinite or finite sequence, resulting in an execution satisfying the liveness property. Informally, liveness means "nothing irremediably bad can happen" during an execution [54].

2.3. INFORMATION FLOW SECURITY PROPERTIES

Consider a system with two security clearances: a $\mathcal{D}_H$ and a $\mathcal{D}_L$. In terms of information flow security properties, the ability to deduce sensitive $\mathcal{D}_H$ information at $\mathcal{D}_L$ is a security violation. The term deduce is a generalization for different ways information flow security can be compromised. For example, a $\mathcal{D}_L$ entity might be allowed to notice that some $\mathcal{D}_H$ activity is going on so long as they are not able to identify it precisely. In another case, $\mathcal{D}_L$ users might need to be kept totally unaware of $\mathcal{D}_H$ activities. As a result, there are different information flow security properties defined for different information flow violations.

The "possibilistic" nature of information flow security properties is a formalization of how an (potentially $\mathcal{D}_L$) observer with sufficient knowledge of the target system can deduce confidential information. This is achieved by constructing (and refining) the set of all possible system behaviors consistent with his set of observations. Mantel distinguished two dimensions of possibilistic security; that the occurrence of a confidential event (i) does not increase possible observations and (ii) does not decrease possible observations at lower clearance levels[55, 56]. It is an information flow security violation if the target system increases or decreases the possibility of observations after a $\mathcal{D}_H$ activity.

Consider the operation of the target system as a nondeterministic state automata $\mathcal{M} = (\mathcal{D}, \mathcal{Q}, \acute{\Phi}, \delta, \mathcal{Q}_0)$. $\mathcal{Q}$ is the set of automata states, $\acute{\Phi}$ is a set of input actions/events, $\delta : \mathcal{Q} \times \acute{\Phi} \rightarrow 2^{\mathcal{Q}}$ is the state transition function and $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is

the set of initial states of the automata. $\mathcal{D}$ is a set of security domains/clearances traditionally, $\mathcal{D}_H$ and $\mathcal{D}_L$.

Definition 4 (**System Automata**). *The system automata $\mathcal{M}$ consists of 5-tuples $(\mathcal{D}, \mathcal{Q}, \acute{\Phi}, \delta, \mathcal{Q}_0)$ where,*

- *$\mathcal{Q}$ is a set automaton states*
- *$\mathcal{Q}_0$ is a set of initial states for the automaton $\mathcal{Q}_0 \subseteq \mathcal{Q}$*
- *$\acute{\Phi}$ is a set of input symbols*
- *δ is the a state transition function $\delta : \mathcal{Q} \times \acute{\Phi} \rightarrow 2^{\mathcal{Q}}$*
- *$\mathcal{D}$ is the set of security domains*

Only a subset of actions $\Phi_{obs} \in \Phi^{\infty}$ (irrespective of input/output distinction) can be observed at any given security domain. Observations are mapped to security domains through specific functions. These functions act as information extractors from the state space. The *trace function* $\zeta : \Sigma^* \rightarrow \Phi^*$ produces the action sequence $\phi^* \in \Phi^*$ associated with the execution $\sigma^* \in \Sigma^*$. Consequently, the trace contains both "confidential" and "non-confidential" actions where only the latter is meant to be observed at the $\mathcal{D}_L$. The *projection function* $\rho : \Phi^* \times \mathcal{D} \rightarrow \Phi_{obs} \cap \bar{\Phi}^* \cap \mathcal{D}$ is a function which takes in a trace and a security domain as input and produces a possible output sequence observable by the elements of $\mathcal{D}$[4]. Lastly, the *purge function* $\pi : \Phi^* \times \mathcal{D} \rightarrow \Phi^* - \Phi^*_{\mathcal{D}}$ removes actions corresponding to a particular security domain from a given trace.

According to Delgado et al., runtime monitoring of safety properties fall under property type–specification languages whereas automata based runtime monitoring is listed under language type–specification languages. They present a thorough survey of recent literature on runtime monitoring and list a taxonomy on common elements of monitoring systems: specification languages, monitor, operational issues and event handler[57]. However, one major shortcoming of this categorization is that EM mechanisms, which they identify under monitoring, are separated from (safety) properties and automata.

[4]Some authors use the notation $\phi \mid_{\Phi'}$, the trace resulting from removing all events not in $\Phi' \subseteq \Phi^{\infty}$, which is semantically equivalent to the projection function

2.3.1. Noninterference Property. Noninterference states that the outputs observable at $\mathcal{D}_L$ stay the same even after removing all $\mathcal{D}_H$ actions. Formally, this is stated as follows:

$$\forall \sigma^* \in \Sigma^* : \zeta(\sigma^*) = \phi^* : \rho(\phi^*, \mathcal{D}_L) = \rho(\pi(\phi^*, \mathcal{D}_H), \mathcal{D}_L) \tag{1}$$

A system is considered *Noninterference secure* if the Equation (1) holds for its executions. In general, Noninterference means that a variation of $\mathcal{D}_H$ inputs does not cause a variation in $\mathcal{D}_L$ outputs[32]. This property is considered one of the conceptually hardest to implement on most real world systems. If the value of a public system variable depends on a private system variable, then for that particular system, Noninterference does not hold[58].

2.3.2. Noninference Property. Noninference is a more general form of Noninterference because it can be directly applied to nondeterministic systems [52]. Here, a $\mathcal{D}_L$ cannot deduce the fact that progress has been made in the $\mathcal{D}_H$ computation [59]. For each valid trace of the system, if the resulting trace of purging $\mathcal{D}_H$ events is still valid in $\mathcal{D}_L$, then the system is considered *Noninference secure*. Thus, Noninference is closed under π. Formally, this is stated as follows:

$$\forall \sigma^* \in \Sigma^* : \zeta(\sigma^*) = \phi^* : \zeta^{-1}(\pi(\phi^*, \mathcal{D}_H)) \in \Sigma^* \tag{2}$$

In general, Noninference means that for every $\mathcal{D}_L$ projection, there must be a possible trace which yields the same projection with no $\mathcal{D}_H$ event occurrence [59]. Noninference is equivalent to Noninterference for deterministic systems when $\mathcal{D}_H$ outputs cannot be generated without $\mathcal{D}_H$ inputs [52]. However, inserting $\mathcal{D}_H$ inputs can influence $\mathcal{D}_L$ outputs.

2.3.3. Nondeducibility Property. Nondeducibility is the most relaxed property out of the three information flow properties introduced. Nondeducibility states that for each $\mathcal{D}_L$ projection there is more than one possible ($\mathcal{D}_H$) trace. Thus, for a Nondeducibility secure system, ρ needs to be a surjective relation for each $\mathcal{D}_L$ observation. Formally. this is stated as follows:

$$\forall \phi^* \in \Phi^*, \exists \psi^* \in \Phi^* \mid \psi^* \neq \phi^* \cap \rho(\phi^*, \mathcal{D}_L) = \rho(\psi^*, \mathcal{D}_L) \tag{3}$$

2.4. RELATED WORK IN ENFORCEABLE SECURITY PROPERTIES

Information flow security policy enforcement is twofold: compile time enforcement and runtime enforcement. Mechanisms implemented under these two methodologies attempt to mitigate both implicit and explicit information flow violations. Explicit information flow violations occur when confidential information is directly (and possibly intentionally) passed to the public domain. In a certain way, this characteristic is similar to that of trojan horse attacks. Implicit information flows are comparatively passive in nature. The flow violations in this case are a result of the structure and/or the interactions within the system. A good example is a cyber-physical system with physically observable changes. Some of the recent approaches to enforce information flow security are *secure-type systems* [32, 34, 60], mechanisms based on *petri nets* [61–63], *process algebra and program logic* [40,46,64–70], *automata and predicates* [38,51,55,56]. For the most part, these are static compile time techniques. Here, preference is given to runtime enforcement mechanisms since it is the main focus of this discussion. Other techniques not explained in this work include *computation slicing* [71–73] and *bisimulation* [68,74–76].

2.4.1. Compile Time Enforcement Techniques. *Security-type systems* are static information flow control mechanisms where a collection of typing rules describe what security type is assigned to each expression or statement in the program [32]. A security label attached to each expression describes how each type value (such as int, double) or the statement as a whole may be read or written. This effectively enforces the information flow policy being used. For example, the decentralized label model in [34] considers a label as the pair *"owner:reader"*. Each owner of a resource or a data item has explicit power to restrict the set of readers who can access the resource under his control. The compiler ensures that the program does not violate information flow security by type checking each branch of the program. An in depth survey by Sabelfeld and Myers identified four future research directions in static information flow enforcement: (i) enriching the expressiveness of the underlying programming language, (ii) exploring the impact

of concurrency on security, (iii) analyzing covert channels, and iv. refining security policies [32].

Petri nets thrive on the idea of reachability and were originally designed to represent and analyze concurrent systems. Petri nets are a particular kind of a directed graph with two types of nodes, "places" and "transitions", and arcs connecting places to transitions. An augmented formalization of petri nets was proposed by [62,63] that can model the security requirements of a system. The author considered nodes as (input and output) channels and transitions as processes in the system. Information present on a particular channel is represented by a token. Associated with each token is a security class. Processes communicate with each other through channels by manipulating tokens, taking from one channel and placing them in another. Here, a state is the location of all the tokens at a given instance. The author characterized information flow secure systems by imposing restrictions on how transitions interact with places and tokens. More specifically, a petri net is information flow secure if the initial state and all subsequent changes, i.e., receival, sending and modifications to the set of tokens along every path of execution, are secure[63]. This work was later extended by [61] who proposed to improve this technique with direct and efficient decision techniques.

Process algebraic methods, especially those involving Communicating Sequential Processes (CSP), are another language based approach used to define information flow security. Most of the recent work in this area attempts to extend the *"Hoare Logic"*[77] with additional semantics to systematically express information flow security requirements (see [40, 64]). A quantification of information flow-based on CSP by Lowe [46] considers an information flow quantity as the number of $\mathcal{D}_H$ behaviors that are distinguishable from $\mathcal{D}_L$'s point of view. Several other notable program logic based approaches involve assertions in combination with operational and axiomatic semantics [65–68].

One prominent shortcoming of static information flow enforcement mechanisms is that they tend to be too imprecise [78]. A static enforcer can reject a program based on a partial analysis when the final outcome guarantees the enforced policy. As an example, consider the sequential program, $V_H := V_L; V_L := V_H$ with two variables $V_H \in \mathcal{D}_H$ and $V_L \in \mathcal{D}_L$ respectively. A static enforcement mechanism,

such as a security-type system, will reject this program considering the second assignment to be an information flow from $\mathcal{D}_H$ to $\mathcal{D}_L$ even if it is not a violation.

2.4.2. Runtime Enforcement Techniques. The earliest threshold on EM enforceable security policies was established in [51] by stating that only safety properties can be enforced using a monitoring mechanism. The monitor in this case, a büchi-like state automata called the *security automata*, enforces a certain security policy by terminating the target when a security violation is detected. The safety requirement, unfortunately, precludes many interesting security properties from being EM enforced. Notably, flow-based security properties are not safety properties [52, 79] and are not enforceable using Schneider's security automata. These properties are defined over sets of execution sets rather than execution sets [52]. Thus, the decision to terminate an execution can not be purely based on a detected violation on a single execution. This fundamental issue has lead to two alternative classes of resolutions:

1. Extend the capability of the security automata using additional structures or operations [38, 45, 55, 56, 80–82]

2. Develop monitors which can detect non-safety properties [54, 80, 83]

2.4.3. Extending Security Automata. Watanabe and Nagatou [38] extended the security automata with an extra structure, an emulator. The input to this augmented automata is the pair $\mathcal{I}_w : \Phi \times \mathcal{Q}$ instead of just Φ as in $\mathcal{M}$. The emulator checks to see if the outcome of each $i \in \mathcal{I}_w$ maintains the required security predicate before allowing it to run on the target system. Thus, Φ^* is refined only to include policy preserving actions/events. The underlying concept behind their extended security automata is the **unwinding theorem** [84]. The unwinding theorem allows appending safe states to an valid execution one at a time. An execution monitor explores possible next states of the security automata. However, for the physical portion of the system, exploring the next state can lead to a side effect of observable actions that cannot be undone if this next state is found to be unsafe.

Edit automata [80, 81, 85] empowers the basic security automata with additional transformational powers: truncation, suppression, insertion. Truncation

automata is quite similar to the security automata in [51]; it can only halt the execution upon detection of a violation. Suppression automata, in addition to halting, can suppress individual actions without terminating the program outright. Insertion automata can insert actions as necessary. An edit automata is the combination of these (suppression and insertion) individual automaton. Edit automata can modify the behavior of the target execution during runtime in addition to being able to interrupt it. Since there is a possibility that the action sequence resulting from a state transition has injected events, the transition function for edit automata is defined as follows: $\delta : \Phi \times \mathcal{Q} \rightarrow 2^{\mathcal{Q}} \times \Phi^*$. The state space and the capability of edit automata was refined by [82] with their finite edit automata. A process algebraic model of the truncation and edit automaton was later presented by [69,70].

A related concept to edit automata is *program-rewriting* [45]. Here, the untrusted program is transformed before execution to one which is incapable of violating the enforced security policy. The authors also characterize the class of policies enforceable with program-rewriting as the *RW-enforceable* policies and argue that program-rewriting is a generalization of EM. The computational model used to represent untrusted programs in program-rewriting is a 3-multi-tape Turing machine called a program machine (PM). The three tapes of PM are, input tape which contains information only available as the execution makes progress (ex. user input, nondeterministic choice outcomes), work tape which is the read/write area for the Turing machine, trace tape which records security relevant behavior and can be observed by enforcement mechanism.

The space of EM enforceable policies is constrained by the capability of the execution monitor. Work by [86] and [87] are focused on lightweight security mechanisms for memory constrained systems. The Büchi-like security automata technically has a full history of previously verified actions/events at its disposal when making the next grant decision. [86] considered only a shallow history in his Shallow History Automata (SHA) concept and proved that it can still express security policies such as the chinese wall policy, low-water mark policy etc. His results were also limited to prefix-closed safety type security policies and provided an information based, fine-grained characterization of EM enforceable policies. The

input to SHA is defined as $\mathcal{I}_{sha} : 2^{\mathcal{Q}} \times \mathcal{Q} \rightarrow 2^{\mathcal{Q}}$ since current state transition depends on a (finite) recent history of state transitions (effectively a trace). Bounded History Automata (BHA) [87] are similar in concept to SHA except they provide a precise characterization of the type of policies enforceable using memory constrained enforcement mechanisms.

2.4.4. Enforcing Non-safety Properties. An invalid execution $\bar{\sigma}^*$, from a safety property perspective, has some finite prefix $\bar{\sigma}[\dots i]$ which invalidates the security predicate, i.e., $\neg\hat{\wp}(\bar{\sigma}[\dots i])$ and $\forall \beta \in \Sigma^{\infty} : \neg\hat{\wp}(\bar{\sigma}[\dots i]\beta)$. However, an edit automaton can inject finite length executions at point $\bar{\sigma}_i$ in order to rectify the violation. By definition, a "corrected" execution of this form is not a safety property. Ligatti et al. characterized such correctible properties as *infinite renewal properties*[54, 80, 83]. An infinite length execution with infinitely many invalid prefixes still satisfies the renewal property as long as it has infinitely many valid prefixes. As a result, every safety property is a renewable property with an infinitely long valid prefix, and some, not all, liveness properties are also renewable properties [83].

The above characterization of non-safety properties as renewable properties allows them to be enforced using edit automata. An edit automata effectively can expand any invalid execution by injecting corrections.

2.4.5. Flow-based Property Composition. McLean was the first to formally propose a general theory of composition for flow-based confidentiality properties [52, 88]. He argued that these are closure properties and composable using a set of trace constructors called Selective Interleaving Functions (SIFs). Further, he introduced three external composition constructs: product, cascade and feedback. By definition, a SIF takes two traces as arguments and returns a third trace. There is a specific type criteria which describes how the resulting trace depends on argument traces [59] (see [52, 88]).

However, the authors of [59, 79] pointed out that McLean's framework is limited in expressibility and restricted to the interleaving trace domain. More specifically, [79] argued that SIFs cannot represent all security properties and require *a priori* knowledge of compatible traces for the composition. As remedies, they proposed a framework based on Low-Level Equivalence Set (LLES), which is a set

of traces with the same $\mathcal{D}_L$ projection. Also, they allowed their framework to be input total so the need for *a priori* knowledge isn't necessary anymore.

The LLES based framework, although is more expressive than SIFs, lacks the general correspondence between closure conditions and security [55,59]. This correspondence is critical for stepwise development of secure systems. He proposed a rich library of security predicates and a generalized framework to express information flow security properties based on the concept of the Basic Security Predicate (BSP). The elegance of this framework lies in the fact that possibilistic security properties can be represented as security predicates and every security predicate is either a single BSP or the conjunction of several BSPs.

3. THE OBSERVABILITY PROBLEM

Even with the best of security measures, the inherently observable events in modern CPSs can lead to confidentiality violations. As a motivational example, consider the subsection of a gas distribution infrastructure shown in Figure 3.1. Here, three different distributors A, B and C use **Intelligent Controllers (IQCs)** to control their respective sector of the pipeline. The pipeline network represents the physical layer of the system. Physical Actuators (PAs) are used to change the flow (*write operation*) on a particular pipeline sector and to acquire the state of the flow (*read operation*). Each PA is connected to a Distributed Grid Intelligence (DGI) [15] device, capable of controlling the PA and communicating with other DGIs. For this discussion, the PA is considered a physical layer component that interfaces with the cyber layer DGI. Similar interconnections can be found in water and power distribution networks [17]. For the purpose of notational and graphing convenience, consider $\text{IQC}_i \equiv F_i$ here onwards. Further, these two terms may be used interchangeably.

If the amount of flow in pipe sector S is taken as $\vec{f_S}$, the steady state gas flow of this system can be described in basic laws of fluid dynamics as follows:

$$\vec{f_A} = \vec{f_B} + \vec{f_C} \tag{4}$$

Whenever one distributor makes a local flow change, the other distributors can observe physical changes in flow/pressure on their respective sectors. For example, consider a case where $Distributor_B$ lowers the flow in sector B by an χ amount – using PA_B – dropping the flow in sector B to $\vec{f_B} - \chi \equiv \chi^{\downarrow}\vec{f_B}$. Unless $Distributor_A$ reacts accordingly, this $\mathcal{D}_H$ change causes an observable $\mathcal{D}_L$ flow increase in pipe section C. To maintain the flow equilibrium of the system, one or both of $\vec{f_A}$ and $\vec{f_C}$ needs to be adjusted. The space of possible aggregate adjustments $\Delta\vec{f}$ is therefore one of the followings:

$$\Delta\vec{f} = \begin{cases} \chi^{\downarrow}\vec{f}_A & \text{if } \mathrm{PA}_A \text{ reacts} \\ \chi^{\uparrow}\vec{f}_C & \text{if } \mathrm{PA}_C \text{ reacts} \\ \chi_1^{\downarrow}\vec{f}_A \textit{ and } \chi_2^{\uparrow}\vec{f}_C & \text{if both } \mathrm{PA}_A \text{ and } \mathrm{PA}_C \text{ react} \mid \chi_1 - \chi_2 \mid = \chi \end{cases}$$

The $\uparrow$ and $\downarrow$ superscripts indicate increase and decrease in flow, respectively. In a situation where $Distributor_B$ and $Distributor_C$ are two rival companies, every adjustment except $\chi^{\downarrow}\vec{f}_A$ reveals some information about $Distributor_B$'s action to $Distributor_C$. Note that there are several possible combinations for the last case of $\Delta\vec{f}$.

Consider the security partitioning of the network in Figure 3.1. $\{Distributor_A, Distributor_B\} \in \mathcal{D}_H$ and $\{Distributor_C\} \in \mathcal{D}_L$. By definition, $\mathcal{D}_L < \mathcal{D}_H$, meaning information flow should be prohibited from $\mathcal{D}_H$ to $\mathcal{D}_L$. Since PA_B initiates the first action, $Distributor_B$ is considered as a $\mathcal{D}_H$ subject. The rivalry naturally makes $Distributor_C$ a $\mathcal{D}_L$ subject; $Distributor_C$ is not supposed to know about

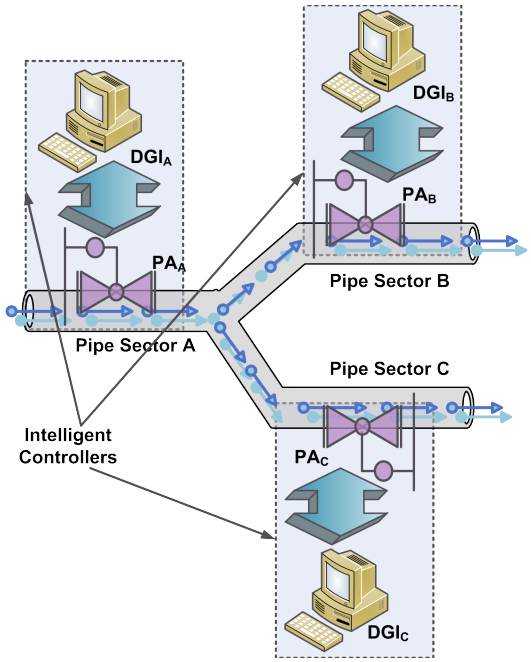


Figure 3.1: A Subsection of the Gas Distribution Infrastructure as an abstract CPS Model

$Distributor_B$'s actions[5]. For the sake of this discussion, $Distributor_A$ is considered a $\mathcal{D}_H$ subject.

Suppose at a particular instance in time, the state of the pipeline $q_k \in \mathcal{Q}$ corresponds to Equation (4). Consider the following trace initiated from q_k in Equation (5). The $read_S()$ represents a user S reading the flow (making an observation) at PA_S. The $write_S^{\uparrow/\downarrow}(\chi)$ operation represents a χ amount of increase($\uparrow$) or decrease($\downarrow$) of flow applied to PA_S. Both *read* and *write* actions are events, i.e., $read, write \in \Phi$. With respect to $Distributor_C$, $write_B^{\uparrow/\downarrow}(\chi) \notin \Phi_{obs}$.

$$\phi_o^* = \{\, read_C(), write_B^{\downarrow}(\chi), read_C(), write_A^{\downarrow}(\chi), read_C() \,\} \tag{5}$$

Now consider the projection and purged projection outputs corresponding to this trace as follows:

$$\rho(\phi_o^*, \mathcal{D}_L) = \{\, \vec{f_C}, \chi^{\downarrow}\vec{f_C}, \vec{f_C} \,\} \tag{6}$$

$$\rho(\pi(\phi_o^*, text\mathcal{D}_H), \mathcal{D}_L) = \{\, \vec{f_C}, \vec{f_C}, \vec{f_C} \,\} \tag{7}$$

According to the Equation (1), ϕ_o^* violates Noninterference as the two outputs, Equations (6) and (7), are not equivalent. The definition of Noninterference prevents $\mathcal{D}_H$ subjects from interfering with what $\mathcal{D}_L$ can observe. Yet, $\rho(\phi_o^*,\mathcal{D}_L)$, at minimum, reveals enough information to derive that some change has occurred in the system; in a Noninterference secure setup, Equation (6) should equal to $\{\vec{f_C}, \vec{f_C}, \vec{f_C}\}$. The disparity between the Equations (6) and (7) is a direct consequence of the information flow restriction explained by Sutherland [43].

This example also demonstrates why treating security in a disjoint fashion fails. A pure cyber model such as an access control matrix would will have an entry to indicate that the $Distributor_B$ is not accessible to the $Distributor_C$. A confidentiality based mechanism (such as the BLP model) would state that the $Distributor_C$ should not read (actions of) $Distributor_B$. Physical access to $Distributor_B$'s facility can also be restricted to anyone associated with $Distributor_C$. Also, in terms of network security, $Distributor_B$ might already have instruments and software to

[5]The Russian-Ukranian gas crisis is an example of this type of system partition [89]

prevent any form of network oriented attack. Nevertheless, none of these security measures were able to prevent the confidential information leakage to $Distributor_C$.

3.1. OBSERVABILITY ANALYSIS

The first phase of developing a confidentiality security model for CPSs is to formally analyze the security violation of the system. The model CPS used in this paper is the advanced power transmission bus network (Smart Power Grid) augmented for reliability using intelligent, computer-controlled power electronics devices under distributed computational control called Flexible AC Transmission System (FACTS). A FACTS device is, by definition, an IQC. These devices enable control of one or more parameters of AC transmission systems and are representative of the functionality of a wide range of Physical Actuators (PAs). FACTS devices manage the operation of the grid in stressed cases to prevent failure and to increase the overall fault tolerance of the network [90].

Observability is external observers' ability to monitor/record/observe system changes. *Deducibility* is the amount of sensitive internal system information a $\mathcal{D}_L$ observer(s) can derive from observed data. Deducibility of a single observer is limited due to his narrow view of the system and differs from what a set of collaborative observers can deduce. Several key research issues related to observability analysis are:

1. What is the minimum number of observers required to derive all the $\mathcal{D}_H$ changes in the system
2. What is the optimal placement of observers in a given network

Theoretically, the minimum number of observers and their placement in the system is related to event confidentiality as follows[6]. The objective of the $\mathcal{D}_L$ observers is to develop a one-to-one correspondence (bijection) between $\mathcal{D}_H$ actions and $\mathcal{D}_L$ observations. This is achieved by identifying unique (collective) observation patterns for each sensitive $\mathcal{D}_H$ action (secret). Once such patterns are identified, the confidentiality of those actions become violated: $\mathcal{D}_L$ observers can

[6]Note that in this discussion, the $\mathcal{D}_L$ observers are considered adversaries to the system

accurately determine what $\mathcal{D}_H$ actions cause a particular observation pattern/sequence. This, from a Nondeducibility standpoint (refer Equation 3), is deducing $\mathcal{D}_H$ traces through $\mathcal{D}_L$ projections. In other words, inferring secret information from other public information [43].

The basic assumption made here is that the control settings and the FACTS changes are sensitive $\mathcal{D}_H$ information that should not be divulged to $\mathcal{D}_L$ observers. Uncovering such settings exposes the overall state, including its critical and potentially weak links. The $\mathcal{D}_L$ observers also gain power in figuring out how the cyber algorithm responds to physical changes and vice versa. The resulting attack model is closely similar to what the Stuxnet worm was alleged of attempting to achieve [91], where it hijacks the system and performs malicious tasks.

The placement of observers is also significant since observations made from some placements might end up being redundant from a deducibility perspective. Additionally, there may exist $\mathcal{D}_H$ actions that do not cause system wide observable changes. Thus, strategic placement of observers is required to record unique observation patterns. The relationship between the deducibility and the observability is a critical aspect of CPS security analysis.

Because of their ability to inject and/or absorb active and reactive power, the cyber domain control decisions of the FACTS devices eventually manifest on the physical domain as flow changes in the power lines. Prior work showed that an external observer capable of measuring flow changes can deduce what the local action was on a particular power line and infer the overall state of the system [16]. However, the minimum number of observers problem has not yet been addressed as determining this number characterizes the confidentiality of the network.

Figure 3.2 is an experimental 20-bus power distribution network topology including five parallel connected FACTS devices. This illustration emphasizes the cyber-physical interaction and the physical layer observations found in a typical CPS. The physical layer consists of transmission lines, generators, and PAs. The cyber layer usually consists of a network of embedded computers, each capable of controlling certain parameters of the physical layer[7]. For example, the FACTS

[7] In general, cyber algorithms are used to improve the overall stability, efficiency, reliability, and performance of the underlying physical infrastructure

device is a combination of a PA in the physical layer controlled by a DGI in the cyber layer. Several FACTS devices controlled by a distributed maximum flow algorithm [90] balance power flow in the event of a power line loss. Thus, the cyber layer is a complex distributed algorithm manipulating the physical flow. External observers (as in Figure 3.2) can monitor and record physical layer changes through observation. Note that a single (virtual) source and sink architecture [90] is used throughout the rest of this analysis.

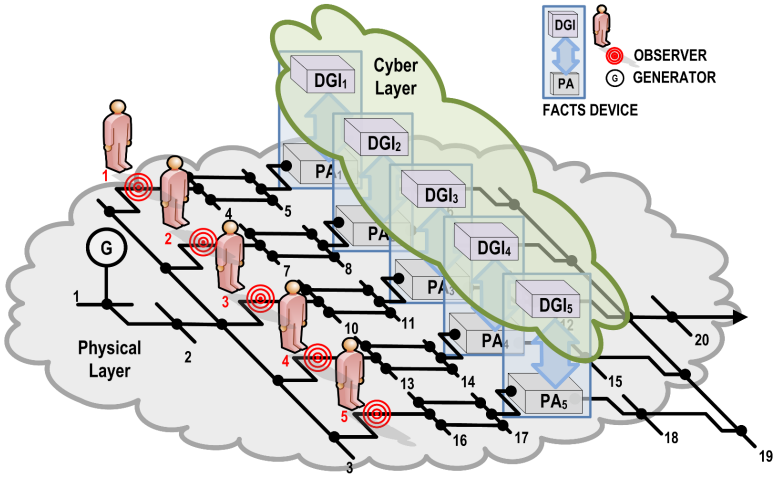


Figure 3.2: The Conceptual View of the Experimental 20-bus Network Topology – Cyber and Physical Elements with $\mathcal{D}_L$ Observers

Most flow networks under cyber control behave and exhibit similar characteristics to that of the smart power grid under FACTS control. Thus, in the rest of this document, the term FACTS is replaced with IQC when introducing new concepts, findings, and theories for generality.

3.1.1. Low-Level Domain ($\mathcal{D}_L$) Observation Matrix. A systematic and accurate $\mathcal{D}_L$ deduction requires a carefully recorded set of observations for each $\mathcal{D}_H$ action. A $\mathcal{D}_L$ *observation matrix* is a comprehensive list of system traces corresponding to each $\mathcal{D}_H$ change of a particular system. Technically, each row represents a trace $\phi_i^* = < \phi_1, \phi_2 \ldots > \in \Phi^*$ where $\phi_1 \in \acute{\Phi} \cap \mathcal{D}_H$ is the $\mathcal{D}_H$ input action resulting in the corresponding output actions $\forall j \neq 1 : \phi_j \in \bar{\Phi} \cap \mathcal{D}_L$. Thus, $\rho(\phi_i^*, \mathcal{D}_L) =$

$\{ \phi_2, \phi_3 \dots \}$ is the projection and $\pi(\phi_i^*, \mathcal{D}_H) = \{\phi_1\}$ is the $\mathcal{D}_H$ trace. An upward arrow associated with an IQC, $F_x \uparrow$, resembles an increase(tightening) of the controller variable F_x while a downward arrow $F_x \downarrow$ resembles a decrease(loosening). An arrow associated with an observer $\circledcirc_x$ as in $\circledcirc_x \uparrow$ or $\circledcirc_x \downarrow$ represents an increase or decrease in that particular observation respectively.

From Equation (3), a system is **nondeducible** secure if every $\mathcal{D}_L$ projection has multiple possible $\mathcal{D}_H$ actions. In other words, assume a system where for each $\mathcal{D}_H$ input action there are two (or more) equivalent $\mathcal{D}_L$ projections. Such a system preserves the confidentiality of $\mathcal{D}_H$ actions because $\mathcal{D}_L$ users are unable to deduce $\mathcal{D}_H$ traces based on $\mathcal{D}_L$ projections.

3.1.2. Parallel Connected Networks. In the most basic form of connectivity, two IQCs are connected in parallel as in Figure 3.3. The corresponding $\mathcal{D}_L$ observation matrix is given in Table 3.1. Note that all experimental power distribution networks presented in this article were tested using a MATLAB® Load Flow [92] simulation and all corresponding $\mathcal{D}_L$ observation matrices were constructed from simulation results.

The experimental network topologies with arbitrary number of buses used in the rest of this analysis are non-compliant to the standard IEEE test networks. They are mostly used to show or prove concepts introduced in the body of this work.

Lemma 1. *[**Partial Deducibility of Base Parallel Connected Networks**] A base parallel network of two controllers preserves Nondeducibility of $\mathcal{D}_H$ actions.*

Proof. The 9-bus experimental test feeder in Figure 3.3 preserves Nondeducibility of $\mathcal{D}_H$ actions since, as per the Equation (3), there are no unique projections present in the corresponding $\mathcal{D}_L$ observation matrix Table 3.1. For example, $F_1 \downarrow$ and $F_2 \uparrow$ both produce $\{\downarrow, \uparrow\}$ as the corresponding $\mathcal{D}_L$ projection.

The four possible $\mathcal{D}_H$ actions only result in two unique $\mathcal{D}_L$ projections. Added is the fact that, each $\mathcal{D}_L$ projection has a inverse surjective relationship with the $\mathcal{D}_H$ actions. For example, consider the following traces: $\phi^* = \{F_2 \uparrow, \circledcirc_1 \uparrow, \circledcirc_2 \downarrow\}$ and

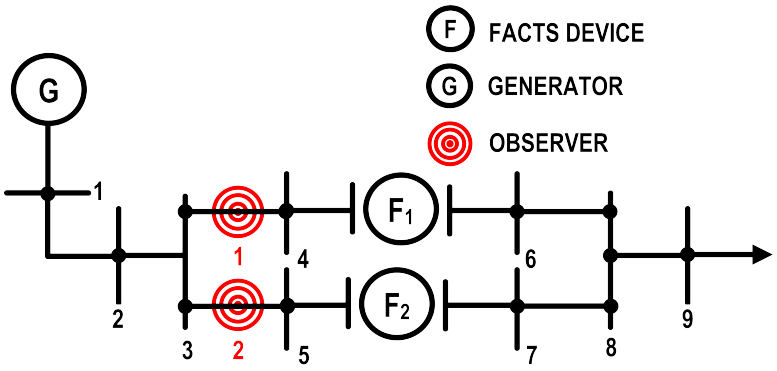


Figure 3.3: 9-bus Tree Structured Test Feeder with Two Parallel Connected Intelligent Controllers (IQCs)

Table 3.1: $\mathcal{D}_L$ Observation Matrix for the 9-bus Tree Structured Test Feeder

	$\mathcal{D}_L$ Observation	
$\mathcal{D}_H$ Change	$\circledcirc_1$	$\circledcirc_2$
$F_1 \uparrow$	$\downarrow$	$\uparrow$
$F_2 \uparrow$	$\uparrow$	$\downarrow$
$F_1 \downarrow$	$\uparrow$	$\downarrow$
$F_2 \downarrow$	$\downarrow$	$\uparrow$

$\psi^* = \{F_1 \downarrow, \circledcirc_1 \uparrow, \circledcirc_2 \downarrow\}$. Though $\psi^* \neq \phi^*$, their $\mathcal{D}_L$ projections are equal, i.e.,

$$\rho(\phi^*, \mathcal{D}_L) = \rho(\psi^*, \mathcal{D}_L) = \{\circledcirc_1 \uparrow, \circledcirc_2 \downarrow\}$$

Thus, according to Equation (3), the $\mathcal{D}_L$ observers can't fully deduce a $\mathcal{D}_H$ action of a base parallel connected controller network. However, the ability to narrow down the possible $\mathcal{D}_H$ actions to two results in a partial deducibility. □

Lemma 2. *[**Deducibility of 3-way Parallel Connected Networks**] A parallel network of three controllers is fully deducible with three observers.*

Proof. Consider the 11-bus test feeder in Figure 3.4 and the corresponding $\mathcal{D}_L$ observation matrix in Table 3.2. Here, three IQCs are connected in parallel. From

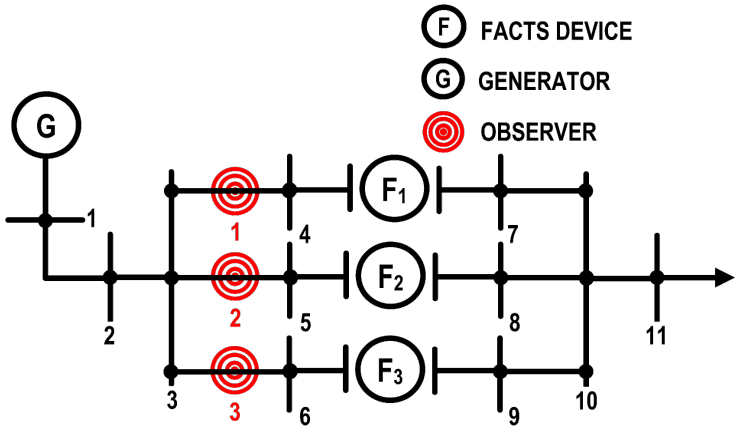


Figure 3.4: 11-bus Tree Structured Test Feeder with Three Parallel Connected IQCs

Table 3.4, all three observers are required to fully deduce the network. Combined, they can identify unique $\mathcal{D}_L$ projections for each of the six possible $\mathcal{D}_H$ actions. □

Lemma 3. *[**Minimum Number of Observers for Parallel Connected Networks**] A network of η parallel connected controllers can be fully deduced with a minimum of $\eta - 1$ number of observers, where $\eta \geq 4$.*

Proof. This claim can be proven using mathematical induction as follows.
Base case: Consider the 13-bus test feeder with $\eta = 4$ parallel connected IQCs in Figure 3.5 and the corresponding $\mathcal{D}_L$ observation matrix in Table 3.3. Evidently, any combination of three $(\eta - 1)$ observers can recognize unique $\mathcal{D}_L$ projections for each of the eight possible $\mathcal{D}_H$ actions. Thus, the claim holds for the base case.
Inductive Hypothesis: Assume that a network of η parallel connected controllers is fully deducible with $\eta - 1$ observers.
Inductive Step: Consider adding one more parallel branch to the η parallel network. This naturally results in an additional observer who is capable of observing the newly added branch. From the inductive hypothesis, the η parallel connected network is fully deduced with $\eta - 1$ minimum observers. The additional new observer results in a new minimum of $(\eta - 1) + 1 = \eta$ observers. As a consequence, a network of $\eta + 1$ parallel connected controllers can be fully deduced with a minimum of η observers. Thus the claim holds for $\eta + 1$. □

Table 3.2: $\mathcal{D}_L$ Observation Matrix for the 11-bus Tree Structured Test Feeder

	$\mathcal{D}_L$ Observation		
$\mathcal{D}_H$ Change	$\circledcirc_1$	$\circledcirc_2$	$\circledcirc_3$
$F_1 \uparrow$	$\downarrow$	$\uparrow$	$\uparrow$
$F_2 \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$
$F_3 \uparrow$	$\uparrow$	$\uparrow$	$\downarrow$
$F_1 \downarrow$	$\uparrow$	$\downarrow$	$\downarrow$
$F_2 \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$
$F_3 \downarrow$	$\downarrow$	$\downarrow$	$\uparrow$

3.1.3. Mix Connected Networks. The next natural progression of system connectivity is to consider additional levels of parallelism. In doing so, devices in different levels exhibit series connectivity with each other while devices in the same level exhibit parallel connectivity. From here onwards, such networks are referred to as *mix* connected networks.

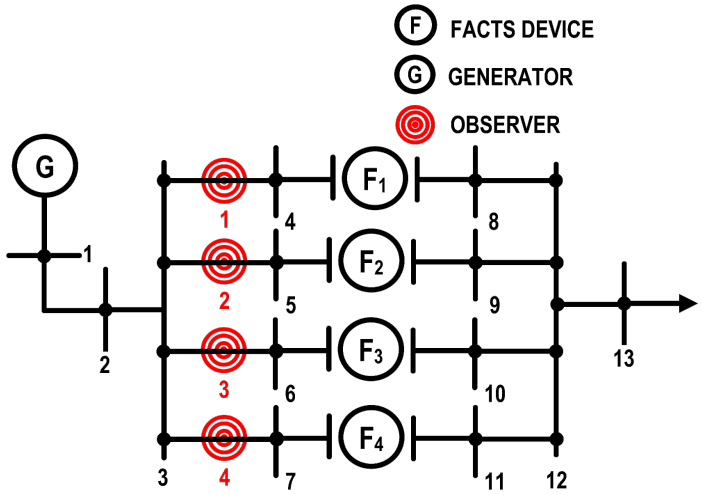


Figure 3.5: 13-bus Tree Structured Test Feeder with Four Parallel Connected IQCs

Table 3.3: $\mathcal{D}_L$ Observation Matrix for the 13-bus Tree Structured Test Feeder

	$\mathcal{D}_L$ Observation			
$\mathcal{D}_H$ Change	$\circledcirc_1$	$\circledcirc_2$	$\circledcirc_3$	$\circledcirc_4$
$F_1 \uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_2 \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$
$F_3 \uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$
$F_4 \uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$
$F_1 \downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_2 \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$
$F_3 \downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$
$F_4 \downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$

Definition 5. *[**Mix Connected Network** $\mathcal{N} = (F, \circledcirc)$] A network of IQCs, F with a corresponding set of associated observers $\circledcirc$ is defined mix connected when the elements in the network resemble both simple series and simple parallel connectivity relationships.*

Mix connected networks exhibit the typical ancestry relationship found in regular tree structures. Consider the 19-bus test feeder in Figure 3.6 of seven mix connected controllers and the corresponding $\mathcal{D}_L$ observation matrix in Table 3.4. Each subtree has a parent node in series with two child nodes connected parallel to each other; in subtree #01 for example, F_2 is in series with F_4 and F_5 while F_4 and F_5 are parallel to each other based on the flow direction (from source bus 1 to sink bus 19). Consequently, controllers in the same level have a *sibling* relationship while between levels they have a *parent-child* relationship. In terms of connectivity, Figure 3.1 directly correlates any individual subtree in Figure 3.6.

Any subtree in Figure 3.6 represents the basic connectivity unit for a mix connected network. The $\mathcal{D}_L$ observation matrix for individual subtrees is presented in Table 3.5. The deducibility of a basic mix connected network is similar to that of a 3-parallel connected network. Unique $\mathcal{D}_L$ projections exist in Table 3.5 for each of the six possible $\mathcal{D}_H$ actions. Therefore, a minimum combination of three observers can fully deduce each subtree.

One key difference between the $\mathcal{D}_L$ projections in Tables 3.2 and 3.5 is that in the latter, there is a clear correlation between the parent and child observers,

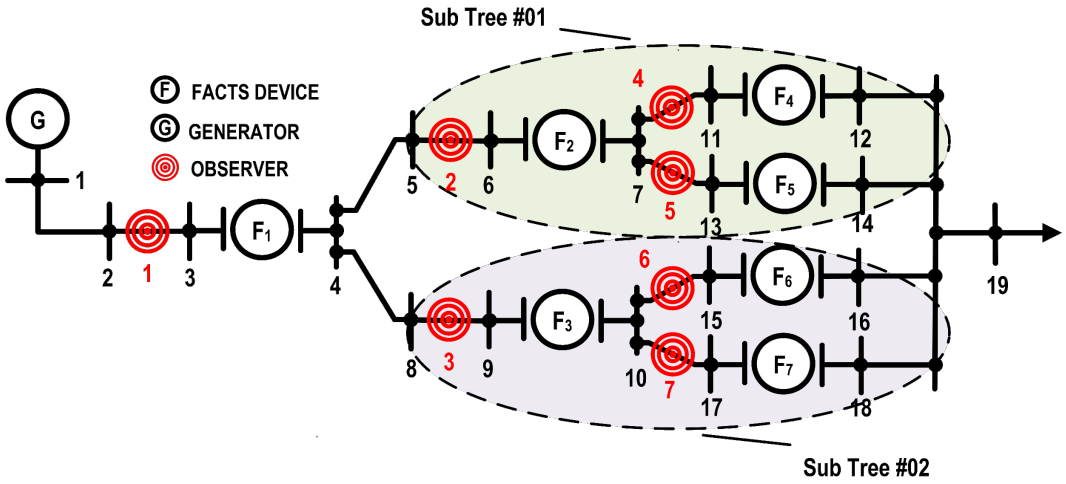


Figure 3.6: A 19-bus Tree Structured Test Network of Seven Mix Connected IQCs

Table 3.4: $\mathcal{D}_L$ Observation Matrix for the 19-bus Mix Network in Figure 3.6

	$\mathcal{D}_L$ Observation						
$\mathcal{D}_H$ Change	$\circledcirc_1$	$\circledcirc_2$	$\circledcirc_3$	$\circledcirc_4$	$\circledcirc_5$	$\circledcirc_6$	$\circledcirc_7$
$F_1 \uparrow$	↓	↓	↓	↓	↓	↓	↓
$F_2 \uparrow$	↓	↓	↑	↓	↓	↑	↑
$F_3 \uparrow$	↓	↑	↓	↑	↑	↓	↓
$F_4 \uparrow$	↓	↓	↑	↓	↑	↑	↑
$F_5 \uparrow$	↓	↓	↑	↑	↓	↑	↑
$F_6 \uparrow$	↓	↑	↓	↑	↑	↓	↑
$F_7 \uparrow$	↓	↑	↓	↑	↑	↑	↓
$F_1 \downarrow$	↑	↑	↑	↑	↑	↑	↑
$F_2 \downarrow$	↑	↑	↓	↑	↑	↓	↓
$F_3 \downarrow$	↑	↓	↑	↓	↓	↑	↑
$F_4 \downarrow$	↑	↑	↓	↑	↓	↓	↓
$F_5 \downarrow$	↑	↑	↓	↓	↑	↓	↓
$F_6 \downarrow$	↑	↓	↑	↓	↓	↑	↓
$F_7 \downarrow$	↑	↓	↑	↓	↓	↓	↑

and between sibling observers when an associated controller is making the change. Definitions 6 and 7 generalize these behaviors for regular mix connected networks.

Definition 6. *[**Inheritance through Replication**] An observer $\circledcirc_y$ inherits observer $\circledcirc_x$ through replication if the former mimics the observation of the latter in direction. For example, if $\circledcirc_x$'s observation is $\circledcirc_x \downarrow$, an observer inheriting this change through replication would produce $\circledcirc_y \downarrow$.*

Definition 7. *[**Inheritance through Flipping**] An observer $\circledcirc_y$ inherits observer $\circledcirc_x$ through flipping if the former mimics the observation of the latter in **opposite** direction. For example, if $\circledcirc_x$'s observation is $\circledcirc_x \downarrow$, an observer inheriting this change through flip would produce $\circledcirc_y \uparrow$.*

The two inheritance characteristics are common to all mix connected networks. They are primary indicators of series characteristics in a mix connected network formalized in Theorem 1.

Table 3.5: $\mathcal{D}_L$ Observation Matrix for Subtree #01/Subtree #02 in Figure 3.6

	$\mathcal{D}_L$ Observation		
$\mathcal{D}_H$ Change	$\circledcirc_2/\circledcirc_3$	$\circledcirc_4/\circledcirc_6$	$\circledcirc_5/\circledcirc_7$
$F_2\uparrow/F_3\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_4\uparrow/F_6\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$
$F_5\uparrow/F_7\uparrow$	$\downarrow$	$\uparrow$	$\downarrow$
$F_2\downarrow/F_3\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_4\downarrow/F_6\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$
$F_5\downarrow/F_7\downarrow$	$\uparrow$	$\downarrow$	$\uparrow$

Theorem 1. *[**Series Characteristics of Mix Connected Networks**] For mix connected networks, series connected parent and child observers inherit through replication while parallel connected siblings inherit through flipping.*

Proof. ***Base case 01:*** Consider the basic mix network in subtree #01 in Figure 3.6 and the associated $\mathcal{D}_L$ matrix in Table 3.5. The $\mathcal{D}_H$ action $F_2\uparrow$ produces a $\mathcal{D}_L$ projection $\{\circledcirc_2\downarrow, \circledcirc_4\downarrow, \circledcirc_5\downarrow\}$. Here the parent observation $\circledcirc_2\downarrow$ is replicated in both child observers. For $F_4\uparrow$ which results in the projection $\{\circledcirc_2\downarrow, \circledcirc_4\downarrow, \circledcirc_5\uparrow\}$,

$\circledcirc_4 \downarrow$ (the observer associated with F_4) is replicated in its parent observer as $\circledcirc_2 \downarrow$ but flipped in the sibling $\circledcirc_5 \uparrow$. This behavior is true for all the parent-child and sibling relationship in Table 3.5

Base case 02: Consider the extended mix network in Figure 3.6 and the associated $\mathcal{D}_L$ matrix in Table 3.4. Here also, the parent node $\circledcirc_1$'s change forward propagates to every child as a replication (see projections for $\circledcirc_1 \uparrow$ and $\circledcirc_1 \downarrow$). For $\circledcirc_2$ and $\circledcirc_3$, replicated changes back propagates to the parent $\circledcirc_1$ and to the immediate children ($\circledcirc_2 := \circledcirc_4, \circledcirc_5$ and $\circledcirc_3 := \circledcirc_6, \circledcirc_7$) while siblings see flipped changes; $\circledcirc_2 \uparrow \implies \circledcirc_3 \downarrow$ and vice-versa. Thus, the claim holds for both base cases.

Inductive Hypothesis: Assume the claim holds for a mix connected network of η number of controllers.

Inductive Step: There are two possibilities to add a new controller to the η mix connected network (i). Add the $\eta + 1$ controller in series, or (ii). Add the $\eta + 1$ controller in parallel with an existing controllers. Either way, this new node will inherit replicated changes from its immediate parent (or donate to immediate children), or inherit flipped changes from one of his siblings. Thus the claim holds for a $\eta + 1$ mix connected controllers. □

3.1.4. Partial Deducibility of Parallel Connected Networks. While not impossible, observing a large and complex network spread over a vast geographical area is hard. Any number of observers below the minimum requirement (refer to Lemma (3)) will not be able to fully deduce the system, but can still partially deduce some $\mathcal{D}_H$ actions.

Consider the $\mathcal{D}_L$ observation matrix in Table 3.6, corresponding to the 15-bus parallel connected network in Figure 3.7. From Lemma (3), any combination of four observers can fully deduce all 10 $\mathcal{D}_H$ traces. Examining Table 3.6 reveals that the observation on the line connected to the device making the change is clearly different from the rest of the observations. As an example, in trace $< F_1 \uparrow, \circledcirc_1 \downarrow, \circledcirc_2 \uparrow, \circledcirc_3 \uparrow, \circledcirc_4 \uparrow, \circledcirc_5 \uparrow >$ where F_1 is causing the $\mathcal{D}_H$ change, only $\circledcirc_1 \downarrow$ breaks the general trend; all other observations are $\uparrow$. This trend transpired to higher order[8] networks leads to the following theorem on partial deducibility in parallel connected networks.

[8]Networks with additional parallel branches in an increasing order

Theorem 2. *[**Partial Deducibility**] For a network of η parallel connected controllers, $\rho : 3 \leq \rho < (\eta - 1)$ number of observers can deduce ρ number of controllers.*

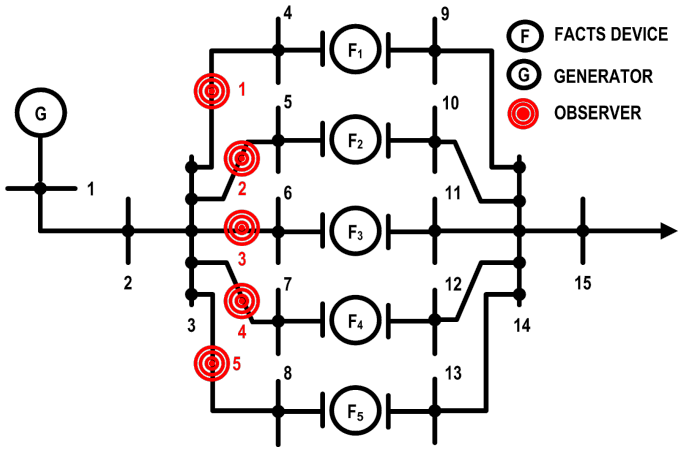


Figure 3.7: An Experimental 15-bus Tree Structured Test Network with Five Parallel IQCs

Proof. ***Base case 01:*** $\eta = 4, \rho\ (3 \leq \rho < 3) = 3$. From Lemma (2) a network of four parallel connected controllers is *fully deducible* with three observers. Obviously, these three observers can deduce three $\mathcal{D}_H$ controllers. Thus, the claim holds for base case 01.

Base case 02: $\eta = 5, \rho\ (3 \leq \rho < 4) = 3$. From Lemma (2) a network of five parallel connected controllers is *fully deducible* with four observers. Discarding one of these observers prevent prevents the deduction of two controllers. Consider the observer combination $\circledcirc_1, \circledcirc_2, \circledcirc_3$ from Table 3.6. These three observers can detect unique $\mathcal{D}_L$ projections for F_1, F_2 & F_3 but F_4 & F_5 both produce $\{\uparrow, \uparrow, \uparrow\}$. Similarly, any other combination of three observers have the same difficulty of deducing all but two $\mathcal{D}_H$ controllers – three observers can deduce only three controllers for a network of five parallel controllers. Thus, the claim holds for the base case 02.

Inductive Hypothesis: Assume the claim holds for a network of η parallel connected controllers and $3 \leq \rho < (\eta - 1)$ observers.

Inductive Step: Consider a network of $\eta + 1$ parallel controllers after adding an additional parallel branch to a η parallel network. From the inductive hypothesis, $3 \leq \rho < \eta - 1$ observers can deduce $\rho = 3, 4, \ldots, (\eta - 2)$ controllers. From Lemma (2) a network of η parallel connected controllers is fully deducible with $\eta - 1$ observers. In other words, $\eta - 1$ observers can deduce η controllers in a $\eta + 1$ parallel connected controller network. This means that $\rho = 3, 4, 5, \ldots, (\eta - 1)$ observers can deduce $3 \leq \rho < (\eta - 1)$ controllers in a $\eta + 1$ parallel connected controller network. Thus, the claim holds for a network of $\eta + 1$ parallel connected controllers. □

Table 3.6: The $\mathcal{D}_L$ observation matrix for the Experimental 15-bus Tree Structured Test Network with Five Parallel IQCs

	$\mathcal{D}_L$ Observation				
$\mathcal{D}_H$ Change	$\circledcirc_1$	$\circledcirc_2$	$\circledcirc_3$	$\circledcirc_4$	$\circledcirc_5$
$F_1 \uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_2 \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_3 \uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$
$F_4 \uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$
$F_5 \uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$
$F_1 \downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_2 \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_3 \downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$
$F_4 \downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$
$F_5 \downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$

Combined, Lemma (2) and Theorem (2) define the ***total deducibility*** of a parallel connected network of η controllers. One special case worth mentioning in Table 3.3 is that any combination of two observers cannot deduce a single $\mathcal{D}_H$ action. Thus, Theorem (2) does not apply for $\eta = 4$ and $\rho = 2$. Yet, any combination of two observers can deduce one $\mathcal{D}_H$ controller in a network of three parallel

connected controllers. As an example, the observer combination $\circledcirc_1, \circledcirc_2$ from Table 3.2 can deduce F_2; $F_2 \uparrow = \{\uparrow, \uparrow\}$ and $F_2 \downarrow = \{\downarrow, \downarrow\}$, are both unique projections.

3.2. OBSERVATION MATRICES FOR MIX CONNECTED NETWORKS

A mere set of observations is not sufficient to develop an accurate $\mathcal{D}_L$ observation matrix unless the observers determine the correlation between their observations and the corresponding $\mathcal{D}_H$ action(s). The question becomes how to generate the $\mathcal{D}_L$ observation matrix for any given mix connected network without a rigorous simulation. The answer lies in Theorem 1.

Theorem 1 provides a systematic approach to accurately derive a $\mathcal{D}_L$ observation matrix by figuring out the overall topology of the system and identifying series and parallel connectivity. Given a mix connected network $\mathscr{N} = (F, \circledcirc)$ with a set of IQCs F and associated observers $\circledcirc$, Algorithm 1 [93] goes through each controller's $\mathcal{D}_H$ change and assigns the corresponding observation to each element in $\circledcirc$. In essence, Algorithm 1 is generalizing the $\mathcal{D}_L$ observation matrix, one trace $\mathscr{T}$ at a time.

Consider the expanded experimental power distribution network in Figure 3.8. This 61-bus test feeder consists of fourteen IQCs arranged as a binary tree structure with a depth of 3. Based on Theorem 1, a $\mathcal{D}_L$ observation matrix for the system in Figure 3.8 can be constructed using Algorithm 1.

First to be assigned is the observer associated with the controller making the change (e.g. $\circledcirc_3$ for F_3). Tightening a controller variable $F_x \uparrow$ results in a decrease in flow $\circledcirc_x \downarrow$ on the observed line, and loosening $F_x \downarrow$ causes an increase in flow $\circledcirc_x \uparrow$. This basically defines the generalized $\neg \updownarrow$ notation; the negation of $\updownarrow$ is simply a flip in observation direction defined as follows.

$$\neg \updownarrow = \begin{cases} \downarrow & \text{if } \updownarrow \equiv \uparrow \\ \uparrow & \text{if } \updownarrow \equiv \downarrow \end{cases}$$

The functions $ProcC()$, $ProcS()$, and $ProcP()$ are used to process child observers, sibling observers, and parent observers respectively. For an IQC $F_x \in F$, all child node observers rooted at F_x inherit $\circledcirc_x$'s change through replication. This is handled by function $ProcC()$. Siblings of F_x inherit through flipping, which is

Algorithm 1: $\mathcal{D}_L$ Observation Matrix Construction

input : A mix connected network $\mathcal{N} = (F, \circledcirc)$ where F is the set of controllers and $\circledcirc$ is the set of corresponding observers
output: An $\mathcal{D}_L$ observation matrix $\mathcal{M}$

```
begin
    foreach f ∈ F do
        foreach D_H change ↕ of f do
            T := f ↕, ⊙_f ¬↕      /* ¬↕ is the negation of ↕ */

            /* f_C is the set of all nodes in the subtree rooted at f.
            Child nodes inherit through replication                  */
            if f_C ≠ ∅ then
                T ⟵ T ⋃ procC(f_C, ¬↕)
            end
            /* f_S is the set of all siblings of f.  Siblings
            inherit through flip                                     */
            if f_S ≠ ∅ then
                foreach s ∈ f_S do
                    T ⟵ T ⋃ procS(s, ↕)
                end
            end
            /* f_P is the parent node of f.  Parent inherits
            through replication                                      */
            if ∃ f_P then
                T ⟵ T ⋃ procP(f_P, ¬↕)
            end
            M ⟵ M ⋃ T
        end
    end
    Return M
end
```

handled by function *ProcS*(). Recursively, any remaining series connected parents inherit through replication, siblings through flipping, and child nodes inherit through replication from their respective root. This is handled by function *ProcP*().

Function ProcC($X, \updownarrow_X$)

input : A set of child nodes X and a $\mathcal{D}_H$ change $\updownarrow_X$
output: Projection of X
begin
  $\mathcal{T}_c$:=<>
  foreach $x \in X$ **do**
    $\mathcal{T}_c \longleftarrow \mathcal{T}_c \bigcup \circledcirc_x \updownarrow_X$
  end
  return $\mathcal{T}_c$
end

Function ProcS($y, \updownarrow_Y$)

input : A sibling node y and a $\mathcal{D}_H$ change $\updownarrow_y$
output: Projection of the subtree rooted at y
begin
  $\mathcal{T}_s := \circledcirc_y \updownarrow_Y$
  /* y_C is the set of all nodes in subtree rooted at y */
  if $y_C \neq \emptyset$ **then**
    $\mathcal{T}_s \longleftarrow \mathcal{T}_s \bigcup$ ProcC(y_C, $\updownarrow_Y$)
  end
  return $\mathcal{T}_s$
end

As an example, consider $F_3 \uparrow$ in Figure 3.8 as the $\mathcal{D}_H$ change, forcing the immediate and trivial consequence of $\circledcirc_3 \downarrow$. Through replication, all the child nodes of the subtree rooted at F_3 and the immediate parent observer $\circledcirc_1$, inherit $\{\circledcirc_1 \downarrow, \circledcirc_7 \downarrow, \circledcirc_8 \downarrow\}$. Observer $\circledcirc_4$ who is the lone sibling of $\circledcirc_3$ inherits through flipping ($\circledcirc_4 \uparrow$). The child nodes of the subtree rooted at F_4 all inherit $\circledcirc_4$'s change through replication: $\{\circledcirc_9 \uparrow, \circledcirc_{10} \uparrow\}$. The sibling of $\circledcirc_1$ and all the child nodes in the subtree rooted at F_2 inherit the flip change of $\circledcirc_1$: $\{\circledcirc_2 \uparrow, \circledcirc_5 \uparrow, \circledcirc_6 \uparrow, \circledcirc_{11} \uparrow, \circledcirc_{12} \uparrow, \circledcirc_{13} \uparrow$ and $\circledcirc_{14} \uparrow\}$.

Function ProcP($z, \updownarrow_Z$)

input : A parent node z and a $\mathcal{D}_H$ change $\updownarrow_X$
output: Projection of z and his siblings
begin
  $\mathscr{T}_p := \circledcirc_z \updownarrow_Z$
  if $\exists\ z_P$ **then** /* z_P is the parent of z */
    $\mathscr{T}_p \longleftarrow \mathscr{T}_p \bigcup \text{procP}(z_P, \updownarrow_Z)$
  else
    return
  end
  if $z_S \neq \emptyset$ **then** /* z_S is the set of all siblings of z */
    foreach $s \in z_S$ **do**
      $\mathscr{T}_p \longleftarrow \mathscr{T}_p \bigcup \text{ProcS}(s, \neg \updownarrow_Z)$
    end
  end
  return $\mathscr{T}_p$
end

Table 3.7 is the $\mathcal{D}_L$ observation matrix of the 61-bus mix connected network in Figure 3.8, constructed using Algorithm 1. These projections were independently verified using a MATLAB® load flow simulation of the network.

3.3. COMPLEXITY ANALYSIS: OBSERVATION MATRIX

Theorem 3. *For a mix connected network $\mathscr{N} = (F, \circledcirc)$ where F is the set of controllers and $\circledcirc$ is the set of corresponding observers such that $\mid F \mid = n$ and $\mid \circledcirc \mid = m$, the cost of building the $\mathcal{D}_L$ observation matrix is $O(nm)$.*

Proof. A change in F_i results in observable changes in all m observers. If the cost of assigning a single observation is $a > 0$ where a is a constant, the total cost of assigning all $\circledcirc$ is $a * m$. Each IQC has two different changes: $F_i \downarrow$ and $F_i \uparrow$. The total number of times the assignment is carried out is $2 * n$. Thus, the total cost of building the $\mathcal{D}_L$ observation matrix is $a * m * n * 2 = b * m * n$, where b is a constant, $b > 0$, which is bound by $O(nm)$. □

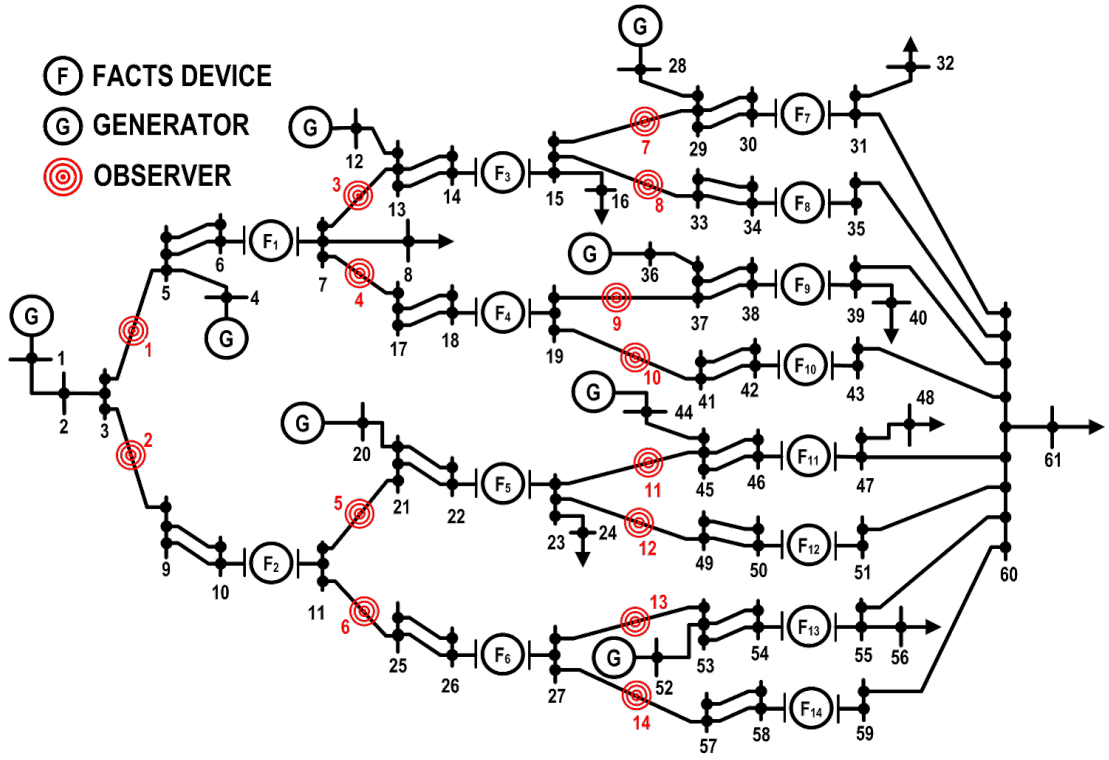


Figure 3.8: An Experimental 61-bus Network with Fourteen Mix Connected IQCs

Table 3.7: The $\mathcal{D}_L$ observation matrix for the Experimental 61-bus Mix Connected Network with Fourteen Mix Connected IQCs

	$\mathcal{D}_L$ Observation													
$\mathcal{D}_H$ Change	$\circledcirc_1$	$\circledcirc_2$	$\circledcirc_3$	$\circledcirc_4$	$\circledcirc_5$	$\circledcirc_6$	$\circledcirc_7$	$\circledcirc_8$	$\circledcirc_9$	$\circledcirc_{10}$	$\circledcirc_{11}$	$\circledcirc_{12}$	$\circledcirc_{13}$	$\circledcirc_{14}$
$F_1 \uparrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_2 \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_3 \uparrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_4 \uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_5 \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$
$F_6 \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$
$F_7 \uparrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_8 \uparrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_9 \uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_{10} \uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_{11} \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_{12} \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$
$F_{13} \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$
$F_{14} \uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$
$F_1 \downarrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_2 \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
$F_3 \downarrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_4 \downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_5 \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$
$F_6 \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\uparrow$
$F_7 \downarrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_8 \downarrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_9 \downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_{10} \downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_{11} \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$
$F_{12} \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$
$F_{13} \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$
$F_{14} \downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\uparrow$

4. CEEME FRAMEWORK

To date, the prominent approach to enforcing security properties and policies is to use a strict safety property [51] whose violation immediately terminates the execution of the target application. Such an approach is not applicable in CPS security analysis since the decision to terminate a cyber process is not sufficient to prevent irremediable consequences in the physical layer; the system has already failed and information flow restrictions have been violated. Also, strict use of safety properties preclude IFPs from being enforced and monitored as IFPs are defined outside the Alpern-Schneider framework [52].

Relaxing the safety property requirement empowers enforcement mechanisms to take appropriate remedial actions at the point of violation [80] : (i) Inject corrective actions [80, 94] or (ii) Backup the application to a previously verified safe state. Either way, the modified execution must maintain the functional integrity of the target system.

4.1. EVENT COMPENSATION

The concept behind *event compensation* is to insert corrective actions at the point where an execution violates a certain security property while still maintaining the functional integrity. In essence, this approach improves the EM security automata [51] by combining it with an emulator [38] and the event insertion capability of the edit automata [80]. Consider the *rearrangement* of actions of trace ϕ_o^* (Equation 5) as a new trace ϕ_r^* in Equation (8) below. The corresponding projection and purged projection outputs are given in Equations (9) and (10) respectively.

$$\phi_r^* = \{\, read_C(), write_B^{\downarrow}(\chi), write_A^{\downarrow}(\chi), read_C(), read_C() \,\} \tag{8}$$

According to Equations (9) and (10), the rearrangement of $\mathcal{D}_H$ actions in trace ϕ_r^* has effectively prevented $\mathcal{D}_L$ subjects from deducing $\mathcal{D}_H$ information. Though there were $\mathcal{D}_H$ changes in the system ($write_B^{\downarrow}(\chi)$ and $write_A^{\downarrow}(\chi)$), these are no longer visible to the $\mathcal{D}_L$. The system behaves as if nothing significant happened from an

outside observer's perspective. In other words, the observable effects of the $\mathcal{D}_H$ action $write_B^{\downarrow}(\chi)$ has been compensated by the $\mathcal{D}_H$ action $write_A^{\downarrow}(\chi)$.

$$\rho(\phi_r^*, \mathcal{D}_L) = \{\ \vec{f_C}, \vec{f_C}, \vec{f_C}\ \} \tag{9}$$

$$\rho(\pi(\phi_r^*, \mathcal{D}_H), \mathcal{D}_L) = \{\ \vec{f_C}, \vec{f_C}, \vec{f_C}\ \} \tag{10}$$

Event compensation formalizes the aforementioned rearrangement of $\mathcal{D}_H$ actions using an enforcement mechanism $\mathcal{E}$. A potential candidate for $\mathcal{E}$ that can accomplish this task needs to possess the following qualities:

- The ability to monitor the executions steps of a target system during runtime and detect security property violations – **Execution Monitoring**
- The ability to identify the action(s) causing the violation – **Safety Property**
- The ability to calculate an appropriate corrective action (sequence) which maintains functional integrity – **Event Compensation**
- The ability to execute the corrective actions in a timely coordinated manner – **Emulation and Enforcement**

Definition 8. *[**Compensation Sequence**] For some identifiable execution violation point $\sigma_j \in \sigma^*$, a compensation sequence $\varsigma \in \Sigma^*$ is defined as a finite sequence of states starting with σ_j which can compensate for σ_j for some property $\mathcal{P}$.*

$$\sigma[\ldots i]\ \sigma[k\ldots] = \sigma[\ldots i]\ \varsigma\ \sigma[k\ldots]$$

$$\text{where } \varsigma \in \Sigma^*, \varsigma = \sigma[j\ldots]$$

Consequently, associated with ς is a finite sequence of compensating actions $\varphi = < \phi_j^c, \phi_{j+1}^c, \ldots, \phi_k^c > \in \Phi^*$ corresponding to each state transition in ς. Thus, a compensated execution takes the following form:

$$\sigma[\ldots i] \xrightarrow{\phi_j^c} \sigma_j \xrightarrow{\phi_{j+1}^c} \sigma_{j+1} \ldots \xrightarrow{\phi_k^c} \sigma[k\ldots] \tag{11}$$

By definition, the first action of a compensating sequence, $\phi_j \equiv \phi_j^c$, is the action that violates the corresponding property $\mathcal{P}$: ϕ_j leads the original execution σ^* from the safe state σ_i to the violation point σ_j. All subsequent actions of the compensating sequence $< \phi_{j+1}^c, \ldots, \phi_k^c > \ \in \varphi$ are actions $\mathcal{E}$ inserts to compensate for the effects of ϕ_j.

Even in projection $\rho(\phi_o^*, \mathcal{D}_L)$ (Equation 6), the value of $\vec{f_C}$ returns to the starting observation at the end of trace. But in contrast to the projection $\rho(\phi_r^*, \mathcal{D}_L)$ (Equation 9), there is a brief time lapse between the first and the second $\mathcal{D}_H$ action when the information flow security is violated. Any $\mathcal{D}_L$ observation made within this period of vulnerability reveals the presence of $\mathcal{D}_H$ activity.

Consider that a particular execution up to state σ_i, i.e., $\sigma[\ldots i]$, is *prefix closed* and **information flow secure**. Prefix closed means that every prefix – every state subsequence starting from the initial state – preserves the underlying security property $\mathcal{P}$. Inherently, the action sequence $\phi[\ldots i]$ is also information flow secure. The state transition $\sigma_i \xrightarrow{\phi_j \equiv \phi_j^c} \sigma_j$ violates $\mathcal{P}$ and takes σ^* to a **information flow vulnerable** state σ_j. During ς the execution stays in this vulnerable state and executes the corresponding φ. σ^* transits back to an information flow secure state σ_k with the last action of $\phi_k^c \in \varphi$. An abstract representation of this notion is presented in Figure 4.1.

Event compensation is only applicable to executions which are *eligible for cleansing*. Cleansing an execution allows it to extend beyond a violation point and prevents it from being discarded. Cleansing, in general, can refer to mechanisms that allow temporary but controlled lapse of property, roll back, or injection of error correction actions. Edit automata [80], which can modify the behavior of an execution during runtime (with suppression and injection), is a good example of execution cleansing.

Definition 9. *[**Execution Cleansing**] Execution cleansing refers to mechanisms which make qualified system executions eligible for extension and prevent them from being discarded.*

Only a certain class of qualified executions can be extended in this manner. Suppose there exists an execution σ with a distinctly identifiable violation point

σ_j, a valid prefix $[\ldots i]$, and a projected postfix $[k\ldots]$. The optimistic assumption is that, in the absence of σ_j, the execution maintains property $\mathcal{P}$. This is similar in concept to the suppression operation introduced in [80]. Formally, this characteristic can be denoted as:

$$\sigma = \sigma[\ldots i]\ \sigma_j\ \sigma[k\ldots]$$

and,

$$\sigma[\ldots i]\ \sigma[k\ldots] \in \mathcal{P} \tag{12}$$

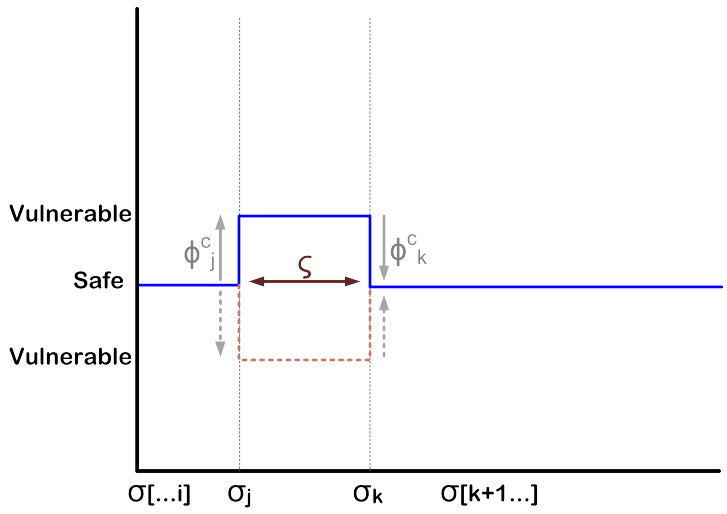


Figure 4.1: A Brief Discrete Vulnerable Period in an Information Flow Secure Environment

Executions similar in form to (12) are eligible for cleansing under $\mathcal{E}$. Mathematically, this allows correction action(s) to be injected immediately after σ_j to extend the execution. However, such an injection needs to compensate for the effects of σ_j in order to maintain the desired property.

In addition, $\mathcal{E}$ must also maintain the functional requirements of the system. By performing event compensation, $\mathcal{E}$ restores the system to an operational and

information flow secure state. Such an optimistic view of the system is also a liveness [49] feature.

4.2. $\mathcal{P}$–COMPENSATE PROPERTY

Definition 10. *[$\mathcal{P}$–**Compensate Property**] A system is compensative with respect to a property $\mathcal{P}$ if and only if, for some execution σ^* with an identifiable violation point σ_j, there exists a compensation sequence ς such that:*

$$\exists \sigma^*,\ \varsigma \in \Sigma^* : \sigma^* \notin \mathcal{P} \implies \neg\hat{\varphi}(\sigma[\ldots j]) \cap \varsigma = \sigma[j\ldots] \cap \sigma[\ldots i]\ \varsigma\ \sigma[k\ldots] \in \mathcal{P} \tag{13}$$

This work quantifies a compensating sequence as a single execution step – a finite sequence of controlled state transitions. By doing so, $\mathcal{E}$ is empowered to inject more than one correction action depending on the requirement and the specific property expected to maintain. Once an eligible execution is identified, a φ is calculated to compensate for σ_j which can lead the overall system back to a safe state with respect to the Information Flow Security Property (IFP) $\mathcal{P}$. The compensative feature of a system also indicates the ability to cleanse executions with respect to some $\mathcal{P}$.

As seen in Figure 4.1, there could be a momentarily lapse in the corresponding security feature. However, the compensated execution as a whole still adheres to the desired property $\mathcal{P}$. The idea is that the system as a whole, not individual operations, need to satisfy the required property [95].

4.3. COMPENSATING COUPLE

In the most basic form, ς consists of a single element and two associated actions, i.e., $|\varphi| = 2$. This is formally defined as a **compensating couple** consisting of a <**action**, **correction**>pair. With this, the system automata in Definition (4) can be extended to a *compensation automata* $\mathcal{M}^c = (\mathcal{D}, \mathcal{Q}, \varphi, \delta, \mathcal{W}, \mathcal{Q}_0)$ as follows. The state space $\mathcal{Q}$ is now divided up into two sets: a set of information flow secure(safe) states $\mathcal{W} \subseteq \mathcal{Q}$ and a set of information flow vulnerable(unsafe) states $\mathcal{V} \subseteq \mathcal{Q}$.

Definition 11 (**Compensation Automata**)**.** *The compensation automata $\mathcal{M}^c$ consists of 6-tuples $(\mathcal{D}, \mathcal{Q}, \varphi, \delta, \mathcal{W}, \mathcal{Q}_0)$ where:*

- $\mathcal{D}$ *is the set of security domains*
- $\mathcal{Q}$ *is a set automaton states*
- $\varphi \subseteq \acute{\Phi}$ *is a subset of input symbols of the form* $< \phi_j^c, \phi_{j+1}^c >: \phi_j^c, \phi_{j+1}^c \in \varphi$
- δ *is the a state transition function* $\delta : \mathcal{Q} \times \varphi \rightarrow 2^{\mathcal{Q}}$ *specified under a predicate* $\hat{\wp}()$
- $\mathcal{W}$ *is a set of final states* $\mathcal{W} \subseteq \mathcal{Q}$
- $\mathcal{Q}_0$ *is a set of initial states for the automaton* $\mathcal{Q}_0 \subseteq \mathcal{Q}$

The net effect of executing a compensating couple is, by definition, null with respect to the particular IFP $\mathcal{P}$ of concern, i.e., $\phi_{j+1}^c - \phi_j^c = \langle\rangle$. The second action of the pair ϕ_{j+1}^c – the correction – moves the state machine back to a IFP secure operationally stable state (or back to the original configuration) while compensating for the IFP violating first action ϕ_j^c. The predicate $\hat{\wp}()$ is used in validating that each compensating transition adheres to the property $\mathcal{P}$. This allows the automata to maintain the $\mathcal{P}$–compensating property during each "compensating" step of the execution. Thus, the following condition holds for executions compensated in this manner.

$$\forall \sigma_j : \neg\hat{\wp}(\sigma[\ldots i] \xrightarrow{\phi_j} \sigma_j[k\ldots]) \cap \exists \varphi =< \phi_j^c, \phi_{j+1}^c >: \phi_j^c \equiv \phi_j \implies \hat{\wp}(\sigma[\ldots i] \xrightarrow{\varphi} \sigma_k) \quad (14)$$

However, the compensating action sequence φ steps through a sequence of unsafe states before reaching a final safe state. The characteristic equation of a compensated execution in Equation (8) steps through the unsafe states, $\sigma_j, \sigma_{j+1}, \ldots \in \mathcal{V}$, before reaching the safe state $\sigma_k \in \mathcal{W}$. As a matter of fact, the states in ς can potentially violate property $\mathcal{P}$, if an $\mathcal{D}_L$ observation is made. Nevertheless, the argument is that ς is finite by definition and φ is executed in a timely and controlled manner to avoid detection by $\mathcal{D}_L$ subjects. Thus, the effect of the inherent security vulnerability is temporary by nature.

The input symbol $\varphi = \ < \phi_j^c, \phi_{j+1}^c >$ to $\mathcal{M}^c$ is the combination of the last state transition command and the next state transition command under the read head of the state machine. Initially, $\phi_j^c = \lambda$ meaning, there is no input command at the

very start. The state transition of ϕ_j^c in $\mathcal{M}^c$ takes the form $q_i \xrightarrow{\phi_j^c} q_j : q_i, q_j \in \mathcal{Q}$. If $q_j \in W$, then ϕ_j^c is the ***correction*** component of an earlier compensating couple. On the other hand, if $q_j \in \mathcal{V}$, then ϕ_j^c is the ***action*** component of a new compensating couple.

The automaton starts in $q_0 \in \mathcal{Q}_0$ and changes to the next state $q_j \in \mathcal{Q}'$ based on the input symbol. The set of all possible next states is given as,

$$\bigcup_{q \in \mathcal{Q}'} \delta(q, \phi_j^c)$$

δ is defined under a predicate $\hat{\wp}()$. If $q_0 \in \mathcal{W}$, the predicate is comparatively simple: the current input command ϕ_j^c under the read head is the < action > component of a new compensating couple. At this point $\mathcal{E}$ starts calculating a φ to compensate ϕ_j^c while $\mathcal{M}^c$ accepts ϕ_j^c free falls to the next state $q_j \in \mathcal{V}$. On the other hand if $q_0 \in V$, half of a compensating couple has already executed. Thus, the current command under the read head σ_{j+1} is the < correction > component. Here, $\hat{\wp}()$ must ensure proper compensation and the set of possible next states is further refined. The predicate can be formally represented as follows:

$$\hat{\wp}(q_j) = \begin{cases} \{q_j \mid q_j \in \mathcal{Q}' \wedge \exists \varphi : \varphi =< \phi_j^c, \ldots >\} & \text{if } q_0 \in \mathcal{W} \\ \{q_j \mid q_j \in \mathcal{Q}' \wedge q_j \in \mathcal{W} \wedge \mathcal{D}_{\phi_j^c} \equiv \mathcal{D}_{\phi_{j+1}^c} \wedge \phi_{j+1}^c - \phi_j^c = \langle\rangle\} & \text{if } q_0 \in \mathcal{V} \end{cases}$$

When $q_0 \in \mathcal{V}$, $\hat{\wp}()$ must ensure that $q_j \in \mathcal{W}$. In addition, the security domain $\mathcal{D}$ in which the next command is going to be executed needs to be equivalent to the security domain of the previous command. This is a key attribute of event compensation and is the only way to prevent subjects in other security domains from deducing information. For example, when $\mathcal{D}_L < \mathcal{D}_H$, $\forall \phi_j^c \in \varphi \; \mathcal{D}_{\phi_j^c} = \mathcal{D}_H$ to prevent information flow from $\mathcal{D}_H$ to $\mathcal{D}_L$.

If $\mathcal{Q}'$ is not empty under $\hat{\wp}(q_j)$, φ can be accepted as a valid state transition command. This can be then passed to the physical system for execution[9]. When ϕ_j^c leaves $\mathcal{M}^c$ in a vulnerable state, $\mathcal{E}$ ensures it reaches a safe state after the next transition and prevents information flow between undesired security domains.

4.4. NOTES ON EVENT COMPENSATION

The compensating couple can be viewed as a ***request-reply*** message passing between two subjects. For example, IQC_i can request IQC_j that it intends to execute a potentially IFP violating $\mathcal{D}_H$ action ϕ_j^c. IQC_j can decide either to acknowledge(reply) or ignore the request. If the request is acknowledged, IQC_i executes $\phi_j^c \in \varphi$ and IQC_j reacts with $\phi_{j+1}^c \in \varphi$. This will ensure that the subjects outside the protection domain of IQC_i and IQC_j are kept unaware of the changes. If the request is ignored, IQC_i can either commit to ϕ_j^c and inject its own anti-action (the reciprocal or the self undo action) as the correction or attempt to reestablish an agreement after a predefined backup time.

In a purely cyber system, a compensating couple represents an ***event undo*** as in a database transaction. This becomes an ***event complement*** in a pure physical system. At most, a third party may be able to observe a sudden pulse as shown in Figure 4.1. But such discrete pulses could have natural causes[10]. By minimizing the time lapse between the action and reaction, two parties can minimize the period of vulnerability and the possibility of $\mathcal{D}_L$ detection.

Event compensation makes sense in actual CPSs since actions (cyber or physical) that affect the equilibrium need to be followed by actions to maintain the overall system stability. An isolated state transition command is more prone to break system equilibrium and potentially violate information flow security. Compensating couples maintain stability and functionality while preventing unauthorized information flow disseminations outside a protection domain.

[9] Similar proposals have been made in [51] and [38]

[10] an air bubble or a clog in the pipeline, lighting striking a power transmission line etc.

Traditional $\mathcal{E}_S$s, such as the security automata in [51], take a conservative approach of enforcement where upon detecting a violation the execution is immediately terminated. This is infeasible for CPSs. This work, on the other hand, considers a more optimistic approach.

From a technical standpoint, event compensation can be used to enforce even some of the stronger IFPs such as Noninterference and Noninference. In doing so, the underlying predicate become too strict such that the proper functional integrity of the system may be compromised. An IFP such as Nondeducibility, which is less stressful on the functional integrity of the system but powerful enough to protect event confidentiality, is a better candidate for IFP enforcement in CPSs.

5. OBFUSCATION THROUGH COMPENSATION

The concept behind event compensation is to inject corrective actions at the point where a particular execution violates a certain security property. With respect to the $\mathcal{D}_H$ event confidentiality of the system, the existence of unique projections in the corresponding $\mathcal{D}_L$ observation matrix is the security violation. Though preventing the physical consequences of a cyber command is not possible within laws of physics, coordinated execution of cyber commands can obfuscate $\mathcal{D}_L$ observations and potentially eliminate unique $\mathcal{D}_L$ projections.

5.1. SELF-OBFUSCATING NETWORKS

A network may present itself to be naturally obfuscated and yield certain $\mathcal{D}_H$ actions Nondeducibility secure. In the network depicted in Figure 3.8, for example, the $\mathcal{D}_H$ actions of the controllers F_1 and F_2 are Nondeducibility secure by default as all four possible commands result in only two $\mathcal{D}_L$ projections (compare $\mathcal{D}_L$ projections for $\{F_1 \uparrow\}$ against $\{F_2 \downarrow\}$ or $\{F_1 \downarrow\}$ against $\{F_2 \uparrow\}$ in Table 3.7).

This particular case of self-obfuscation is explainable using Lemma (3) and Theorem (1). If the subtrees rooted at F_1 and F_2 are ignored, the remaining network resembles a network of two parallel branches, similar in form to Figure 3.3. All observers in each subtree simply inherit the corresponding root observer through replication (Theorem 1). Thus, they do not contribute to deducing actions of F_1 or F_2 through their observations.

In general, every mix connected network including any sub network within one can be analyzed as an rudimentary (parallel or mix) connectivity unit. Even in their simplest form, not every network is Nondeducibility secure since not all networks have the self-obfuscating feature. Even in the ones that are, not every $\mathcal{D}_H$ action is obfuscated. This is where the proposed "event compensation" mechanism makes a contribution.

Once committed, no amount of remedial actions can reverse the physical consequences of a cyber action. Thus, backing up an execution to a previous safe state (as opposed to terminating as in a safety property violation) to maintain a desired

system property is not practically applicable to CPSs. This leaves inserting remedial actions, in this case, compensating actions to enforcing (security) properties at runtime.

5.2. OBFUSCATION THROUGH COMPENSATION

Consider a trace $\breve{\phi}^*$ (associated with an execution $\breve{\sigma}^*$) with an identifiable violation point $\breve{\phi}_i$ with respect to a property $\mathcal{P}$. For this discussion, $\mathcal{P}$ is the property of maintaining $\mathcal{D}_H$ event confidentiality (a $\mathcal{P}$–compensate property [96]) of the CPS and $\breve{\phi}_i$ is a particular $\mathcal{D}_H$ action with a potentially unique $\mathcal{D}_L$ projection. $\{F_7 \uparrow\}$ in Table 3.7 is an example $\breve{\phi}_i$ resulting in $\breve{\sigma}^* \notin \mathcal{P}$. Event compensation inserts additional action(s) immediately after $\breve{\phi}_i$ such that the combined operation of $\breve{\phi}_i$ and these additional actions lead the overall execution to a $\mathcal{P}$–compensatable secure state.

Event compensation can force-obfuscate $\mathcal{D}_H$ actions through compensating action sequences $\varphi = < \phi_j^c, \phi_{j+1}^c, \ldots, \phi_k^c > \in \Phi^*$, in situations where self-obfuscation is not present. A φ can force a particular $\mathcal{D}_L$ projection not to be unique by matching up a potentially $\mathcal{D}_H$ event confidentiality violating action with appropriate correction action(s). One way of doing this is to make sure that the modified projection is similar to (at least) one of the existing projections.

Consider the three traces taken from Table 3.7 in Equation (15) and the corresponding $\mathcal{D}_L$ projections in Equation (16). Note that except for observers $\circledcirc_3, \circledcirc_7$ and $\circledcirc_8$, none of the other observers contribute towards deducibility, as they make the same observation for each of the three $\mathcal{D}_H$ actions.

$$\begin{aligned}
\breve{\phi}_1^* &= \{F_3 \uparrow, \circledcirc_1 \downarrow, \circledcirc_2 \uparrow, \circledcirc_3 \downarrow, \cdots, \circledcirc_7 \downarrow, \circledcirc_8 \downarrow, \cdots, \circledcirc_{14} \uparrow\} \\
\breve{\phi}_2^* &= \{F_7 \uparrow, \circledcirc_1 \downarrow, \circledcirc_2 \uparrow, \circledcirc_3 \downarrow, \cdots, \circledcirc_7 \downarrow, \circledcirc_8 \uparrow, \cdots, \circledcirc_{14} \uparrow\} \\
\breve{\phi}_3^* &= \{F_8 \uparrow, \circledcirc_1 \downarrow, \circledcirc_2 \uparrow, \circledcirc_3 \downarrow, \cdots, \circledcirc_7 \uparrow, \circledcirc_8 \downarrow, \cdots, \circledcirc_{14} \uparrow\}
\end{aligned} \tag{15}$$

$$\begin{aligned}
\rho(\breve{\phi}_1^*, \mathcal{D}_L) &= \{\downarrow, \uparrow, \downarrow, \uparrow, \uparrow, \uparrow, \downarrow, \downarrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow\} \\
\rho(\breve{\phi}_2^*, \mathcal{D}_L) &= \{\downarrow, \uparrow, \downarrow, \uparrow, \uparrow, \uparrow, \downarrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow\} \\
\rho(\breve{\phi}_3^*, \mathcal{D}_L) &= \{\downarrow, \uparrow, \downarrow, \uparrow, \uparrow, \uparrow, \uparrow, \downarrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow\}
\end{aligned} \tag{16}$$

A φ including F_3 and F_7 can prevent $\circledcirc_8$ from making an observation and force $F_3 \uparrow$ and $F_7 \uparrow$ to obfuscate each other. For example, consider $\check{\phi}_i \equiv \phi_j^c = F_7 \uparrow$ and $\phi_{j+1}^c = F_3 \uparrow$. Thus, $\varphi = < F_7 \uparrow, F_3 \uparrow >$. The resulting projection is given in Equation (17). A visual representation of the controller hierarchy abstraction of the 61-bus experimental system (Figure 3.8) is shown in Figure 5.1.

$$\rho(\varphi, \mathcal{D}_L) = \{\downarrow, \uparrow, \downarrow, \uparrow, \uparrow, \uparrow, \uparrow, \leftrightarrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow, \uparrow\} \tag{17}$$

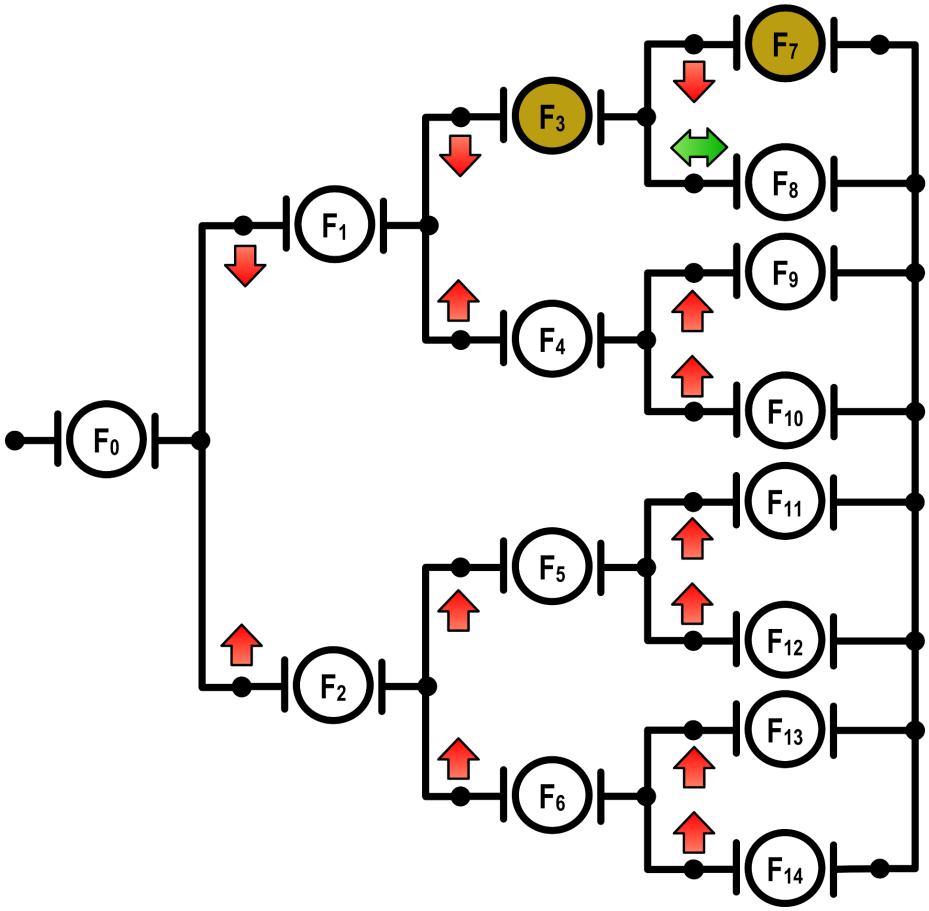


Figure 5.1: A Visual Representation of IQCs 3 and 7 Compensating Observation 8

Note that the symbol $\leftrightarrow$ stands for no change in observation. Effectively, $F_7 \uparrow \in \varphi$ is the action while $F_3 \uparrow \in \varphi$ is the correction in a compensating couple [97].

5.3. THE TIME DOMAIN RESPONSE

The time domain response of the compensating couple $\varphi = < F_7 \uparrow, F_3 \uparrow >$ for observer $\circledcirc_8$ is shown in Figure 5.2. All three subfigures show the same time domain response with different time lapses (a time difference) between the action and the correction. In other words, these are compensating couples with different time lapses[11].

For example, Figure 5.2(a) has a time lapse of two seconds. The wider the time lapse, the longer the period of vulnerability of an information flow security violation [97]. An output sample taken within this period would reveal the existence multiple $\mathcal{D}_H$ actions.

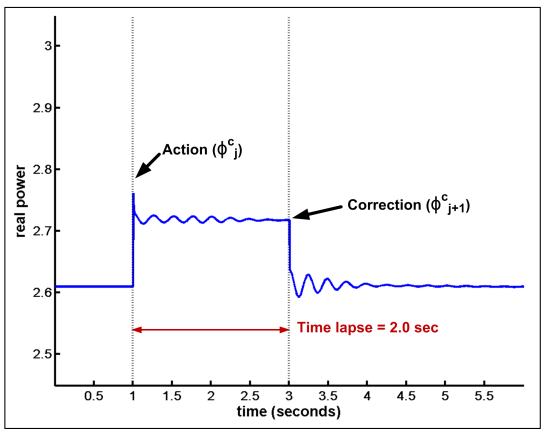
However, it is possible to hide $\mathcal{D}_H$ actions within the *system dynamics* by reducing the time lapse of a compensating couple. The natural system dynamic response is such that the observed outputs respond to any system change by oscillating for a brief period of time before reaching the next steady state. This is evident in Figure 5.2(a) immediately after both action and the correction. The impulse response in Figure 5.2(c) is a result of an instantaneous compensation – a compensating couple with no time lapse. The instantaneous compensation has effectively hidden the compensating couple $\varphi = < F_7 \uparrow, F_3 \uparrow >$ within the system dynamics as the impulse response does not reveal the existence of multiple $\mathcal{D}_H$ actions[12].

Additionally, the response is Noninference secure as a $\mathcal{D}_L$ observer cannot distinguish whether the impulse response is due to some $\mathcal{D}_H$ action sequence or some other natural cause; a sudden surge of power also produce a similar impulse response.

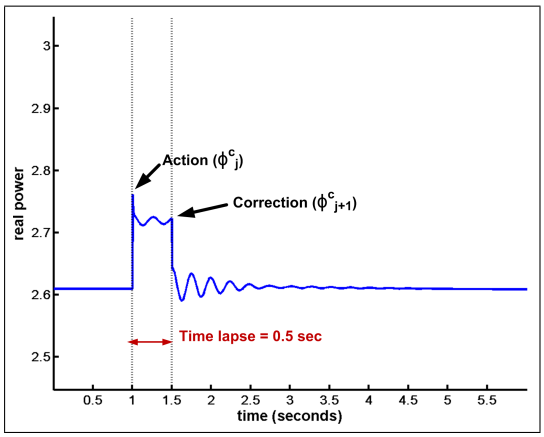
[11] The pulse response was first introduced at a conceptual level in [97]

[12] Conceivably, if the dynamics of the impulse can be predicted, frequency domain analysis by the observer could still separate the two actions. Predicting this response would be extremely difficult, however.

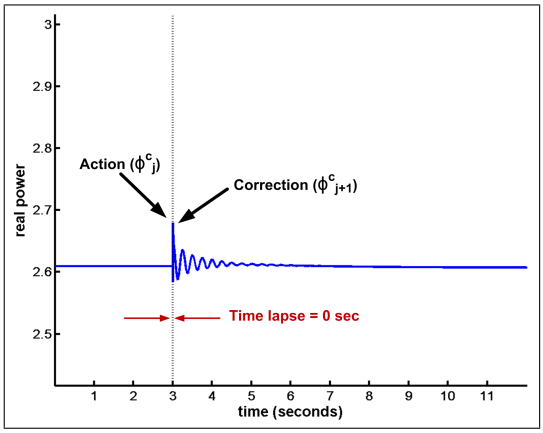
In theory, it is possible to hide any number of $\mathcal{D}_H$ actions within the system dynamics by executing them as compensating actions, as long as such an execution does not violate the operational limits of the system.



(a) Time Lapse 2.0sec



(b) Time Lapse 0.5sec



(c) Time Lapse 0sec

Figure 5.2: Time Domain Response of the Compensating Couple $\varphi = < F_7 \uparrow, F_3 \uparrow >$ on Observer $\circledcirc_8$ with Different Time Lapses

6. CONFIDENTIALITY MODELS FOR CPSs: PART I

Developing a CPS confidentiality model starts with defining a system invariant on flow that accounts for both the commodity flow in the physical layer and the information flow in the cyber layer. In other words, *the control argument needs to be tied to a flow argument* of the system. What is attractive about this approach is that its physical flow and the resulting observations can be controlled with cyber commands with the intention of maintaining a particular system property such as Nondeducibility.

In Chapter 3 the overall system operation was presented as a flow invariant while segregating the actions that change the variables of the invariant into two information flow security domains. Further, the confidentiality violation caused by the inherent cyber-physical interaction was systematically analyzed for regular parallel and mix connected networks by composing basic connectivity blocks into larger and complex networks, resulting in a generalized set of theories that explain the relationship between connectivity and information flow violation. Two important question arise from this analysis:

1. How to identify if a certain $\mathcal{D}_H$ action causes an information violation
2. How to mitigate violations in a finite amount of time

Algorithm (1) provides a basis for an answer to the first question. This algorithm systematically explores a given system and lists systemwide observations pertaining to each $\mathcal{D}_H$ change. Unique $\mathcal{D}_L$ projection entries in the $\mathcal{D}_L$ observation matrix generated using Algorithm (1) are potential information flow violating $\mathcal{D}_H$ system actions. For a runtime enforcement scheme, this information needs to be readily available through preprocessing. $\mathcal{D}_H$ actions corresponding to $\mathcal{D}_L$ projections that are not unique can be safely bypassed since they are, by definition, self-obfuscated (Section 5.1). However, there is a strong likelihood that most entries in a $\mathcal{D}_L$ observation matrix are unique to start with (ex. Table 3.7).

Alternatively, what if there was a way to make sure that each $\mathcal{D}_H$ action is nondeducible in the first place? In other words, obfuscate through compensation.

As demonstrated in Section 5.2, two combined $\mathcal{D}_H$ actions can potentially cancel out and remove observations from appearing on certain lines in the final $\mathcal{D}_L$ projection. Another perspective of this is that a "second action" cancels observations caused by a "first action". This is an instantiation of the ***compensating couple***.

6.1. SYSTEM BEHAVIORS

The act of preventing observations caused by some $\mathcal{D}_H$ action from propagating through the network using the second $\mathcal{D}_H$ action is formally called "**System Behaviors**" here onwards. The device that executes the first action is termed the "*change-origin*" $\mathscr{O}$ and the device that executes the second action is termed the "*behavior-enforcer*" $\mathscr{B}$. Their settings are defined as "*origin setting*" and "*enforcer setting*" respectively. In addition, there is a third element termed "*target*" $\mathscr{T}$ which is the node with an incoming edge whose observation needs to be removed. Depending on the area of confinement and how the behavior is enforced, there are four basic types of system behaviors:

1. Egress Blocking
2. Ingress Blocking
3. Routing
4. Redistribution

Definition 12. *[**System Behaviors for Mix Connected Networks**] For a mix connected network, a system behavior is defined as the act of preventing observations caused by some $\mathcal{D}_H$ change in the system from propagating through the rest of the network through use of a second $\mathcal{D}_H$ change. The device that commits the first change is termed change-origin $\mathscr{O}$ while the second device is termed behavior-enforcer $\mathscr{B}$. Their settings are defined origin setting and enforcer setting respectively. The target $\mathscr{T}$ is the node where the edge whose observation needs to be removed is an incoming edge. The corresponding edge is the $\mathscr{T}$-line.*

Definition 13. *[**Enforcement of System Behaviors**] For a mix connected network, the enforcement of a system behavior is defined as the behavior-enforcer $\mathscr{B}$ using its*

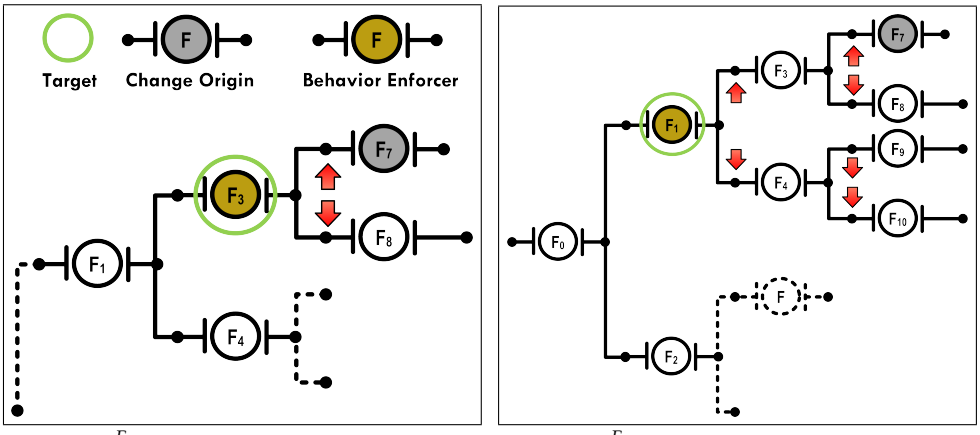
enforcer setting to remove observable changes caused by the origin setting in a target line $\mathscr{T}$-line.

For the rest of this analysis, consider $\text{IQC}_i \equiv F_i$. These terms may be used interchangeably. The four primary system behavior types are explained as follows.

6.1.1. Egress Blocking [B_b^E]. The concept behind Egress Blocking is to confine all observable changes within a subnetwork. Thus, the IQC acting as the $\mathscr{B}$ prevents observable changes of an origin setting from propagating outside the subnetwork rooted at the $\mathscr{T}$ where the $\mathscr{T}$-line is an incoming edge (of $\mathscr{T}$). *The key attribute in Egress Blocking is that change-origin $\mathscr{O}$ is a member of the subtree rooted at target node $\mathscr{T}$.* Thus, for Egress Blocking, the $\mathscr{T}$ and the $\mathscr{B}$ are the same.

Definition 14. ***[Egress Blocking System Behavior** B_b^E **]** A node "b" prevents a change made by one of its descendants from leaving the subnetwork rooted at "b". The changes will be seen by all descendants of "b" but no one else in the system.*

Figure 6.1 shows two examples for Egress Blocking applied on the network shown in Figure 5.1. An Egress Block at IQC_3, (F_3) $B_{F_3}^E$, is shown in Figure 6.1(a). Here, $\mathscr{T}$ is an immediate parent of the $\mathscr{O}$. In Figure 6.1(b), $\mathscr{O}$ and $\mathscr{T}$ are separated by two levels $B_{F_1}^E$. Once the block is applied, the rest of the network beyond the $\mathscr{B}$ does not observe any changes in both these cases.

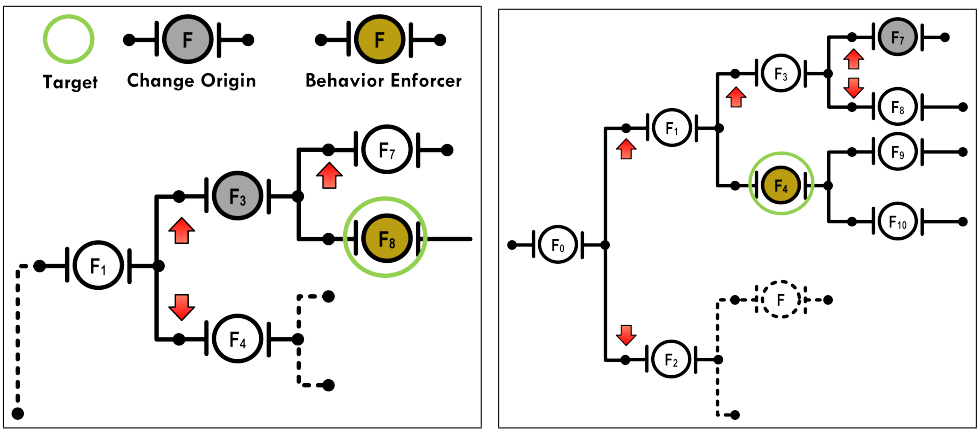


(a) $B_{F_3}^E$: Egress Blocking at IQC_3 (b) $B_{F_1}^E$: Egress Blocking at IQC_1

Figure 6.1: Egress Blocking System Behavior

6.1.2. Ingress Blocking $[B_b^I]$. Ingress Blocking is similar in concept to Egress Blocking except that the $\mathscr{B}$ ensures that no observable changes enter the subnetwork rooted at the $\mathscr{T}$. As was the case with Egress Blocking, both $\mathscr{B}$ and $\mathscr{T}$ are the same node. *The key attribute in Ingress Blocking is that change-origin $\mathscr{O}$ is not a member of the subtree rooted at target node $\mathscr{T}$.* Two examples of Ingress Blocking are given in Figure 6.2. In Figure 6.2(a), the IQC_8 prevents observable changes of its immediate parent IQC_3 from entering its subtree. Thus, $B_{F_8}^I$ is Ingress Block at IQC_8. In Figure 6.2(b), $\mathscr{O}$ and $\mathscr{B}$ have a two levels separation. Thus, for Ingress Blocking, the $\mathscr{T}$ and the $\mathscr{B}$ are the same.

Definition 15. *[**Ingress Blocking System Behavior** B_b^I] A node "b" prevents an external change from entering its subsystem, where an external change is defined as a change made by a non-descendent node. The changes will be hidden from all descendants of "b".*



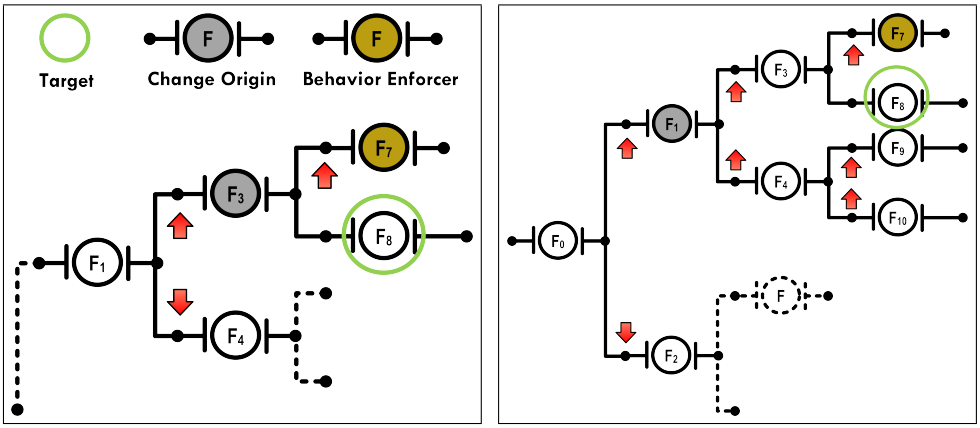
(a) $B_{F_8}^I$: Ingress Blocking at IQC_8 (b) $B_{F_4}^I$: Ingress Blocking at IQC_4

Figure 6.2: Ingress Blocking System Behavior

6.1.3. Routing $[R_{b,t}]$. The Routing system behavior differs from the two blocking behaviors since the objective of $\mathscr{B}$ is to block observations from entering a subnetwork other than its own. Thus, $\mathscr{B}$ and $\mathscr{T}$ are distinct nodes. The $\mathscr{B}$ acts as a proxy Ingress Blocker at the $\mathscr{T}$. However, in certain situations $\mathscr{T}$ can be $\mathscr{O}$

as well. Figure 6.3 shows three examples of the Routing system behavior. *The key attribute in Routing is that change-origin $\mathscr{O}$ is not a member of the subtree rooted at the target node $\mathscr{T}$ and the enforcer $\mathscr{B}$ is in the subtree of one of the $\mathscr{T}$'s siblings subtree.*

Definition 16. *[**Routing System Behavior** $R_{b,t}$] A node "b" prevents an external change from entering a subsystem rooted at "t" by routing the change through itself. The change will be hidden from all children of "t" This behavior mimics the Ingress Blocking behavior at the target.*



(a) R_{F_7,F_8}: Routing IQC_8 through IQC_7 (b) R_{F_7,F_8}: Routing IQC_8 through IQC_7

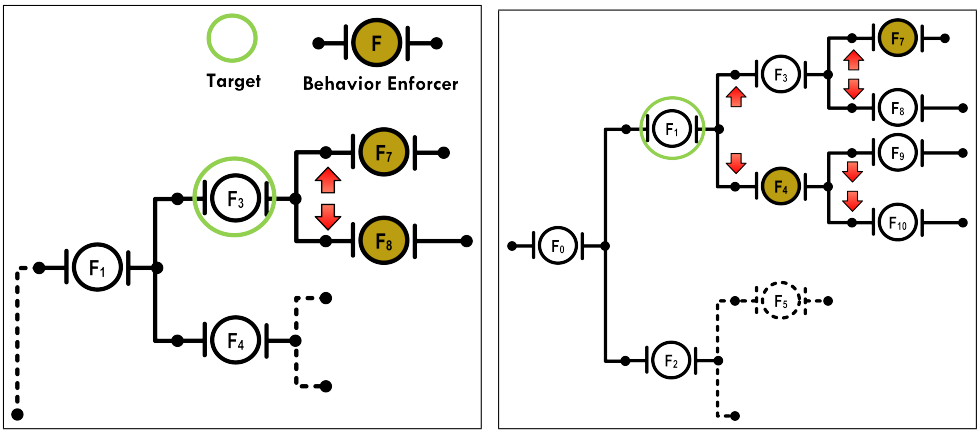
(c) R_{F_7,F_4}: Routing IQC_4 through IQC_7

Figure 6.3: Routing System Behavior

In Figure 6.3(a), $\mathscr{O}$, $\mathscr{B}$, and $\mathscr{T}$ are IQC_3, IQC_7, and IQC_8 respectively. Here, IQC_7 absorbs IQC_3's changes on IQC_8 by routing the change through itself. Definition 6, asserts that the subtree rooted at IQC_8 will not have observable changes when its root IQC_8 has no observable change. This analysis also applies to Figure 6.3(b), except $\mathscr{O}$ is not an immediate parent of either $\mathscr{B}$ nor $\mathscr{T}$. In both Figure 6.3(a) and Figure 6.3(b), $\mathscr{B}$ and $\mathscr{T}$ have a sibling relationship to each other. Figure 6.3(c) shows that the Routing system behavior can be applied even when $\mathscr{B}$ and $\mathscr{T}$ are in one of $\mathscr{O}$'s sibling's subtree.

6.1.4. Redistribution $[D_{b,t}]$. If Routing is a proxy Ingress Blocking, the Redistribution system behavior is the proxy Egress Blocking. Two redistribution nodes can commit opposite changes that cancel-out observations at their first shared ancestor. *The key attribute in Redistribution is that both the behavior-enforcer $\mathscr{B}$ and the change-origin $\mathscr{O}$ are members of the subtree rooted at the target $\mathscr{T}$.*

Consider Figure 6.4 which shows two cases of the Redistribution system behavior. Note in Figure 6.4(a), IQC_7 and IQC_8 make opposite changes that effectively enforce a proxy Egress Block at their shared ancestor IQC_3. The same analysis applies to 6.4(b) where F_7 and IQC_4 cancel observations beyond IQC_1.



(a) D_{F_7,F_3}: Redistribution by IQC_7 for IQC_3

(b) R_{F_7,F_3}: Redistribution by IQC_7 for IQC_3

Figure 6.4: Redistribution System Behavior

Definition 17. *[**Redistribution System Behavior** $D_{b,t}$] A node "b" makes an opposite change in response to an origin change in one of its sibling's subtree such that it nullifies observations at the first shared ancestor "t". This behavior mimics the Egress Blocking behavior at the shared ancestor.*

Note that in Figure 6.4(a), IQC_8 becomes $\mathscr{B}$ when the $\mathscr{O}$ is IQC_7 and vice versa. Similarly, for Figure 6.4(b), IQC_4 becomes the $\mathscr{B}$ when IQC_7 is the $\mathscr{O}$ vice versa.

6.2. SYSTEM BEHAVIORS AND NONDEDUCIBILITY

System behaviors provide an elegant method to obfuscate observations. This approach removes the adversary's ability to deduce two single action settings by reducing the number of observations present in the final $\mathcal{D}_L$ projection. The two actions in a system behavior are interchangeable, meaning that the origin can be replaced by $\mathscr{B}$ to produce the same set of $\mathcal{D}_L$ observations. Thus, looking at the residual observations, an adversary cannot distinguish either origin or the enforcer as the actual system change. This effectively maintains the event confidentiality of the two $\mathcal{D}_H$ actions.

Theorem 4. *[**Nondeducible Settings in Egress Blocking and Redistribution**] For each change-enforcer combination in Egress Blocking system behavior in a mix-connected network, there exists a nondeducible change-enforcer combination in Redistribution system behavior and vice versa.*

Theorem 5. *[**Nondeducible Settings in Ingress Blocking and Routing**] For each change-enforcer combination in Ingress Blocking system behavior in a mix-connected network, there exists a nondeducible change-enforcer combination in Routing system behavior and vice versa.*

Proof. These two theorems can be proven using a flow invariant as follows. Consider two sets of real value variables $|P| = n, |Q| = m : n \geq 1, m > 1$ and their relationship as shown in Equation 18. For the purpose of this analysis, consider the

sets P and Q as incoming and outgoing flows and their relationship preserves the *flow conservation rule* at a particular vertex with no storage.

$$p_1 + p_2 + \ldots + p_n = q_1 + q_2 + \ldots + q_m \tag{18}$$

Figure 3.1 is one such example with $n = 1$ and $m = 2$. Assume a $\Delta q_i : q_i \in Q$ change in one of the right hand side variables in Equation (18). In response to this change, one or more variables in the set $P \cup Q - q_i$ have to change[13]. Suppose the strategy is not to let variables in the set P change their values. There are two methods of achieving this goal. The first one is by explicitly fixing the values in the set P to their starting values. This forces variables in the set $Q - q_i$ to be reassigned. The second method is to reassign values to the variables in $Q - q_i$ such that the set P does not change. Since both these methods achieve the same final outcome, they are equivalent by definition. With respect to system behaviors, Egress Blocking is the method of explicitly fixing the values for set P while Redistribution is reassigning values to set $Q - q_i$. The change-origin is represented in Δq_i. The equivalence relationship between the two behaviors results in the existence of duality in solutions. Thus, each change-enforcer combination can be achieved by both system behaviors resulting in a nondeducible secure setting.

Similarly, assume a $\Delta p_j : p_j \in P$ change in one of the left hand side variables in Equation (18). In this case, consider the objective is to maintain the initial values for a subset of variables $\hat{Q} \subseteq Q$. Again, there are two methods of achieving this goal. The first method is to explicitly fix the values in the set $\hat{Q}$. In other words, this is the Ingress Blocking system behavior. The second method is by reassigning values to the variables in set $\{Q \cup P - (\hat{Q} \cup p_j)\}$ such that the set $\hat{Q}$ does not change. This is achieved by the Routing system behavior. Using the same line of reasoning as above, it can be shown that there is an equivalence relationship between the Ingress Blocking system behavior and the Routing system behavior. As a consequence, each change-enforcer combination that can be achieved in Ingress Blocking can be replicated using Routing and vice versa. □

[13] If more than one variable is involved, the aggregate change must equal Δq_i

As an example, consider the Egress Block $B^E_{F_7}$ in Figure 6.1(a) and the Redistribution D_{F_7,F_8} in Figure 6.4(a). As shown, the change-enforcer combinations for these two system behaviors are $\varphi_1 = < F_7 \uparrow, F_3 \downarrow >$ and $\varphi_2 = < F_7 \uparrow, F_8 \downarrow >$. For the purpose of analysis, assume that the traces for each of these two system behaviors are $\phi^*_{B^E}$ and ϕ^*_D. Thus,

$$\phi^*_{B^E} = \{F_7 \uparrow, F_3 \downarrow, \circledcirc_7 \uparrow, \circledcirc_8 \downarrow\}$$
$$\phi^*_D = \{F_7 \uparrow, F_8 \downarrow, \circledcirc_7 \uparrow, \circledcirc_8 \downarrow\}$$

A direct comparison between these two figures shows how the same origin-enforcer combination can effectively control which subnetwork will see observations. Moreover, the $\mathcal{D}_L$ projection for both $\phi^*_{B^E}$ and ϕ^*_D are equivalent, i.e.,

$$\rho(\phi^*_{B^E}, \mathcal{D}_L) = \rho(\phi^*_D, \mathcal{D}_L) = \{\circledcirc_7 \uparrow, \circledcirc_8 \downarrow\}$$

In other words the combinations $\varphi_1 = < F_7 \uparrow, F_3 \downarrow >$ and $\varphi_2 = < F_7 \uparrow, F_8 \downarrow >$ are nondeducible. These are, by definition, compensating couples which by Definition 10 preserve the Nondeducibility-Compensate property. A similar analysis applies for the case studies shown in Figures 6.1(b) an 6.4(b).

Consider the Ingress Block $B^I_{F_8}$ in Figure 6.2(a) and the Routing R_{F_7,F_8} in Figure 6.3(a). The origin-enforcer combinations for $B^I_{F_8}$ and R_{F_7,F_8} are $\varphi_3 = < F_3 \uparrow, F_8 \downarrow >$ and $\varphi_4 = < F_3 \uparrow, F_7 \uparrow >$ respectively.

The corresponding traces $\phi^*_{B^I}$ and ϕ^*_R can be listed as follows:

$$\phi^*_{B^I} = \{F_3 \uparrow, F_8 \downarrow, \circledcirc_1 \uparrow, \circledcirc_2 \downarrow, \circledcirc_3 \uparrow, \circledcirc_4 \downarrow, \circledcirc_5 \downarrow, \circledcirc_6 \downarrow, \circledcirc_7 \uparrow, \circledcirc_8 \leftrightarrow, \circledcirc_9 \downarrow, \circledcirc_{10} \downarrow, \ldots\}$$
$$\phi^*_R = \{F_3 \uparrow, F_7 \uparrow, \circledcirc_1 \uparrow, \circledcirc_2 \downarrow, \circledcirc_3 \uparrow, \circledcirc_4 \downarrow, \circledcirc_5 \downarrow, \circledcirc_6 \downarrow, \circledcirc_7 \uparrow, \circledcirc_8 \leftrightarrow, \circledcirc_9 \downarrow, \circledcirc_{10} \downarrow, \ldots\}$$

Since, both these system behaviors prevent observations from entering the subtree rooted at F_8, they both produce the same $\mathcal{D}_L$ projection. Thus, φ_3 and φ_4 are nondeducible.

$$\rho(\phi^*_{B^I}, \mathcal{D}_L) = \rho(\phi^*_R, \mathcal{D}_L) = \{\circledcirc_1 \uparrow, \circledcirc_2 \downarrow, \circledcirc_3 \uparrow, \circledcirc_4 \downarrow, \circledcirc_5 \downarrow, \circledcirc_6 \downarrow, \circledcirc_7 \uparrow, \circledcirc_8 \leftrightarrow, \circledcirc_9 \downarrow, \circledcirc_{10} \downarrow, \ldots\}$$

From an EM enforcement perspective, evaluating the uniqueness of a $\mathcal{D}_L$ projection resulting from a system behavior at runtime is not an easy task. If the residual $\mathcal{D}_L$ projection of a system behavior is unique, the purpose of using them in the first place is lost. The projection in question needs to be compared against the set of all possible simultaneous $\mathcal{D}_H$ changes of the system to see if it matches the existing projections. For a network of n controllable devices each with k different changes, there are $\binom{n}{2} * 2^k$ total $\mathcal{D}_H$ change combinations, increasing preprocessing requirements.

Corollary 1. *[**Nondeducible System Behaviors: Egress Blocking and Redistribution**] Egress Blocking and Redistribution system behaviors are nondeducible equivalent.*

Corollary 2. *[**Nondeducible System Behaviors: Ingress Blocking and Routing**] Ingress Blocking and Routing system behaviors are nondeducible equivalent.*

Proof. Corollaries 1 and 2 are derived from Theorems 4 and 5 respectively. From Theorem 4, there exists a nondeducible counter part in Redistribution system behavior for each action-enforcer combination in Egress Blocking system behavior. This means that from a $\mathcal{D}_L$ observation point of view, an Egress Blocking system behavior enforcement cannot be distinguished from a Redistribution system behavior enforcement and vice versa. Thus, Egress Blocking and Redistribution system behaviors are nondeducible equivalent. Ingress Blocking and Routing system behaviors are also nondeducible equivalent. This can be proven using a similar argument as above. □

As a consequence of Corollaries (1) and (2), system behaviors are not required to be checked for uniqueness in $\mathcal{D}_L$ projections. For each residual observation of a system behavior, there is an equivalent dual in another system behavior. This duality in solutions preserves the event confidentiality of origin-enforcer combination.

7. CONFIDENTIALITY MODELS FOR CPSs: PART II

As an alternative to the system behavior approach, there is a second method to produce nondeducible $\mathcal{D}_H$ settings in regular mix connected networks. This method replaces the original $\mathcal{D}_H$ action, the setting of $\mathcal{O}$ in other words, with other actions that meet the same functional requirements and produce the same $\mathcal{D}_L$ projection. In flow networks, the underlying objective of any $\mathcal{D}_H$ action is to enforce either a positive (increase flow) or a negative (decrease flow) flow constraint on a certain set of edges. The same constraint can be achieved by enforcing flow values on other edges in such way that the intended amount of flow goes through the initially selected set of edges.

7.1. REPLACEMENT SOLUTIONS

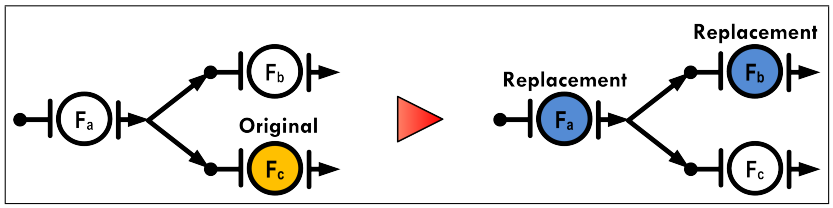
Definition 18. *[**Upstream Replacement ($\mathcal{R}_{\blacktriangle}$)**] The act of replacing a $\mathcal{D}_H$ change with its immediate parent and, if they exist, all its siblings is termed Upstream Replacement ($\mathcal{R}_{\blacktriangle}$).*

Definition 19. *[**Downstream Replacement ($\mathcal{R}_{\blacktriangledown}$)**] The act of replacing a $\mathcal{D}_H$ change with its immediate children is termed Downstream Replacement ($\mathcal{R}_{\blacktriangledown}$).*

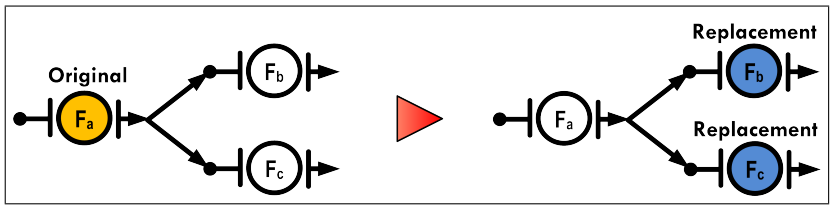
Figure 7.1 shows two basic schemes to replace a single $\mathcal{D}_H$ action for a mix connected network. The two schemes are termed $\mathcal{R}_{\blacktriangle}$ and $\mathcal{R}_{\blacktriangledown}$ respectively based on the direction of the move in comparison to the flow direction. For example, the original $\mathcal{D}_H$ change was moved to the immediate parent and sibling couple in Figure 7.1(a) to form an $\mathcal{R}_{\blacktriangle}$. In Figure 7.1(b), the original change location was moved to the immediate children to form a $\mathcal{R}_{\blacktriangledown}$. Both these schemes adhere to the flow conservation rule.

Consider the line flows for IQC_a, IQC_b, and IQC_c in Figure 7.1 as $\vec{f_a}$, $\vec{f_b}$, and $\vec{f_c}$ respectively. Based on the flow direction, the relationship between the flow values at stable state can be stated as, $\vec{f_a} = \vec{f_b} + \vec{f_c}$. For the $\mathcal{R}_{\blacktriangle}$ scheme in Figure7.1(a), F_c enforcing $\vec{f_c}$ is equivalent to F_a and F_b enforcing flow values $\vec{f_a}$ and $\vec{f_b}$. The same argument applies for the $\mathcal{R}_{\blacktriangledown}$ scheme in Figure 7.1(b). Thus, any flow value of

the original line can be easily maintained using its replacement counterparts. In theory, this would yield the same $\mathcal{D}_L$ observation as the original change.



(a) Upstream Replacement ($\mathcal{R}_{\blacktriangle}$)



(b) Downstream Replacement ($\mathcal{R}_{\blacktriangledown}$)

Figure 7.1: Single High-Level Domain ($\mathcal{D}_H$) Action Replacement Schemes for Mix Connected Networks

In the system behavior approach, the original $\mathcal{D}_H$ action remains in the final nondeducible $\mathcal{D}_H$ action combination. In contrary, the replacement of solutions will produce alternative $\mathcal{D}_H$ action combinations that do not include the original action. However, each replacement solution will produce the same $\mathcal{D}_L$ projection as the original action.

Theorem 6. *[**Nondeducibility of Replacement Solutions**] Any replacement solution of a $\mathcal{D}_H$ change is nondeducible.*

Proof. The proof of this theorem also follows a flow invariant argument. Consider the earlier introduced Equation 18. Assume a $\Delta q_i : q_i \in Q$ change in one of the right hand side variables. Thus, $\hat{q_i} - q_i = \Delta q_i$ (action). Due to flow conservation, this change will excite all other variables in the equation to new values $\hat{P}$ and $(\hat{Q} - q_i)$ (consequence). Now consider the converse of this assignment where $\hat{P}$ and $(\hat{Q} - q_i)$ force Δq_i in q_i. If $\hat{q_i} \rightarrow \hat{P} \cup (\hat{Q} - q_i)$ is considered the original $\mathcal{D}_H$ change, the converse $\hat{P} \cup (\hat{Q} - q_i) \rightarrow \hat{q_i}$ is the $\mathcal{R}_{\blacktriangle}$. The flow values are equivalent in both

cases and thus, will produce the same $\mathcal{D}_L$ projection. Any subsequent recursive application of $\mathcal{R}_{\blacktriangle}$ will also produce the same $\mathcal{D}_L$ projection. Thus, $\mathcal{R}_{\blacktriangle}$ solutions are nondeducible.

Assume $\forall p_i \in P : p_i \to \hat{p_i}$ for all incoming flows (R.H.S. variables) where $\hat{p_i} - p_i = \Delta p_i$. Such a change will consequently result in $\forall q_i \in Q : q_i \to \hat{q_i}$. If this transition $\hat{P} \to \hat{Q}$ is considered the original $\mathcal{D}_H$ change, it's converse $\hat{Q} \to \hat{P}$ becomes the $\mathcal{R}_{\blacktriangledown}$. Both cases maintain the same flow values and the same $\mathcal{D}_L$ projections. Similar to $\mathcal{R}_{\blacktriangle}$ analysis above, any recursive application will also produce the same $\mathcal{D}_L$ projection. Thus, $\mathcal{R}_{\blacktriangledown}$ are also nondeducible.

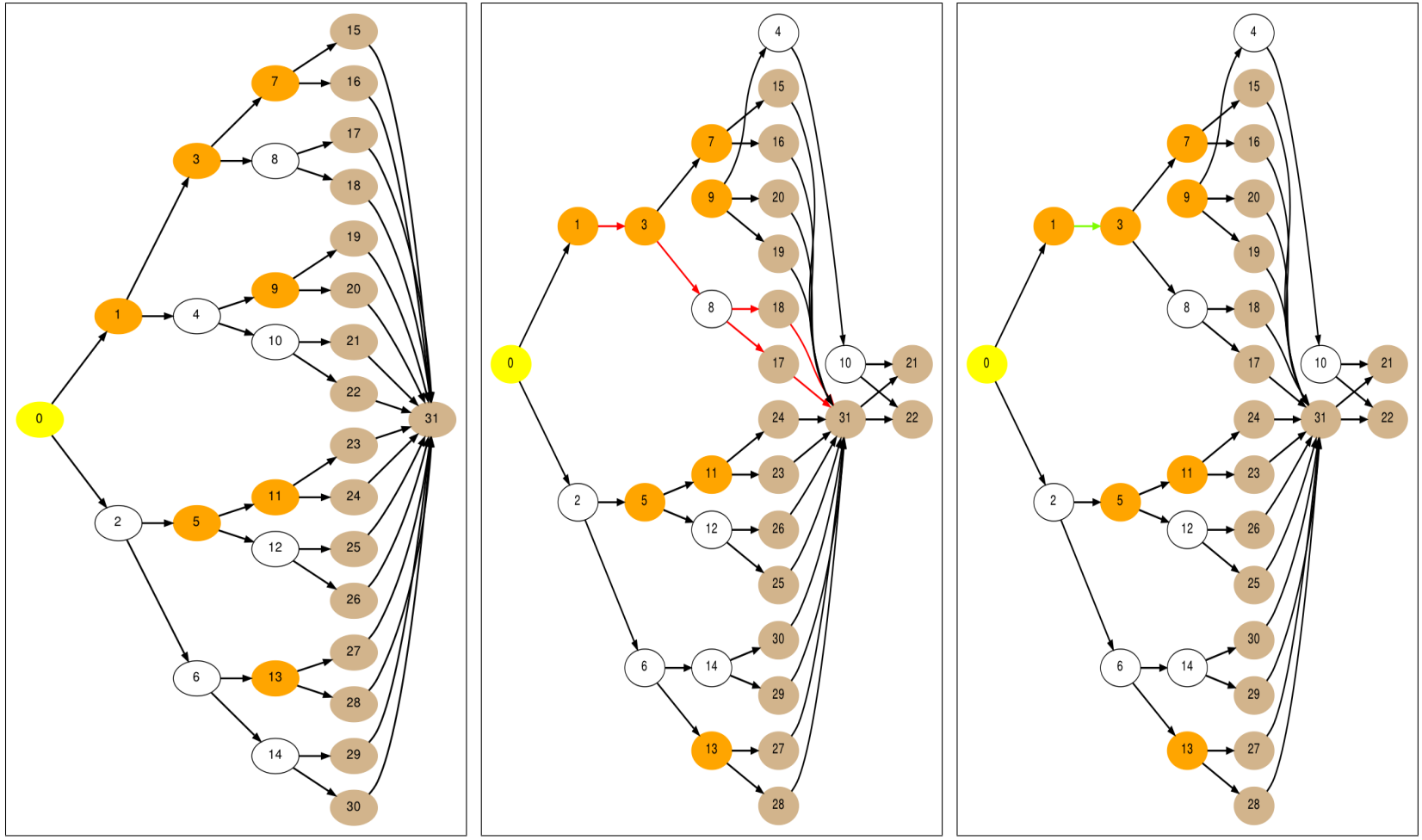
Additionally, a $\mathcal{R}_{\blacktriangledown}$ of a $\mathcal{R}_{\blacktriangle}$ and its converse are also nondeducible since any combination of replacements will preserve the flow conversation property, flow values, and $\mathcal{D}_L$ projections. Thus, any replacement solution of a $\mathcal{D}_H$ action is nondeducible. □

Consider the three states of the tree structured 31-bus test network shown in Figure 7.2. Figure 7.2(a) is the initial state of the network before a line outage. In Figure 7.2(b), the line between nodes 1-4 is removed as a line contingency. As a result, the edges in the path $1 \to 3 \to 8 \to 17, 18 \to 31$ are overloaded. This is illustrated with red colored edges in Figure 7.2(b).

To alleviate this outage, an IQC can be placed between nodes 1 and 3 to restrict flow. This is shown with a green edge in Figure 7.2(c). Here, the yellow colored node numbered 0 is the swing bus. The orange colored nodes (nodes $1, 2, 5, 7, 9, 11,$ and 13) are buses with attached generators and the light brown colored nodes (nodes $15-31$) are buses with attached loads. The function of the swing bus is to absorb or release any additional power in the system[14].

From an external observer's perspective, the state transitions from initial state to the outage state and outage state to the FACTS state both produce observable changes. In this analysis, the focus is on the latter transition since the main cause of this set of observations is an action taken by an IQC. This is in fact the specific $\mathcal{D}_L$ projection that can potentially be used in deducibility.

[14]The rearrangement of certain nodes in subsequent figures is an unintentional side effect of a limitation in the open source graphing tool "graphviz" `http://www.graphviz.org/` used for plotting.



(a) Initial State Before Line 1-4 Outage (b) Outage State During Line 1-4 Outage (c) IQC State After Line 1-4 Outage Correction

Figure 7.2: The Experimental 31-bus Tree Structured Test Network with 3 States

Figure 7.3 shows the $\mathcal{D}_L$ observable changes of the FACTS state in comparison to the outage state. Here, the edge label "U" denotes increase in flow (equivalent to the $\uparrow$ notation used in earlier chapters) while the "D" denotes decrease ($\downarrow$ notation). Abiding to the Definition (6), all edges under the subtree rooted at node 1 have decreased flow. The subtree rooted at node 2, which is the sibling of node 1, inherit through flip (see Definition 7).

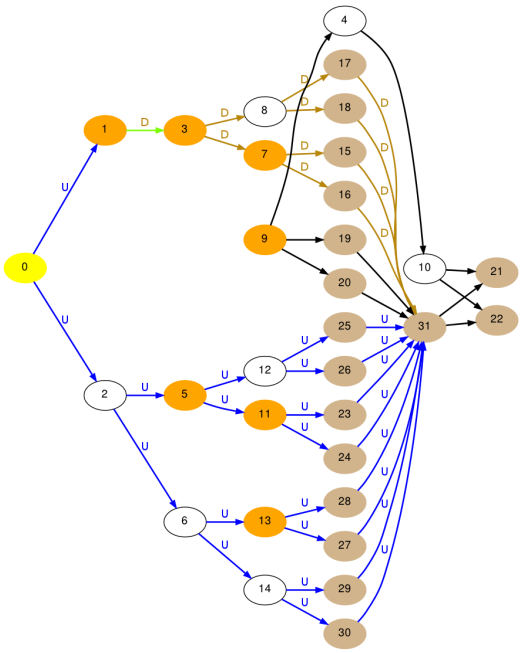
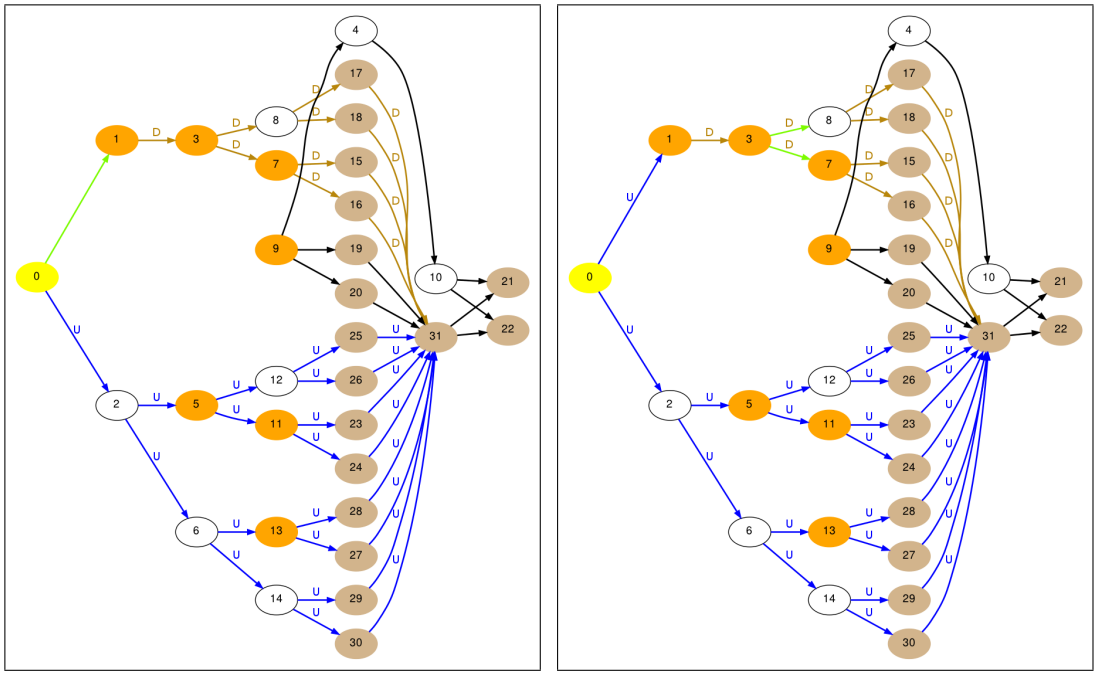


Figure 7.3: The IQC State of the Experimental 31-bus Tree Network with $\mathcal{D}_L$ Observations Compared to the Outage State

Due to the behavior of the swing bus, the link between nodes $0-1$ does not abide by Definition (6). This is considered a special exception to the observation inheritance rules (Definitions 6 and 7) for power distribution networks. Between the outage state and the FACTS state, there are no observable changes in the subnetwork rooted at node 9 since this subnetwork is outside the control of the IQC at edge $1-3$. The outage at edge $1-4$ caused the subtree rooted at node 4 to be isolated from the rest of the network, which resulted in the lone generator at node 9 to become a pseudo source node for this subnetwork.

Figure 7.4(a) is the $\mathcal{R}_{\blacktriangle}$ of the IQC in Figure 7.3. Similarly, Figure 7.4(b) is the $\mathcal{R}_{\blacktriangledown}$ of the same IQC placement. Note that in all three cases, the set of $\mathcal{D}_L$ observations of the network stays the same. Just by looking at the set of $\mathcal{D}_L$ observations, an adversary is not able to deduce which set of IQC placements produced the $\mathcal{D}_L$ projection. Thus, from a Nondeducibility perspective, the two replacement solutions preserve the event confidentiality of the original IQC placement and setting.

The two replacement schemes can be applied recursively to produce additional replacement solutions. For the IQC placement in Figure 7.2(c), there are 31 replacement solutions that yield the same $\mathcal{D}_L$ observation. Most of these are theoretical solutions; the number of IQC devices involved in a particular replacement solution increases as the distance from the original location increases.



(a) $\mathcal{R}_{\blacktriangle}$ for Figure 7.2(c)

(b) $\mathcal{R}_{\blacktriangledown}$ for Figure 7.2(c)

Figure 7.4: The Upstream Replacement ($\mathcal{R}_{\blacktriangle}$) and Downstream Replacement ($\mathcal{R}_{\blacktriangledown}$) Replacement Schemes for Figure 7.2(c)

For practical purposes, it is not possible to place such a high number of IQC devices. A more feasible option would be to bound the solution space to a certain number of hops from the original location and only consider solutions that comprise of an agreeable number of IQCs.

7.2. CALCULATING REPLACEMENT SOLUTIONS

The total number of replacement solutions for a mix-connected regular network with a given IQC placement depends on the original placement and how it partitions the given network. Based on the direction of replacement, upstream or downstream, the network gets partitioned into two subnetworks – the $\mathcal{R}_{\blacktriangledown}$–tree and the $\mathcal{R}_{\blacktriangle}$–tree. The replacement solutions from these two subtrees are mutually exclusive, meaning that there does not exist a replacement solution that include devices from both subnetworks at the same time. Thus, from the point of view of generating replacement solutions, the $\mathcal{R}_{\blacktriangle}$–tree and the $\mathcal{R}_{\blacktriangledown}$–tree are mutually exclusive.

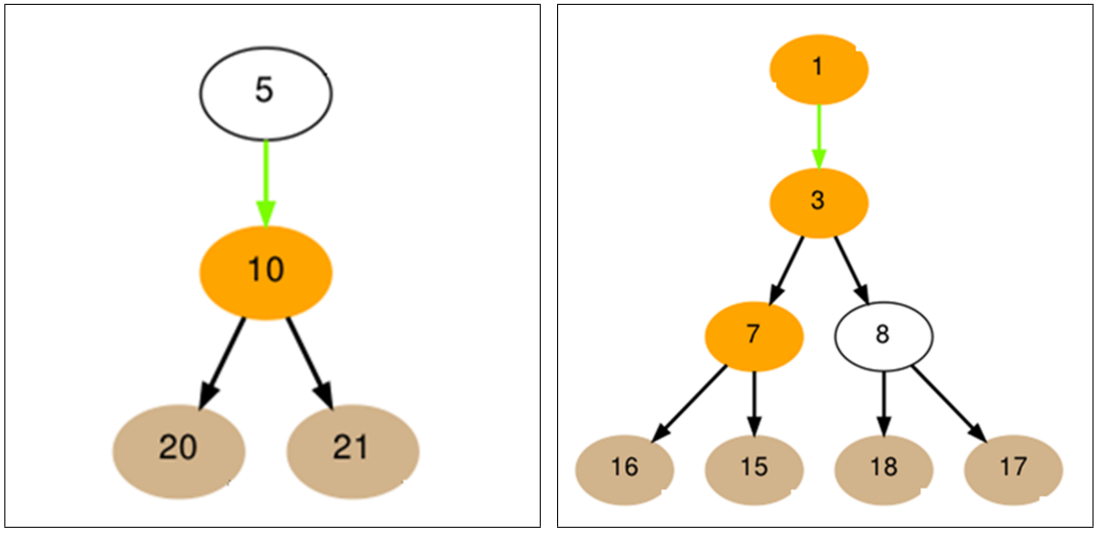
In its most basic form, the formula to calculate the total number of all possible replacement solutions for regular mix-connected network with a given original placement (inclusive the original placement) takes the form:

$$total_{sol} = downtream_{sol} + upstream_{sol} + original_{sol} \tag{19}$$

Calculating $downtream_{sol}$ is relatively straightforward. For $\mathcal{R}_{\blacktriangledown}$s, the total number of replacement solutions is the sum of the total number of permutations in each subtree rooted under the original placement multiplied by each other and the original placement solution. For example, consider the basic $\mathcal{R}_{\blacktriangledown}$ tree with a single-level in Figure 7.5(a). Here, each subtree under node 10 has a single permutation. The total number of replacement solutions equal $1 * 1 = 1$. Added to this is the original placement between nodes $5 - 10$, which brings up the total number of nondeducible solutions to 2.

This process can be recursively applied to trees with higher levels. For example, in Figure 7.5(b), the two subtrees under node 3 form a single-level subtree by themselves with 2 replacement solutions each. The immediate result of this is

that the total number of replacement solutions under the subtree rooted at node 3 is $2 * 2 = 4$. Thus, the total number of replacement solutions for a two-level tree is $replacements + original$ which is $4 + 1 = 5$.



(a) Basic $\mathcal{R}_{\blacktriangledown}$ with a Single-Level Tree (b) Basic $\mathcal{R}_{\blacktriangledown}$ with a Two-Level Tree

Figure 7.5: Basic Downstream Replacement ($\mathcal{R}_{\blacktriangledown}$) Schemes

The $\mathcal{R}_{\blacktriangledown}$ solution calculation method described above can be use to calculate the $\mathcal{R}_{\blacktriangle}$ solution count as follows. Consider each subtree rooted under each direct ancestor of the original placement, i.e., parent, grandparent, great-grandparent, etc., as a $\mathcal{R}_{\blacktriangledown}$ subtree under the $\mathcal{R}_{\blacktriangle}$–tree. The subtree rooted under the parent node is considered the *first* $\mathcal{R}_{\blacktriangledown}$*–subtree of the* $\mathcal{R}_{\blacktriangle}$*–tree*. The subtree rooted under the grandparent is the *second* $\mathcal{R}_{\blacktriangledown}$*–subtree of the* $\mathcal{R}_{\blacktriangle}$*–tree*. With this numbering scheme in place, the total number of replacement solutions at the i^{th} $\mathcal{R}_{\blacktriangledown}$–subtree of the $\mathcal{R}_{\blacktriangle}$–tree and beyond can be expressed as follows:

$$upstream_{sol_i} = sol_i + sol_i * upstream_{sol_{i+1}}$$

Here, sol_i denotes the number of replacement solutions in the i^{th} $\mathcal{R}_{\blacktriangledown}$–subtree of the $\mathcal{R}_{\blacktriangle}$–tree. The general form of equation to calculate the number of replacement solutions in the $\mathcal{R}_{\blacktriangle}$ tree can be expressed as follows:

$$upstream_{sol} = sol_1 + sol_1 * (sol_2 + sol_2 * (sol_3 + sol_3 * (\ldots))\ldots))) \tag{20}$$

With these formulae in place, consider the network in Figure 7.2(c) with the given IQC placement between nodes $1-3$. The $\mathcal{R}_{\blacktriangledown}$–subtree rooted at node 3 is a two level tree equivalent to Figure 7.5(b). Thus, $downstream_{sol} = 4$. The first $\mathcal{R}_{\blacktriangledown}$–subtree of the $\mathcal{R}_{\blacktriangle}$–tree consists of the series parent edge between nodes $0-1$. The second $\mathcal{R}_{\blacktriangledown}$–subtree of the $\mathcal{R}_{\blacktriangle}$–tree is rooted at node 0. Consider the two subtree rooted at node 2. Technically, this is a tree-level $\mathcal{R}_{\blacktriangledown}$ tree. These are two two-level subtrees each carrying 5 replacement solutions. Thus, the total number of replacement solutions under the subtree rooted at node 2 equals $5 * 5 = 25$. This in series with the other possible replacement between nodes 0-2 adds up to being 26 different replacement solutions for the second $\mathcal{R}_{\blacktriangledown}$–subtree of the $\mathcal{R}_{\blacktriangle}$–tree w.r.t. the original placement. Thus:

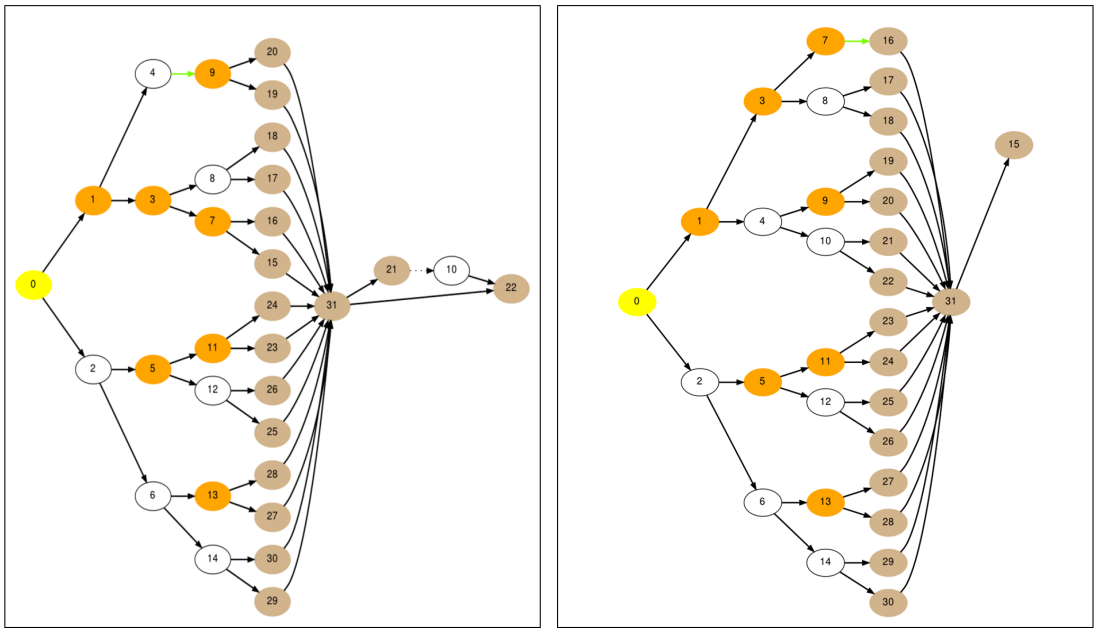
$$upstream_{sol} = (sol_1 + sol_1 * (upstream_{sol_2})) = (1 + 1 * (26) = 27$$

For the regular mix-connected network with the given placement between nodes $1-3$ as in Figure 7.2(c), the total number of all possible replacement solutions equals:

$$\begin{aligned} total_{sol} &= downtream_{sol} + upstream_{sol} + original_{sol} \\ &= 4 + 27 + 1 \\ &= 32 \end{aligned}$$

Using the same formulae, it is possible to calculate the total number of replacement solutions for a given placement at any level. As examples, consider two additional original placements for the same 31-bus tree structured test networks shown in Figure 7.6.

In Figure 7.6(a), the original IQC placement is between the nodes 4−9, which is at level 2 of the tree rooted at node 0. According to the Equation (19), $total_{sol} = 1 + (1 + 1 * (5 + 5 * (26))) + 1 = 138$. In Figure 7.6(b), the original IQC placement is between the nodes 7 − 16, which is at level 3 of the tree rooted at node 0. For this case, $total_{sol} = 0 + (1 + 1 * (2 + 2 * (5 + 5 * (26)))) + 1 = 274$. These values were verified using independent simulation results. More specifically, Algorithm 3 was used to calculate the total number of solutions for all possible initial IQC placements in the 31-bus tree structured test network shown in 7.2(a) and these calculated values matched with values derived using Equation 19.



(a) Given IQC placement at level 2 (b) Given IQC placement at level 3

Figure 7.6: 31-bus Test Network with Given IQC placement at level 2 and 3

7.3. NOTES ON CALCULATING REPLACEMENT SOLUTIONS

In general, Equation 19 can be used to calculate the total number of replacement solutions for any given regular mix-connected network with a given initial placement. The number of subtrees in a branch and the number of replacement

solutions in each branch will affect the total number of solutions. Care must be taken to consider all possible permutations and combinations when dealing with trees that are not balanced or complete. Thus, a manual, by hand calculation may be required.

8. THE CEEME ARCHITECTURE

Building on the theoretical foundations introduced in the previous chapters, this chapter introduces the architecture of CEEME as a runtime enforcement scheme. The security property being enforced is the Nondeducibility IFP with power transmission control using FACTS devices in a smart grid network. FACTS devices are equivalent to IQCs for the smart grid network. This network is only a specific example of the architecture. It can be expanded to other problems.

Figure 8.1 illustrates the CEEME control architecture highlighting the augmentation of the existing control architecture with the use of a *"Compensator"* module. As explained previously in Section 3.1, FACTS devices are used in the smart power distribution network to control the overall power flow of the network in stressed situations (e.g. cascading failure scenario) and to provide fault tolerance. In [90], a max-flow based solution was proposed to find the control values for FACTS devices in such situations.

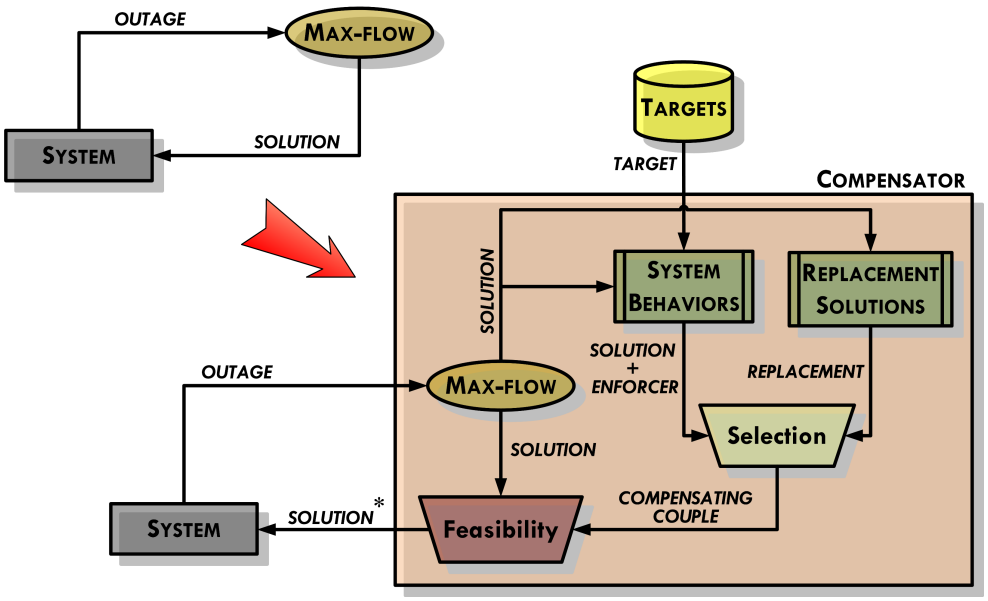


Figure 8.1: The Compensating Events based Execution Monitoring Enforcement (CEEME) Architecture

A brief explanation of max-flow and its utility in power transmission control is as follows. A flow network $G = (V, E)$ is defined as a directed graph where each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. Given a flow network with a single source s and a single sink t, the maximum-flow (max-flow) problem is to find a flow $s \rightarrow t$ that is maximum [98]. Power transmission networks can be modeled as a flow network with power flowing from generators (sources) to loads (sinks). The vertices V represent buses of the power network and the edges represent transmission lines.

When a line outage occurs, the power system attempts to sustain the original power flow by forcing flow on the residual transmission lines. This can overload other lines and lead to further outages causing a cascading failure [99]. Max-flow can be used to calculate a feasible maximum flow for the network that does not result in line overloads preventing potential cascading failures [90]. FACTS devices can be used to ensure that the solution proposed by the max-flow algorithm is enforced in the real network as a load flow (power flow) solution. Since the max-flow algorithm was designed for a single source-single sink model, Armbruster et al. [90] proposed the use of a virtual source and virtual sink that connects all generators and loads together respectively.

In the existing architecture shown in the top left corner of Figure 8.1, an instantiation of max-flow algorithm, denoted as the max-flow module, calculates a feasible maximum flow solution when the system suffers an outage. The calculated line values are sent back to the system to be enforced using the FACTS devices that are in place. From an event confidentiality perspective, the confidentiality violation occurs during this enforcement step if the actions proposed to FACTS devices produce deducible $\mathcal{D}_L$ observations.

In CEEME architecture, the outage is instead reported to the "compensator" module, which generates Nondeducibility secure settings for the FACTS devices. Although this specific example uses the smart grid as an example, the proposed architecture could produce secure settings for any IQC. The compensator module consists of several submodules – Max-Flow, System Behaviors, Replacement Solutions, Selection, and Feasibility. A description of each submodule is provided

below. In addition, it is assumed that the compensator module keeps an up-to-date flow network abstraction of the real system which is readily available to all the submodules. It is also assumed that the network is densely deployed, meaning there is a sufficient number of FACTS devices deployed in the network to control any target edge, or a FACTS device exists on each edge.

Max-Flow The max-flow submodule performs the same function it handled in the original architecture, except it sends the calculated solution the *Behavior Enforcer* and *Replacement Solutions* submodules. The Compensator achieves the functional requirement , which is to mitigate the overload, by ensuring that there exists a feasible max-flow solution, irrespective whether there is a Nondeducibility secure alternative solution. For this reason, the $solution$ is also reported to the Feasibility submodule.

System Behaviors When given the original placement of FACTS devices (as listed in the solution proposed by the Max-Flow submodule) and a list of target lines to prevent observations from occurring on, the Behavior Enforcer module will generate a list of system behavior compliant alternative max-flow solutions. The number of system behavior solutions depends on the placement of the change-origin and the targets.

Deciding what lines should be hidden is outside the scope of the compensator module. This decision would be made by a governing entity such as an operator, a system supervisor, or an engineer and fed as an external input to the System Behaviors submodule. In the absence of such external intervention, the target line can be queried from a database of previously used effective lines. With each alternative solution proposed, the System Behaviors module suggests where to place enforcers to complement the original placements.

Replacement Solutions This module works in parallel with the Behavior Enforcer module. Based on the location of the original placements and how it partitions the flow network, the Replacement Solutions module will produce a list of replacement solutions for a given proposed solution.Two external arguments – minimum and maximum tree depth – can be provided as external

inputs to bound the solution space and limit the number of additional FACTS devices required enforce the alternative solutions.

Selection This is the most important submodule in terms of Nondeducibility security yet, by definition, a very loosely defined one. The idea here is to take all possible alternative solutions generated from the System Behaviors and the Replacement Solutions submodules and select one particular solution as the final compensating couple *nondeterministically*.

If the selected solution comes from the Behavior Enforcer side, its enforcement in the actual system will have a limited number of $\mathcal{D}_L$ observations present. From Theorem (4) and (5), this solution is nondeducible secure. If the selected solution comes from the Replacement Solution side, it will have the same $\mathcal{D}_L$ projection as the originally proposed solution of the Max-Flow module. From Theorem (6) any one of the replacement solutions is nondeducible secure. This means that the output of the Selection submodule will always produce a nondeducible solution – a compensating couple for Nondeducibility IFP.

Feasibility The Feasibility module is an optional submodule that can be used to handle sparsely deployed networks or to filter solutions that only include a certain subset of deployed FACTS devices. The feasibility module uses the knowledge of the $\mathcal{D}_L$ observation matrix to determine if compensating couple proposed by the Selection submodule meets the functional requirements in terms of outage mitigation and the Nondeducibility security requirement in terms of $\mathcal{D}_L$ projection, in comparison to the original solution generated by the Max-Flow submodule.

8.1. STANDARD OPERATIONAL PROCEDURE

The standard operational procedure of handling line outages and the corresponding state transitions are shown in Figure 8.2. The system moves to what is termed an outage state from the initial steady state following a line contingency. At this point, max-flow intervenes to calculate a feasible maximum-flow solution for the system and to see if the use of IQCs can mitigate the outage. A greedy scheme

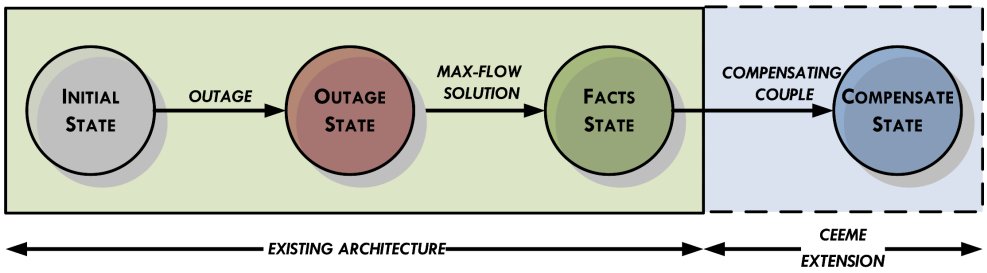


Figure 8.2: The Standard Operational Procedure of Max-Flow with CEEME Extension

is used to enforce the max-flow solution by setting IQCs with the max-flow values as its setting on the most overloaded line, until the contingency is mitigated. If the enforcement is successful, the system moves to a new state termed the FACTS state. The current architecture stops at this point since the functional requirements are met. But CEEME takes the system a step beyond to what is termed the compensate state. In addition to meeting the functional requirement, the compensate state, if it exists, provides event confidentiality for the IQC settings either through System Behaviors or Replacement Solutions.

8.2. SYSTEM BEHAVIORS ALGORITHM

Enforcing Egress Blocking and Ingress Blocking is comparatively straightforward (using FACTS devices) since both these System Behaviors follow a simple flow restriction argument. If the initial flow value (the flow value before the origin setting) is known on the target line (the line which the enforcer is on), the enforcer can set that value to prevent externally observable changes from occurring.

Routing and Redistribution however are slightly complex to enforce. Here, the enforcer(s) acts on behalf of a separate target line(s). The enforcer needs to set values in such a way so that the target line does not show observable changes. This can be achieved using a max-flow argument.

The solution generated by a max-flow algorithm will attempt to maximize the total flow from source to sink by saturating as much edges in the network as possible to their full capacities without overloading any of them. Thus, the solution directly depends on the edge capacities. However, for the same system, it is possible to artificially manipulate

line capacities on selected lines to make max-flow think that those lines have already reached their maximum capacity. In such a situation, max-flow will avoid these lines during its calculation.

This works to the advantage when enforcing Routing and Redistribution behaviors. The strategy is to find out what the load flow value was on the target line before the origin setting and artificially fix that value as the new "fake" capacity, before feeding the flow network to the max-flow algorithm. The new solution generated by max-flow, if one exists, will have the target line at its pre origin setting value or fake capacity and a max-flow compliant flow values for the rest of the edges, including the enforcer line. This basically is an alternative max-flow solution for the flow network. The argument is that, by enforcing the enforcer setting, the target line will naturally but prematurely, saturate at its flow value that it had before origin setting was committed. Thus, yielding no externally observable changes when transiting from outage state to the FACTS state.

Figure 8.3 is the program flow chart for the System Behaviors submodule. The first step is to calculate the enforcer set using the origin setting, the target list, and the current state of the flow network. The Function $CalcEnforcers(\mathscr{N}, \mathscr{T}, \mathscr{O})$ listed below, which encompasses all key attributes of each system behavior in Section 6.1 defines the logic to do this. To recap, the logic is as follows.

Consider Target, Change Origin, and Enforcer as $\mathscr{T}, \mathscr{O}$, and $\mathscr{B}$ respectively. If $\mathscr{O}$ is not an element of the subtree rooted at $\mathscr{T}$ i.e., $Subtree(\mathscr{T}) \notin \mathscr{O}$, the case is eligible for either Ingress Blocking B_b^I or Routing $R_{b,t}$ system behavior. $R_{b,t}$ is possible when none of $\mathscr{T}$'s siblings carry $\mathscr{O}$ in their subtrees. In this situation, the siblings become the $\mathscr{B}$[15]. If this is not the case, then $\mathscr{T}$ becomes $\mathscr{B}$ to form B_b^I. If $\mathscr{O}$ is an element of the subtree rooted at $\mathscr{T}$ i.e., $Subtree(\mathscr{T}) \in \mathscr{O}$ the case is eligible for Egress Blocking B_b^E or Redistribution $D_{a,b}$. For $D_{a,b}$, all $\mathscr{T}$'s children who does not include $\mathscr{O}$ in their respective subtrees can be used as $\mathscr{B}$. Otherwise, B_b^E can be used by taking $\mathscr{T}$ as $\mathscr{B}$.

Once the list of $\mathscr{B}$s is calculated, the next step is to add them as constraints. This will result in a modified flow network with artificial capacities in $\mathscr{B}$ lines.

[15]If additional solutions are required, any of these selected siblings can be replaced by all its child nodes following Definition 19

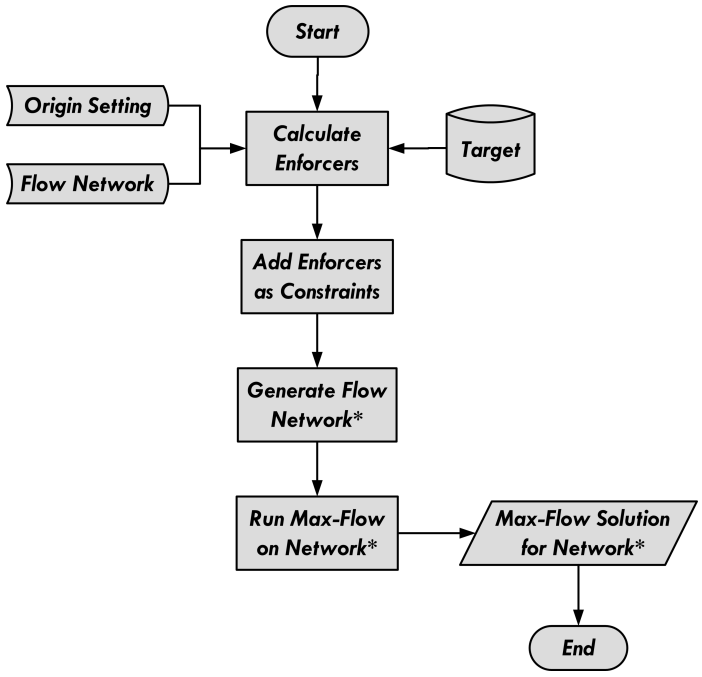


Figure 8.3: The Program Flow Chart for System Behaviors Submodule

Thus, $network \rightarrow network^*$. Another round of max-flow calculation is done on $network^*$ to generate the flow values for $\mathscr{B}$.

Algorithm 2 listed below takes in a directed weighted flow network, a change-origin, a target (list), and load flow values of the system before the first change (origin setting) occurred as input arguments and produce a system behavior compliant max-flow solution. The Function $CalcEnforcers()$ is used to determine which system behavior is appropriate and useful given a origin and a target. The Function $GetLFVal()$ is used to find the pre-origin setting load flow value of the incoming line(s) for a given vertex. The Functions $SetEdge()$ and $GetEdge()$ are used to get and set the edges respectively. The Function $RunMaxFlow()$ produce a max-flow solution for a directed weighted flow network given as its lone argument.

8.3. COMPLEXITY ANALYSIS: SYSTEM BEHAVIORS

Theorem 7. *The cost of calculating the applicable system behavior for a given change origin $\mathscr{O}$ and a target $\mathscr{T}$ in directed weighted flow network $\mathscr{N} = (V, E)$ where V is a set of vertices and E is a set of edges is $O(\mid V \mid + \mid E \mid)$.*

Algorithm 2: Behavior Enforcement Algorithm

input : A directed weighted flow network $\mathscr{N} = (V, E)$ where V is a set of vertices and E is a set of edges, the pre-origin setting load flow values of $\mathscr{N}$ $\mathscr{L}_{\mathscr{N}}$, a change-origin $\mathscr{O}$, a target $\mathscr{T}$

output: A system behavior compliant max-flow solution $\mathscr{S}$

```
B := CalcEnforcers(N, T, O)  /* Calculate the enforcer set */
Ǹ := N     /* Make a copy the network for modifications */
foreach b ∈ B do
    /* Set the pre-origin setting flow value as the new
    capacity for each enforcer line                      */
    lfVal := GetLFVal(L_N, b)
    edge := GetEdge(Ǹ.E, b)
    edge.capacity := lfVal
    Ǹ.E ← SetEdge(N.E, b, edge)
end
S := RunMaxFlow(Ǹ)   /* Calculate a max-flow solution for the
modified network */
return S
```

Proof. The problem of calculating the system behavior for a given $\mathscr{O}$ and a $\mathscr{T}$ can be reduced to a tree search problem as follows. The objective is to find $\mathscr{T}$'s relation to $\mathscr{O}$ in terms of their placement. In other words, figuring out whose subtree contains the other.

In the worst case, $\mathscr{T}$ could be the root of the representative tree of the network while $\mathscr{O}$ is one of the leaf nodes. In this case, the search cost is $| V | + | E |$ since the algorithm must visit every node V and every edge E in the network. □

8.4. REPLACEMENT SOLUTIONS ALGORITHM

Calculating all possible replacement solutions for a given initial placement is naturally a recursive process. For any given placement on a regular structure, there could exist either an $\mathcal{R}_{\blacktriangle}$ or a $\mathcal{R}_{\blacktriangledown}$ or both. Each of these candidate solutions can be further explored for additional solutions by applying replacement schemes on individual enforcers in each candidate solution recursively.

Figure 8.4 is the program flow chart for finding replacement solutions. The candidate list is basically the elements of the most recent placement considered for replacement. The idea is to recursively replace each enforcer $\mathscr{B}$ in every replacement solution found until there aren't any more possible combinations left. Every $\mathscr{B}$ partitions the network into two subnetworks. Thus, the recursive step needs to be applied on both subtrees. The solution container accumulates replacement solutions as the algorithm makes it way through the network. Avoidance list is used to prevent replacement scheme from reconsidering placements that have been already considered or found to be solutions in other combinations. Thus the minimum (min) and maximum (max) depths define the upper and lower bounds of the candidate search space. The height is zero at the root, the source node in this case, and increase downwards as the tree grows.

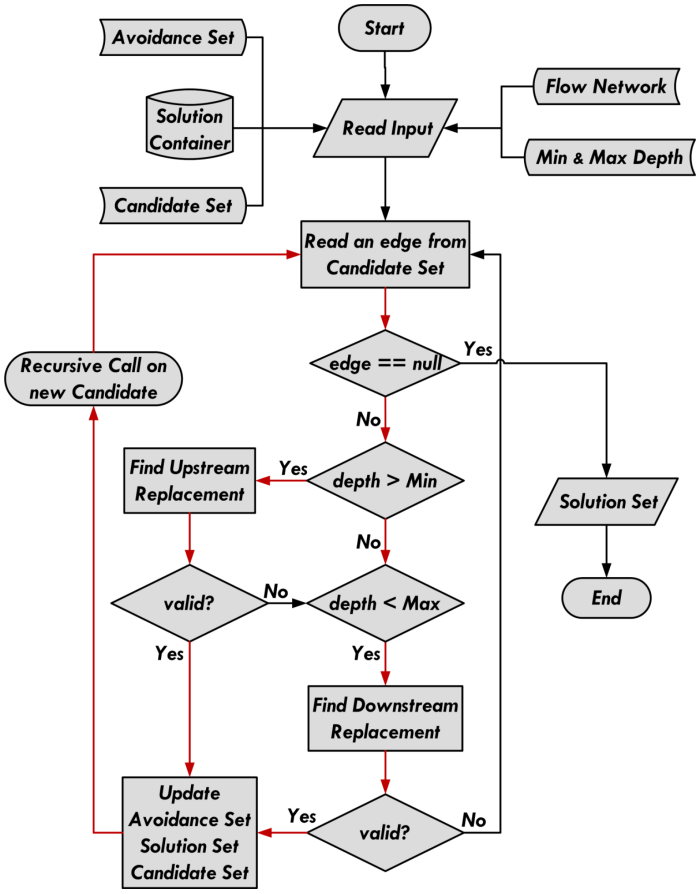


Figure 8.4: The Program Flow Chart for Replacement Solutions Submodule

CalcEnforcers($\mathscr{N}, \mathscr{T}, \mathscr{O}$)

input : A directed weighted flow network $\mathscr{N} = (V, E)$ where V is a set of vertices and E is a set of edges, the pre-origin setting load flow values of $\mathscr{N}$ $\mathscr{L}_{\mathscr{N}}$, a change-origin $\mathscr{O}$, a target $\mathscr{T}$

output: A set of enforcer nodes $\mathscr{B}$

$\mathscr{B} := \emptyset$

begin

```
/* If the origin is not a descendent of the subtree rooted
at target, consider either Routing or Ingress Blocking
system behavior                                          */
```

if $\mathscr{O} \notin$ Subtree($\mathscr{T}$) **then**

```
/* If none of the siblings of target include the origin
in their subtree, apply Routing system behavior at all
siblings of target                                       */
```

if $\mathscr{O} \notin$ Subtree($\mathscr{T}.siblings$) **then**

foreach $\mathscr{T}.sibling$ **do**

$\mathscr{B} \leftarrow \mathscr{B} \bigcup \mathscr{T}.sibling$

end

else

```
/* If one of the siblings of the target does include
origin in its subtree, apply Ingress Blocking at the
target                                                   */
```

$\mathscr{B} \leftarrow \mathscr{T}$

end

else

```
/* If the origin is a descendent of the subtree rooted
at target, consider either Redistribution or Egress
Blocking system behavior                                 */
```

if $\exists\, \mathscr{T}.child : \mathscr{O} \in$ Subtree($\mathscr{T}.child$) **then**

```
/* Apply Redistribution at child nodes of target
where the origin is not in any of their subtrees         */
```

foreach $\mathscr{T}.child : \mathscr{O} \in$ Subtree($\mathscr{T}.child$) **do**

$\mathscr{B} \leftarrow \mathscr{B} \bigcup \mathscr{T}.child$

end

else

```
/* Apply Egress Blocking at the target                   */
```

$\mathscr{B} \leftarrow \mathscr{T}$

end

end

return $\mathscr{T}_p$

end

Before exploring the $\mathcal{R}_{\blacktriangle}$ subtree, the candidate enforcer needs to be checked against the min height to see if it is below the upper bound. It this test is successful, $\mathscr{B}$ is replaced with its immediate $\mathcal{R}_{\blacktriangle}$ (see Figure 7.1(a)). At this step the recursion goes through the newly found candidate solution and replaces its component enforcers one by one, and the $\mathscr{B}$ which lead to the new solution becomes the avoidance set for the current iteration. In addition, the newly found candidate solution is added to the list of solutions found up to this point as a valid replacement solution.

If the upper bound depth check fails, the candidate is checked against the lower bound, the max depth, and is used to find a $\mathcal{R}_{\blacktriangledown}$ replacement (see Figure 7.1(b)). If a valid solution is found, the solution container is updated with the new solution, the original placement solution is marked as the avoidance set allowing a recursion on the newly found $\mathcal{R}_{\blacktriangledown}$ solution.

The replacement solution finder algorithm is listed in Algorithm 3 below. A directed weighted flow network, a solution list container which stores all possible solutions, a list of candidate edges for which replacement solutions are looking, a list of placement avoidance edges, a minimum placement depth, and a maximum placement depth are passed to the algorithm. The placement avoidance list is adopted to prevent getting edges selected repeatedly in different solutions. Specifying a maximum and minimum placement depth makes the solution search space smaller. Thus, solutions are searched bound to a given confined area.

Starting with the initial placement, which is first candidate solution, two replacement solutions, $\mathcal{R}_{\blacktriangle}$ and $\mathcal{R}_{\blacktriangledown}$ are calculated if they lie within the max and min depths. Functions *Replaceupstream*() and *ReplaceDownstream* are used for this purpose respectively; These two functions behave the same way in terms of recursion, but traverse opposite directions from the initial placement.

Having a nonempty result from Function *ReplaceUpstream*() consequent for a recursive algorithm invocation. This newly found result becomes the candidate list for the recursive call. In order to prevent traversing the tree downward, across the edge which called the *ReplaceUpstream*(), the edge is added to the placement avoidance list. This recursive process will traverse the tree upward from the original edge up to the minimum and maximum level.

Algorithm 3: Replacement Solution Finder Algorithm: FindRepSolSet($\mathscr{N}, \mathscr{S}, E_C, E_S, Min_d, Max_d$)

input : A directed weighted flow network $\mathscr{N} = (V, E)$ where V is a set of vertices and E is a set of edges, a solution list container $\mathscr{S}$, a list of candidate edges E_C, a list of placement avoidance edges E_S, minimum placement depth Min_d, maximum placement depth Max_d

output: A solution set of device placements $\mathscr{S}$

```
begin
    /* Try to replace each enforcer in every replacement
    solution with both upstream and downstream replacements */
    foreach e_c ∈ E_C do
        /* Check height upper bound for upstream replacement */
        if GetDepth(e_c) > Min_d then
            /* Find an upstream replacement candidate.  If a
            valid one exists, add to solution container      */
            result ← ReplaceUpstream(N, e_c, E_C, E_S)
            if result ≠ ∅ and result ∉ S then
                /* Recurse on the candidate                 */
                S ← FindRepSolSet(N, S + result, result, E_S +
                e_c, Min_d, Max_d)
            end
        end
        /* Check height lower bound for downstream replacement
        */
        if GetDepth(e_c) < Max_d then
            /* Find an downstream replacement candidate.  If a
            valid one exists, add to solution container      */
            result ← ReplaceDownstream(N, e_c, E_C, E_S)
            if result ≠ ∅ and result ∉ S then
                /* Recurse on the candidate                 */
                S ← FindRepSolSet(N, S + result, result, E_S +
                e_c, Min_d, Max_d)
            end
        end
    end
    return S
end
```

Function ReplaceUpstream($\mathcal{N}, \mathcal{T}, E_C, E_S$)

input : A directed weighted flow network $\mathcal{N} = (V, E)$ where V is a set of vertices and E is a set of edges, a target device to replace $\mathcal{T}$, a list of candidate edges E_C, a list of placement avoidance edges E_S

output: An Upstream Replacement solution candidate

```
begin
    result ← E_C      /* Start building a candidate with using the
    target as a reference */
    /* Add all parents of target not in the avoid list to the
    candidate                                                    */
    foreach T.parent do
        if T.parent ∉ E_S then
            result ← result + T.parent
        end
    end
    /* Add all siblings of target not in the avoid list to the
    candidate                                                    */
    foreach T.sibling do
        if T.sibling ∉ E_S then
            result ← result + T.sibling
        end
    end
    /* Check if this combination is already considered.  If
    not finalize the candidate by removing the target            */
    if result ≠ E_C then
        result ← result − T
    else
        /* If not unique combination is invalid                 */
        result ← ∅
    end
    return result
end
```

Function ReplaceDownstream($\mathcal{N}, \mathcal{T}, E_C, E_S$)

input : A directed weighted flow network $\mathcal{N} = (V, E)$ where V is a set of vertices and E is a set of edges, a target device to replace $\mathcal{T}$, a list of candidate edges E_C, a list of placement avoidance edges E_S

output: An Downstream Replacement solution candidate

begin

result $\leftarrow E_C$ /* Start building a candidate with using the target as a reference */

/* Add all children of target not in the avoid list to the candidate */

foreach $\mathcal{T}.child$ **do**

if $\mathcal{T}.child \notin E_S$ **then**

result $\leftarrow$ result $+ \mathcal{T}.child$

end

end

/* Check if this combination is already considered. If not finalize the candidate by removing the target */

if result $\neq E_C$ **then**

result $\leftarrow$ result $- \mathcal{T}$

else

result $\leftarrow \emptyset$

end

return result

end

Similarly, *ReplaceDownstream*() traverse the tree downward from the original edge up to the maximum allowed level. As an observation, recursive callings of *ReplaceUpstream*() may call both *ReplaceUpstream*() and *ReplaceDownstream*() at the next level, while *ReplaceDownstream*() only calls *ReplaceDownstream*() in its subsequent calls. This difference is clearly distinguishable in the function implementations for $\mathcal{R}_{\blacktriangle}$ and $\mathcal{R}_{\blacktriangledown}$ listed below, where Function *ReplaceUpstream()* iterates on both parents and siblings while Function *ReplaceDownstream()* iterates only on its children. After the recursive step returns, all the remaining edges in the candidate list of the previous call continuous to follow the same procedure. The solution set that returns once the all the candidate in the first set has been considered consists of all possible solutions under the constraints of the maximum-minimum depth and placement avoidance set.

8.5. COMPLEXITY ANALYSIS: REPLACEMENT SOLUTIONS

Theorem 8. *The cost of calculating all replacement solutions for a given initial placement in a directed weighted flow network $\mathcal{N} = (V, E)$ where V is a set of vertices and E is a set of edges is $O(E^2)$.*

Proof. Consider the height of the initial placement from the root as h, the height of the $\mathcal{R}_{\blacktriangledown}$–tree as $h_{max} - h = h_d$, height of the $\mathcal{R}_{\blacktriangle}$–tree as $h - h_{min} = h_u$. The root is at height h_{min} while the leaf nodes are at height h_{max}. Thus, the total time complexity of finding all replacement solutions with the initial placement at height h is,

$$T(h) = D(h_d) + U(h_u) \tag{21}$$

where $D(h_d)$ is the time complexity of finding $\mathcal{R}_{\blacktriangledown}$ solutions and $U(h_u)$ is the time complexity of finding $\mathcal{R}_{\blacktriangle}$ solutions. For $D(h_d)$, a recurrence relation can be written as follows:

$$D(h_d) = k \cdot D(h_d - 1) + c \cdot f(h_d) \tag{22}$$

The cost of calculating $\mathcal{R}_{\blacktriangledown}$ solutions for a tree of height h_d is the sum of calculating the cost for the subtrees rooted at each of the k child nodes below the

original placement plus the overhead involved in transforming the problem into sub problems $f(h_d)$. The height of each child node is $h_d - 1$ thus, the cost associated with each subtree is $D(h_d - 1)$. $c > 0$ is a constant. The recurrence in Equation (22) can be solved using the recursion-tree method as follows:

$$\begin{aligned} D(h_d) &= k \cdot D(h_d - 1) + c \cdot f(h_d) \\ &= k \cdot [k \cdot D(h_d - 2) + f(h_d - 1)] + c \cdot f(h_d) \\ &= k^2 \cdot D(h_d - 2) + k^1 \cdot f(h_d - 1) + c \cdot f(h_d) \\ &\vdots \\ &= k^{h_d - 1} \cdot D(1) + k^{h_d - 2} \cdot f(2) + k^{h_d - 3} \cdot f(3) + \ldots + c \cdot f(h_d) \end{aligned} \tag{23}$$

The cost of calculating replacement solutions at the base case $D(1) = 1$. Also, k is a constant that is fixed for given topology. The amount of work done to transform the problem into subproblems $\forall 0 \leq i \leq h_d - 2 : f(h_d - i)$ is constant time $O(1)$ since to move from height h_d to $h_d - 1$, the original placement is moved to child nodes. Thus, the Equation (23) can be reduced to the following form:

$$\begin{aligned} D(h_d) &= k^{h_d - 1} + [1 + k + k^2 + k^3 + \ldots + k^{h_d - 2}] \\ D(h_d) &= \sum_{i=0}^{h_d - 1} k^i \\ &= \frac{k^{h_d} - 1}{k - 1} \end{aligned} \tag{24}$$

For $U(h_u)$, the total time complexity is the sum of calculating $\mathcal{R}_\blacktriangledown$ solutions for each of the $\mathcal{R}_\blacktriangledown$–subtrees of the $\mathcal{R}_\blacktriangle$–tree. Each time the recursion moves a level up the tree, there are $(k - 1)$ siblings that are considered for $\mathcal{R}_\blacktriangledown$. Thus, the recurrence equation takes the following form:

$$\begin{aligned} U(h_u) &= (k - 1) \cdot D(h_d) + U(h_d + 1) + c' \cdot g(h_u - (h_u - 1)) \\ &= (k - 1) \cdot D(h_d) + U(h_d + 1) + c' \cdot g(1) \end{aligned} \tag{25}$$

Using recursion-tree method, the recurrence in Equation (25) can be solved as follows:

$$\begin{aligned}
U(h_u) &= (k-1)\cdot D(h_d) + U(h_d+1) + c'\cdot g(1) \\
&= (k-1)\cdot D(h_d) + (k-1)\cdot D(h_d+1) + U(h_d+2) + c'\cdot g(2) + c'\cdot g(1) \\
&= (k-1)\cdot D(h_d) + (k-1)\cdot D(h_d+1) + \dots \\
&\qquad + (k-1)\cdot D(h_d+h_u) + c'\cdot g(h_u) + \dots + c'\cdot g(1) \\
&\vdots \\
&= (k-1)\cdot \sum_{j=0}^{h_u} D(h_d+j) + c'\cdot \sum_{l=1}^{h_u} g(l)
\end{aligned} \tag{26}$$

Here also, $k-1$ is a constant that is fixed for given topology. $c' > 0$ is also a constant. the amount of work done to transform the problem into subproblems $\forall 1 \le l \le h_u : g(l)$ is constant time $O(1)$ since to move from height h_d to h_d+1, the original placement is moved to siblings and the immediate parent. Thus, the Equation (26) can be reduced to the following form:

$$\begin{aligned}
U(h_u) &= (k-1)\cdot \sum_{j=0}^{h_u} D(h_d+j) + c'\cdot \sum_{l=1}^{h_u} g(l) \\
&= (k-1)\cdot \sum_{j=0}^{h_u} D(h_d+j) + c'\cdot h_u
\end{aligned} \tag{27}$$

The combination of Equation (24) and Equation (27), which is the total time complexity $T(h)$, is given in Equation 28. Note that $h_d + h_u = h_{max} - h_{min}$, which is the height of the original tree that is searched for replacement solutions. $h_{max} - h_{min} \simeq lg_k(E)$. $(k-1)$ is a constant for a given topology and $h_u = h_{max} - h_{min} - h_d$. In worse case scenario, $h_d = 0$ and $h_u = h_{max} - h_{min}$, which is bounded by $lg_k(E)$. For $k > 0$, $\frac{1}{k-1} < 1$.

$$
\begin{aligned}
T(h) &= D(h_d) + U(h_u) \\
&= \frac{k^{h_d}-1}{k-1} + (k-1)\cdot\sum_{j=0}^{h_u} D(h_d+j) + c'\cdot h_u \\
&= \frac{k^{h_d}-1}{k-1} + (k-1)\cdot\sum_{j=0}^{h_u} \frac{k^{h_d+j}-1}{k-1} + c'\cdot h_u \\
&= \frac{k^{h_d}-1}{k-1} + \sum_{j=0}^{h_u} k^{h_d+j} - 1 + c'\cdot h_u \\
&= \frac{k^{h_d}-1}{k-1} + \sum_{j=0}^{h_u} k^{h_d+j} - \sum_{j=0}^{h_u} 1 + c'\cdot h_u \\
&= \frac{k^{h_d}-1}{k-1} + \sum_{j=0}^{h_u} k^{h_d+j} + (c'-1)\cdot h_u \\
&= \frac{k^{h_d}-1}{k-1} + \frac{k^{h_d}(k^{h_u+1}-1)}{k-1} + (c'-1)\cdot h_u \\
&= \frac{k\cdot k^{h_d+h_u}-1}{k-1} + (c'-1)\cdot h_u \\
&= \frac{k\cdot k^{lg_k(|E|)}}{k-1} - \frac{1}{k-1} + (c'-1)\cdot lg_k(E) \\
&= \frac{k\cdot k^{lg_k(|E|)}}{k-1} + C\cdot lg_k(E) \\
&= \frac{k}{k-1}\cdot|E| + C\cdot lg_k(E) \\
&= C''\cdot|E| + C\cdot lg_k(E) \\
&= O(|E|)
\end{aligned}
\tag{28}
$$

Thus, the total complexity in finding replacement solutions for a given tree with E edges is bounded by $O(E)$. However, this calculation has to be performed over all $e_c \in E_C$, which in worst case, has all edges of the tree. Thus, the time complexity of the replacement solution algorithm is bounded by $O(E^2)$. □

9. CEEME ON THE IEEE 118-BUS SYSTEM

As a proof of concept, this chapter presents the application of System Behaviors and Replacement Solutions schemes on several well studied cascading failure scenarios in the standard IEEE 118-bus test system. At a conceptual level, the two confidentiality models presented in this work should be applicable to irregular structures with little to no modification since the flow conservation rule applies to irregular flow networks the same way it applies to regular flow networks. Both System Behaviors and Replacement Solutions models are based on a flow conservation rule thus, a hypothesis can be made that *System Behaviors and Replacement Solutions are applicable for irregular networks*. However, the algorithms that have been proposed to calculate and them would require some modification before they could be readily used. For that reason, CEEME in its present implementation can not be directly applied to irregular structures.

Even though these standard test systems fall outside the regular structures considered in this dissertation, techniques exist in literature to parse irregular structures into regular series-parallel digraphs [100–102]. This is in fact the next logical extension to this body of work – analyze irregular structures under compensating events and IFPs. Even without doing a rigorous reduction, it is still possible to identify mix connected flow networks within the 118-bus test network, which was used in the analysis below.

Figure 9.1 shows the standard IEEE 118-bus test system[16]. A thorough cascading failure scenarios analysis for this test system has been done in [103, 104]. Out of these, this work considers test cases $4-5$, $37-39$, and $47-69$.

Figure 9.2 is the flow network representation of the 118-bus system. Here, the vertices (V) and edges (E) represent buses and transmission lines respectively. Buses with generators attached are represented in orange while buses with loads attached are represented in Brown. The yellow colored node 69 is the swing bus of the system. Dotted edges represent no power flow, Red colored edges represent overloaded lines, Green colored edges represent IQC controlled lines. Blue

[16]Image Source: Power Systems Test Case Archive `http://www.ee.washington.edu/research/pstca/`

colored edges with a label "U" represent an increase in flow in comparison to another state as seen by a $\mathcal{D}_L$ observer. Similarly, brown colored edges with a label "D" represents a decrease in flow. A change in observation is considered anything above a 1% difference in readings between states. This ensures a high resolution in $\mathcal{D}_L$ observability. For this system FACTS devices act as the IQCs.

9.1. LINE OUTAGE AT E_{4-5}

Figure 9.3 shows the sequence of state transitions as the system goes through the operational procedure listed in Section 8.1, following the line outage at e_{4-5}. Figure 9.3(a) is the outage state with e_{5-11} indicating a line overload. A greedy FACTS placement on the overloaded line mitigates the overload [104] as shown in Figure 9.3(b). The comparison is made against the outage state to show the $\mathcal{D}_L$ observations.

Due to the irregular connectivity, the $\mathcal{D}_L$ observations of the initial FACTS placement do not propagate through the network in the same manner regular networks[17]. However, the inheritance rules defined under Section 3.1.3 still hold. The $\mathcal{D}_L$ observation on e_{5-11} flips for the sibling e_{5-6} and replicates for e_{5-8}. The main source of power in this case is flowing from the generator at v_{10}. However, between the outage state and the FACTS state, the flow from the generator does not change. As a result, the decrease in flow on e_{5-8} is attributed to the increase in flow in e_{8-30}. The excess power flow to v_{11} through the path $v_8 \to v_{30} \to v_{17} \to v_{18} \to v_{12} \to v_{11}$ to meet its load demand.

Figure 9.3(c) shows the first $\mathcal{R}_{\blacktriangle}$ of the FACTS device at e_{5-11}. Except for a flow decrease from $v_{69} \to v_{68} \to v_{65}$, there are no $\mathcal{D}_L$ observable changes in this state in comparison to the FACTS state in Figure 9.3(b). What this means is that the original placement shown in the FACTS state has at least one nondeducible $\mathcal{R}_{\blacktriangle}$ solution since both the original placement and its $\mathcal{R}_{\blacktriangle}$ produce the same $\mathcal{D}_L$ projection.

However, the $\mathcal{R}_{\blacktriangledown}$ was not as successful in obfuscating the initial FACTS placement as shown in Figure 9.3(d). Placing a FACTS on e_{4-11} is impossible it would restrict the sink node at v_4 from meeting its load demand. The unintended side

[17] Only the subnetworks with $\mathcal{D}_L$ observations are shown in the subsequent figures of this chapter

effect of placing a FACTS device on the only eligible $\mathcal{R}_{\blacktriangledown}$ candidate at e_{11-13} caused the initial overload to reappear. Not only that, there are a lot of $\mathcal{D}_L$ observation differences between the original placement and the $\mathcal{R}_{\blacktriangledown}$.

Because the $\mathcal{D}_L$ projection of the change-origin is limited to a subnetwork, there are only a handful of meaningful targets for System Behaviors. e_{8-30} is one of them. With v_{10} being the primary source, the relationship between the target and the origin allows a Ingress Block at e_{8-30}. This is illustrated in Figure 9.3(e) with a comparison made against the outage state. The target, however, is in a critical path between $v_{10} \rightarrow v_{11}$ and would not be able to hide all $\mathcal{D}_L$ observations under the subtree rooted at v_{30}. Yet, there is a significant reduction in $\mathcal{D}_L$ observations due to B^I_{30}, especially in subnetworks that branch off the critical path such as $v_{26} \rightarrow v_{30}$ and $v_{15} \rightarrow v_{17}$.

A Redistribution at v_6 with target at v_5 is illustrated in Figure 9.3(f). The $\mathcal{D}_L$ projection of this state is very similar to that of B^I_{30} in Figure 9.3(e), with only a few exceptions, most notably, the subtree that spans out v_{26}. The similarities in these two states outweigh the differences by a considerable margin. Additionally, there is still a considerable reduction in the overall number of $\mathcal{D}_L$ observations compared to the FACTS states. Interestingly enough, according to Theorems (4) and (5), these are not System Behaviors from the same Nondeducibility family. This observation not only reenforces the earlier hypothesis, but also hints at the existence of a super family of nondeducible System Behaviors for irregular structures. Based on Theorem 4, a simple nondeducible counterpart for Redistribution $D_{6,5}$ would be Egress Blocking B^E_5. It is highly likely that all three of these System Behaviors will produce nondeducible solutions in irregular structures.

9.2. LINE OUTAGE AT E_{37-39}

Figures 9.4 and 9.5 show the sequence of state transitions as the system goes through the operational procedure following a line outage at e_{37-39}. Figure 9.4(a) is the outage state with e_{37-40} indicating a line overload. A greedy FACTS placement on the overloaded line mitigates the overload [104] as shown in Figure 9.4(b). The comparison is made against the outage state to show the $\mathcal{D}_L$ observations.

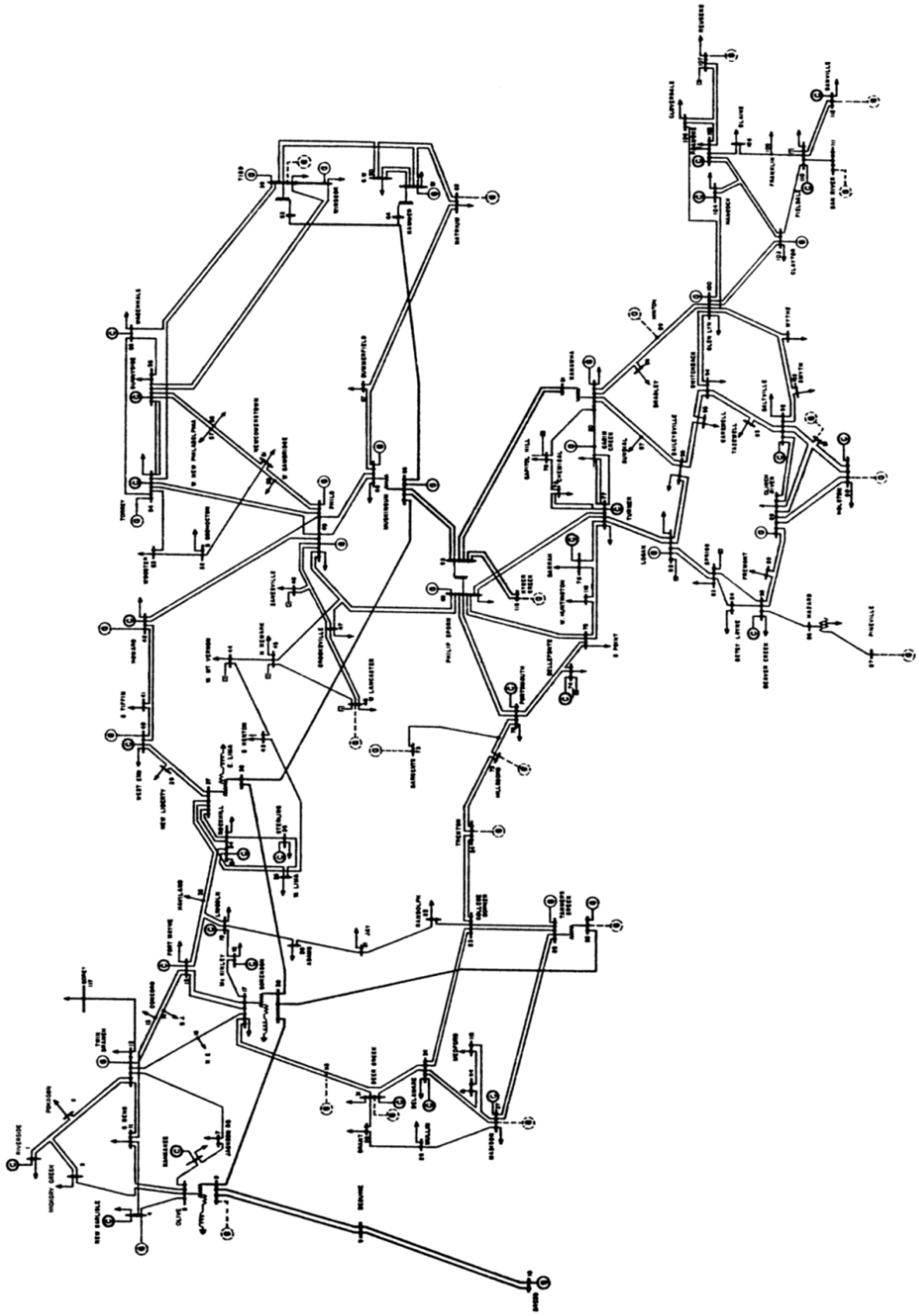


Figure 9.1: The Standard IEEE 118-bus Test System

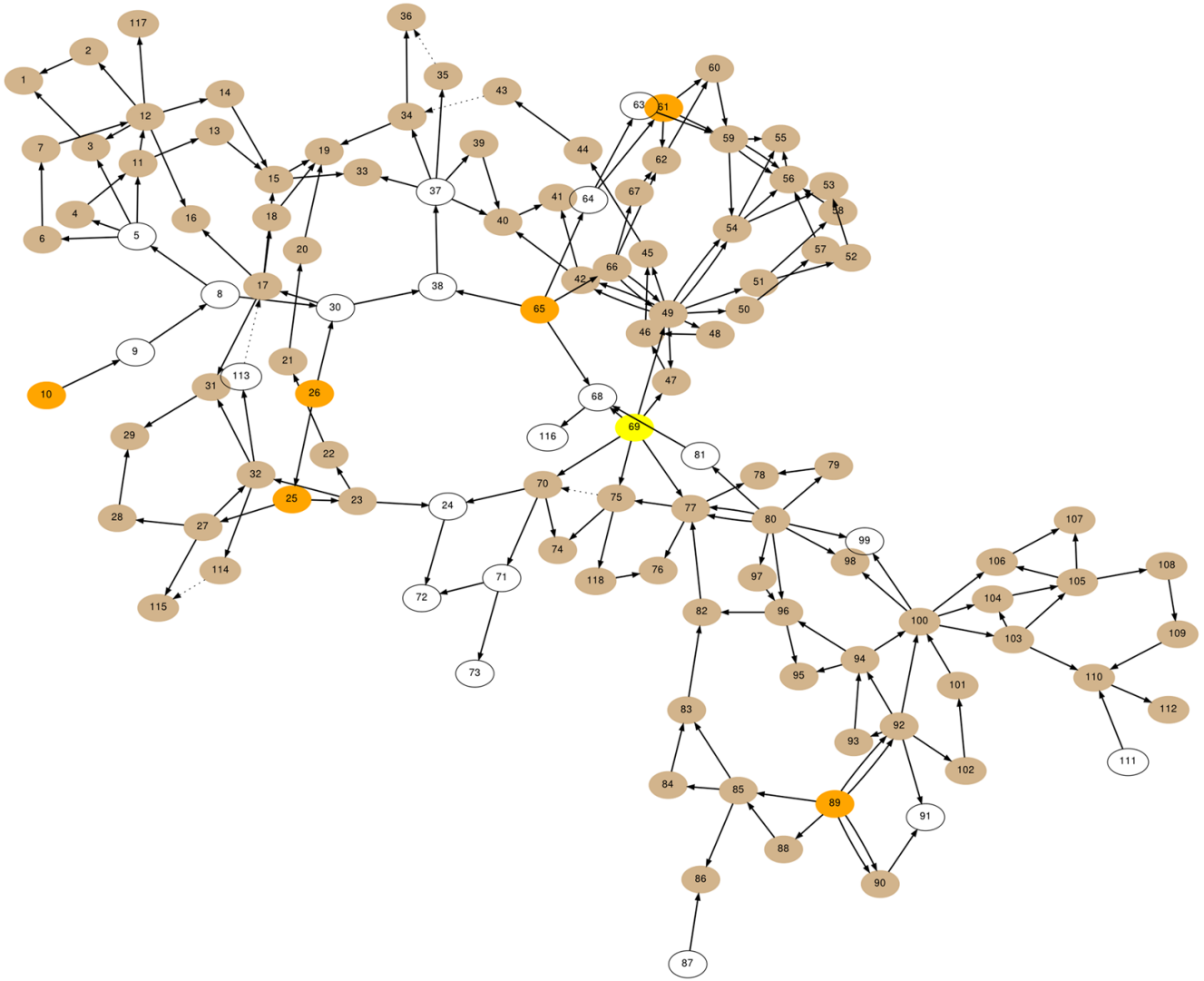
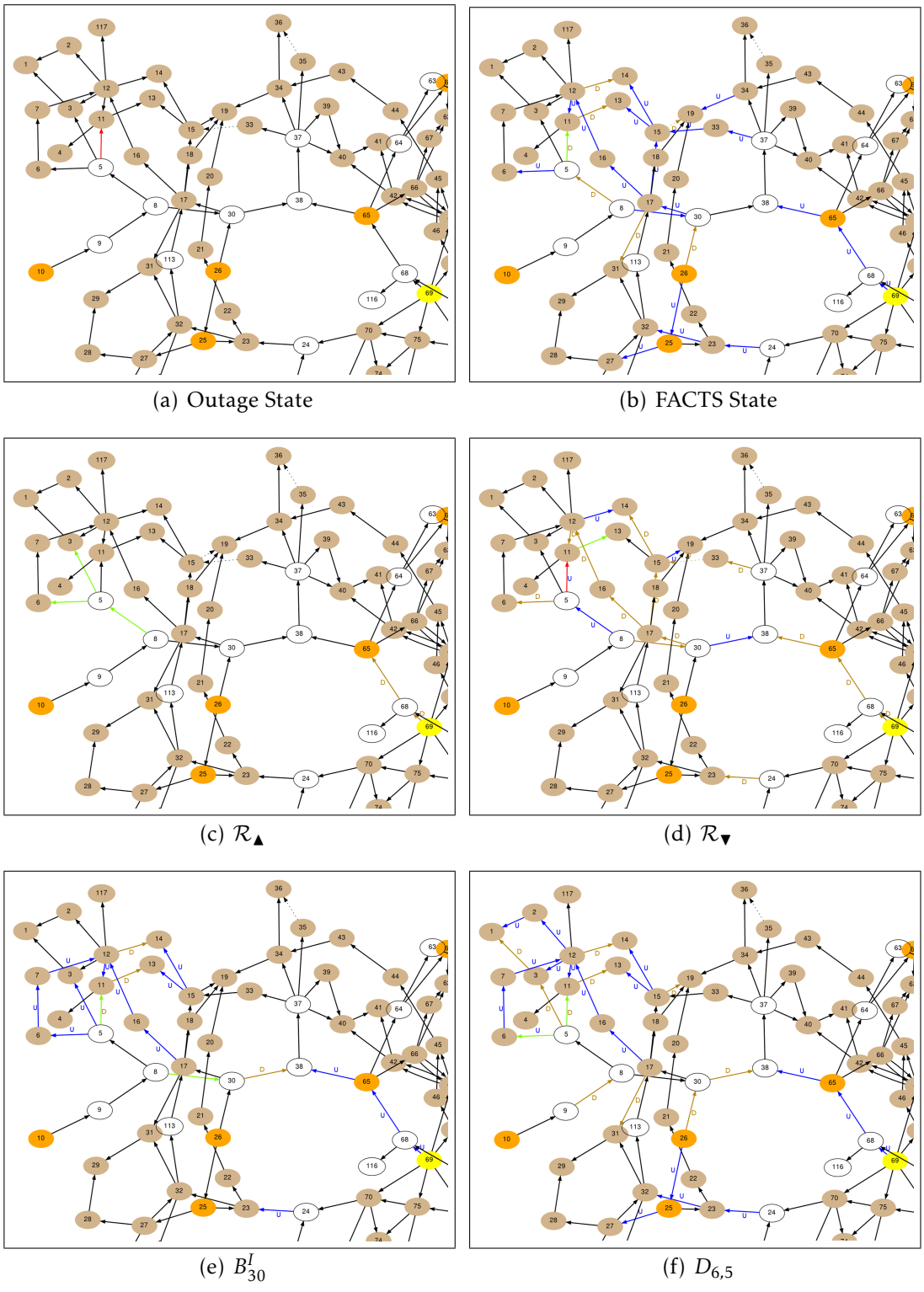


Figure 9.2: A Flow Network Representation of the 118-bus Test System



(a) Outage State (b) FACTS State

(c) $\mathcal{R}_{\blacktriangle}$ (d) $\mathcal{R}_{\blacktriangledown}$

(e) B^{I}_{30} (f) $D_{6,5}$

Figure 9.3: Line Outage e_{4-5} States

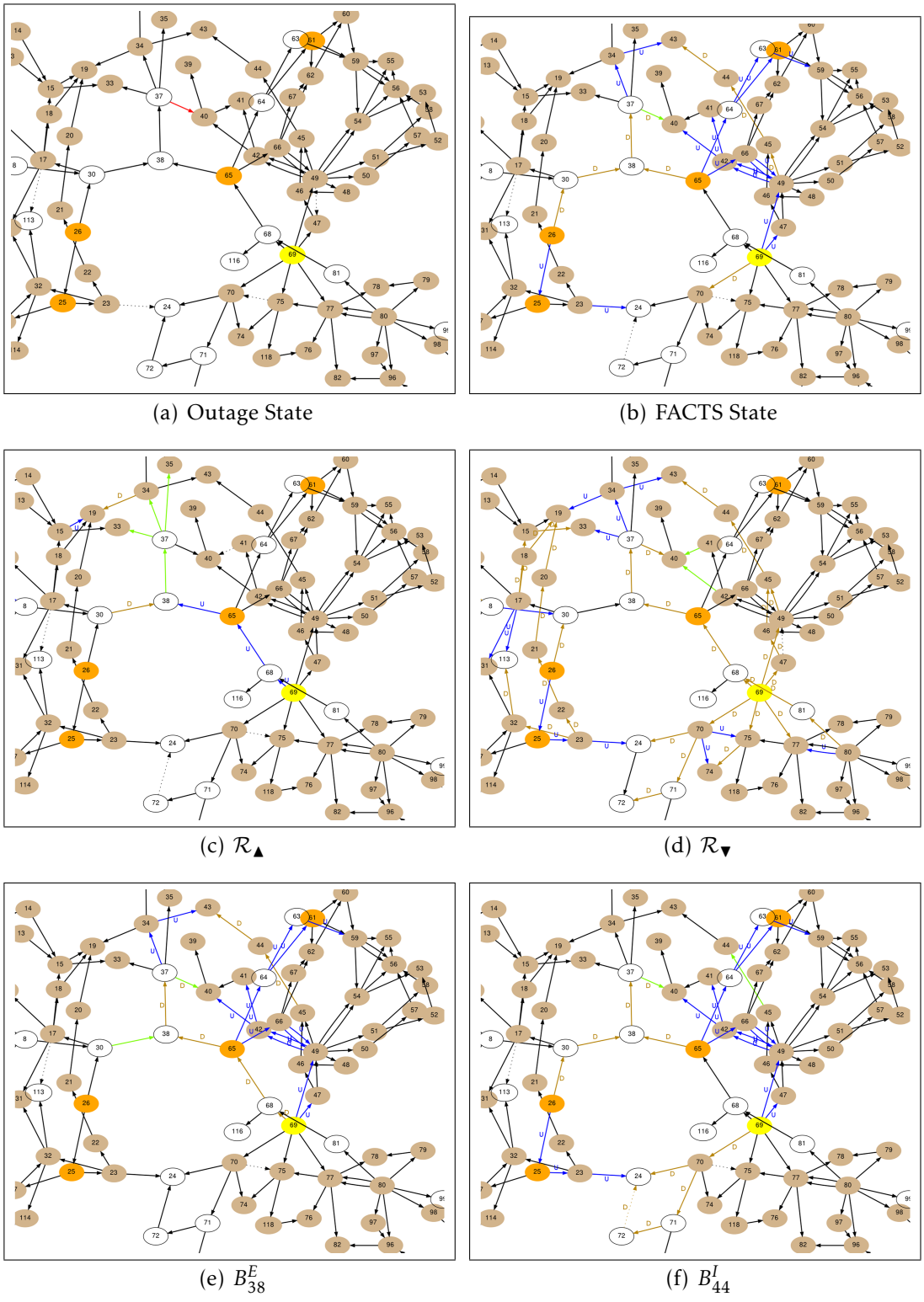
Figure 9.4(c) shows the $\mathcal{R}_{\blacktriangle}$ of the FACTS device at e_{37-40}. In comparison to the respective FACTS state in Figure 9.4(b), this replacement only has a small number of $\mathcal{D}_L$ observable changes. Even the ones that appear in Figure 9.4(c) are miniscule in comparison to the outage state. Thus, the original placement shown in the FACTS state has at least one nondeducible $\mathcal{R}_{\blacktriangle}$ solution, which in turn preserves event confidentiality and Nondeducibility IFP.

Similar to the line outage e_{4-5} case study above, the $\mathcal{R}_{\blacktriangledown}$ fails since the number of $\mathcal{D}_L$ observations significantly increased in comparison to the FACTS state. This is shown in Figure 9.4(d). From a theoretical perspective, placing a FACTS device on e_{39-40} is not possible since v_{39} is a sink node. As an alternative, the other paths to v_{40} were restricted to try to mimic a $\mathcal{R}_{\blacktriangledown}$. Unfortunately, this only resulted in producing massive amounts of additional $\mathcal{D}_L$ observations.

In Figure 9.4(e), an Egress Block at v_{38} (B^E_{38}) was considered. This produced favorable results in one incoming path by removing all $\mathcal{D}_L$ observations in e_{30-38}, e_{26-30}, e_{25-26}, e_{23-24}. The node v_{38} is irregular by nature since there are two parent nodes – v_{30} and v_{65}. The $\mathcal{D}_L$ observations on the path through v_{65} however, did not disappear as expected.

Figure 9.4(f) shows Ingress Block B^I_{44} at v_{44}. As expected, $\mathcal{D}_L$ observations in the path $v_{45} \rightarrow v_{44} \rightarrow v_{43}$ were removed by this system behavior. In addition, observations in the path $v_{37} \rightarrow v_{34} \rightarrow v_{43}$ were also removed mimicking a $D_{34,37}$. This is another instance where the two mutually exclusive nondeducible system behavior families for regular network overlap; Redistribution and Ingress Blocking are from different nondeducible system behavior families. Another interesting observation is that these two System Behaviors are on two different paths that lead to a shared sink.

Figure 9.5(a) shows Ingress Block B^I_{64} at v_{64}. The expected outcome of this blocking behavior is to remove the $\mathcal{D}_L$ observations in the subtree rooted at v_{64}. This was successfully achieved as shown in the corresponding illustration. The nondeducible counterpart of B^I_{64}, which is $R_{66,64}$ also achieved the same goal with very few discrepancies in $\mathcal{D}_L$ projection. In comparison, most differences in $\mathcal{D}_L$ projection occur around v_{69}.

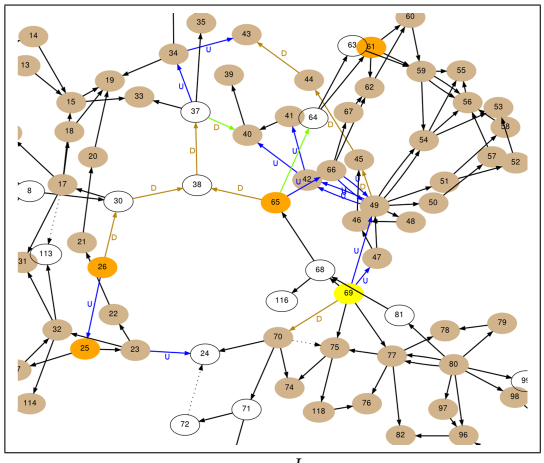


(a) Outage State (b) FACTS State

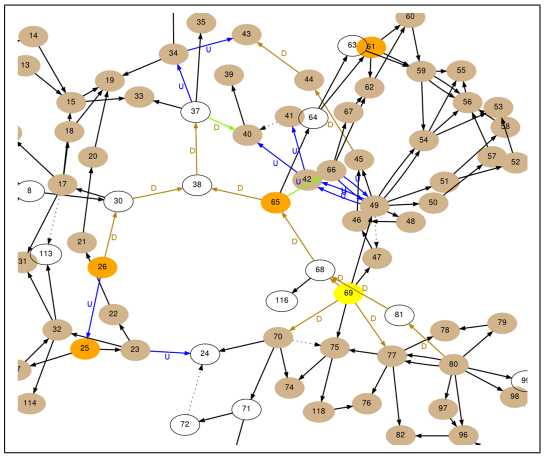
(c) $\mathcal{R}_{\blacktriangle}$ (d) $\mathcal{R}_{\blacktriangledown}$

(e) B^{E}_{38} (f) B^{I}_{44}

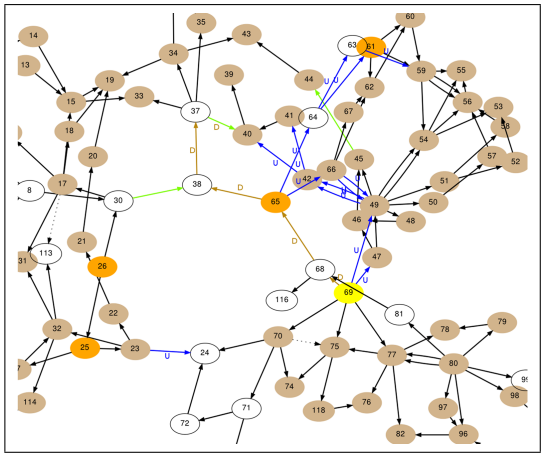
Figure 9.4: Line Outage e_{37-39} States I



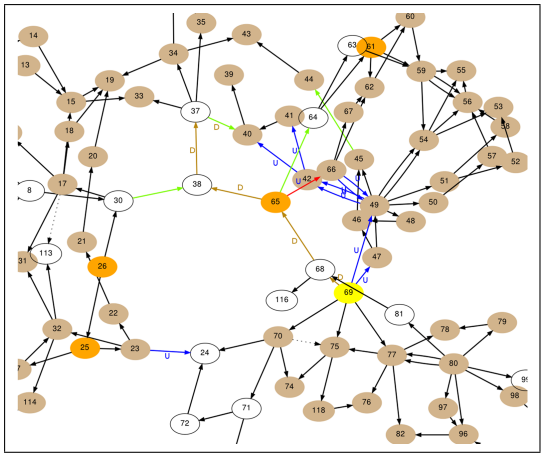
(a) B^I_{64}



(b) $R_{66,64}$



(c) $B^E_{38} + B^I_{44}$



(d) $B^E_{38} + B^I_{44} + B^I_{64}$

Figure 9.5: Line Outage e_{37-39} States II

To the naked eye, it can be seen that the different System Behaviors eliminate disjoint sets of $\mathcal{D}_L$ observations. Two experiments were carried out to see if composing System Behaviors can eliminate a larger set of observations. In the first one, $B^E_{38}+B^I_{44}$ were enforced together. The result of this composed system behavior enforcement is shown in Figure 9.5(c). Interestingly, this did produce the intended result. Figure 9.5(c) in comparison to Figures 9.4(e) and 9.4(f) is a good example for this. Going a step further, a third system behavior B^I_{64} was composed with the $B^E_{38}+B^I_{44}$ to produce $B^E_{38}+B^I_{44}+B^I_{64}$ as shown in Figure 9.5(d). Although the number of System Behaviors significantly reduced, restricting the flow in basically all outgoing branches of v_{65} caused the original overload to reappear. Having said that, it is encouraging to observe the possibility of composing System Behaviors to achieve better results in terms of Nondeducibility and $\mathcal{D}_L$ projection pruning.

9.3. LINE OUTAGE AT E_{47-69}

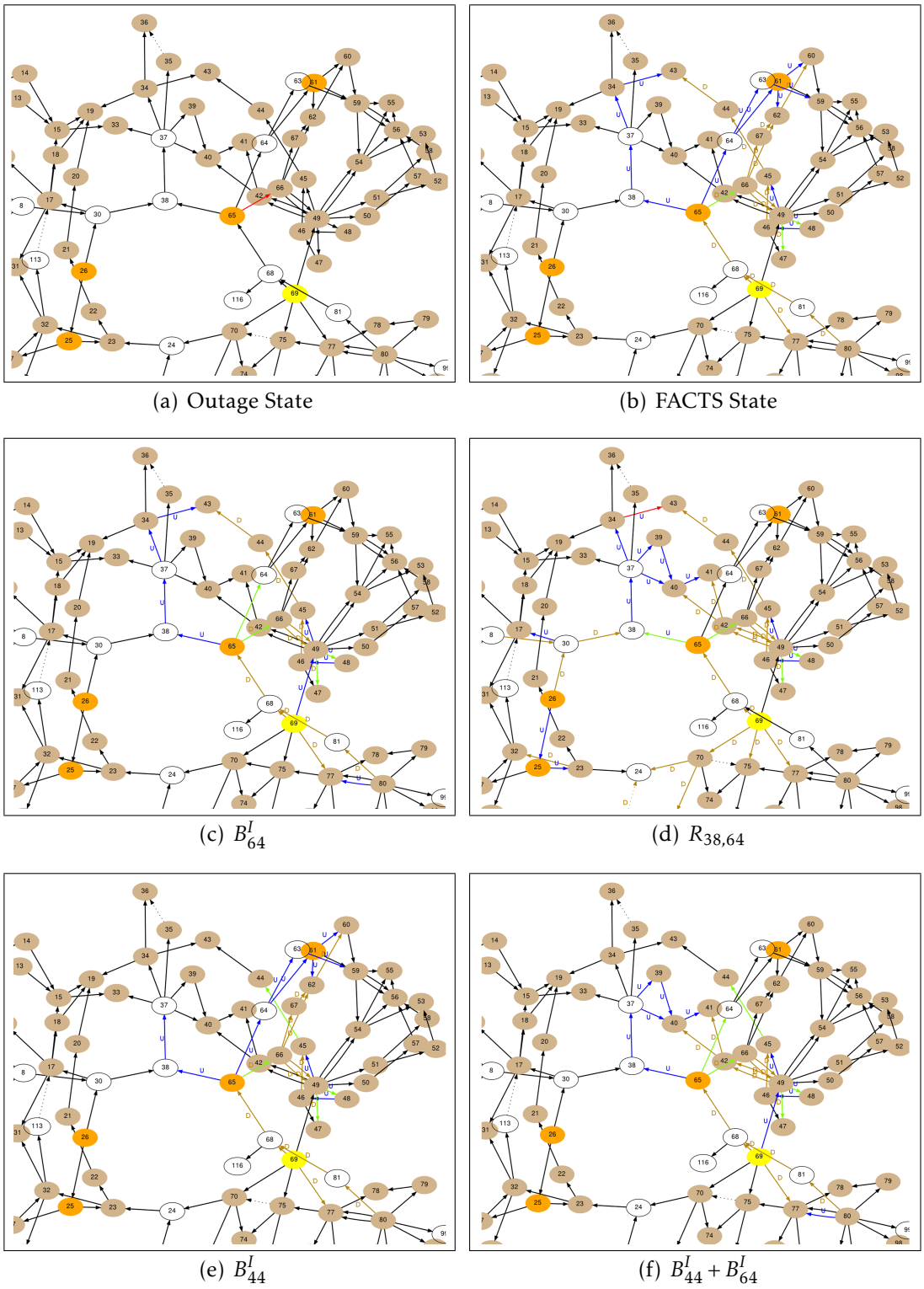
Figure 9.6 shows the sequence of state transitions as the system goes through the operational procedure following the line outage at e_{47-69}. Figure 9.6(a) is the outage state with e_{47-69} indicating a line overload. As noted in [104], the greedy FACTS placement approach can not be directly applied in this case here since more lines end up overloading. Max-flow assumes a lossless load flow system. The actual power distribution system however, is not without losses in lines due to reactive power. However setting values just below what max-flow suggested on e_{65-66}, e_{47-49}, and $, e_{48-49}$ will remove all overloads from the system [104]. This is shown in Figure 9.6(b), which is compared against the outage state to show the $\mathcal{D}_L$ observations.

Figure 9.6(e) shows Ingress Block B^I_{44} at v_{44}. The expected outcome of this blocking behavior is to remove the $\mathcal{D}_L$ observations on the path $v_{45} \rightarrow v_{44} \rightarrow v_{43}$, which has been successfully achieved. Similar to the earlier analysis on Figure 9.4(f), additional $\mathcal{D}_L$ observations on the path $v_{37} \rightarrow v_{34} \rightarrow v_{43}$ have also disappeared.

The Ingress Block B^I_{64} shown in Figure 9.6(c) not only removed all $\mathcal{D}_L$ observations of the subtree rooted at v_{64} but also the ones of the subtree rooted at v_{66} with the exception of the v_{49-66} outbound branch. The nondeducible counterpart

of B^I_{64}, which is $R_{38,64}$ shown in Figure 9.6(d) also removed $\mathcal{D}_L$ observations of the subtree rooted at v_{64}. However, the latter produced a line overload in $e_{34,43}$ which was mainly due to the flow decrease from $v_{26\rightarrow v_{38}}$ that resulted in a flow increase in the path $v_{38} \rightarrow v_{37} \rightarrow v_{34} \rightarrow v_{43}$.

A system behavior composition of $B^I_{44} + B^I_{64}$ produced favorable results in terms of further reducing the number of $\mathcal{D}_L$ observation in comparison to enforcing them individually. This is shown in Figure 9.6(f).



(a) Outage State

(b) FACTS State

(c) B^I_{64}

(d) $R_{38,64}$

(e) B^I_{44}

(f) $B^I_{44} + B^I_{64}$

Figure 9.6: Line Outage e_{47-69} States

10. CONCLUSIONS AND FUTURE WORK

10.1. CONCLUSIONS

This research is driven by the desire to develop a fundamentally sound CPS security framework that accounts for the inherent physical observability of cyber-physical interactions, with a far-reaching objective of developing a science of self-obfuscating systems based on the composition of simple building blocks. To this end, the most important contribution of this dissertation is the development of a **foundation** for a CPS security framework by taking IFP as a suitable medium. Specific attention was given to $\mathcal{D}_H$ event confidentiality violations, through $\mathcal{D}_L$ external observations. The problem was theoretically formulated and modeled as a Nondeducibility security violation that arises due the inherent cyber-physical interaction of the system.

Most, if not all, realistic CPS systems are irregular structures with complex inter and intra connections. Analyzing and characterizing such systems is not an easy feat by any means simply due to the complexity involved. From a security standpoint, the lack of readily available established security models for such systems only worsens the situation and demands a more fundamental bottom up approach of building a theory. Thus, this work followed the natural progression of system analysis of going from regular structures to irregular structures.

10.2. CHAPTER BY CHAPTER CONTRIBUTIONS

- There are different substructures even within regular structures based on their connectivity – series, parallel, and mix (series-parallel) connected. In Chapter 3, regular structures were analyzed for their inherent ability to leak $\mathcal{D}_H$ information to $\mathcal{D}_L$ external observers in terms of violating the Nondeducibility security property. This chapter:
  - Expressed the physical flow of the system in terms of a system invariant that describes both the commodity flow (physical) and the information flow (cyber).

  - Analyzed event confidentiality violation in regular series, parallel, and mix connected networks.
  - Derived the minimum number of observers required for Parallel networks and extended it to mix connected networks.
  - Derived theorems on series characteristics and partial deducibility of mix connected networks.
  - Defined an algorithm to construct a $\mathcal{D}_L$ observation matrix for a given mix connected network.

- The heart of this dissertation is Chapter 4 where the novel concepts of *compensating couple* and its enforcement model the *compensating automata* are formally defined. The key contribution of this chapter is the introduction of the CEEME framework including its theoretical foundation. What makes the proposed framework attractive is its ability to handle a wide variety of properties, irrespective of their functional or non-functional (e.g. security) nature through the newly defined $\mathscr{P}$-compensate property. In this body of work $\mathscr{P}$ was the Nondeducibility IFP.

- In Chapter 5, the self-obfuscation characteristics of mix-connected networks were analyzed. The key contribution of this chapter is showing how compensation can be used to force obfuscate observations in systems by combining multiple actions. Conceptually, this is a significant result as it naturally provides the basis for implementing the compensating couple as a < action-correction(s) >pair.

- Chapters 6 and 7 presented two confidentiality models for CPSs. In essence, these are two ways of instantiating the compensating automata. The particular property that was enforced was the Nondeducibility-compensate property for regular structures. They key contribution in 6 is the formulation of *System Behaviors* while in Chapter 7 its the *Replacement Solutions* concept.

- Chapter 8 assembled all the theoretical foundations introduced in this body of work into an architecture that can be used to provide event confidentiality. In Chapter 9 the proposed architecture was compared against an existing architecture which is used to mitigate cascading failures in a smart grid environment. Also provided are the algorithms to calculate replacement solutions and System Behaviors for a given, possibly IFP violating, (Nondeducibility security in this case), $\mathcal{D}_H$ action.

10.3. ACHIEVED RESEARCH GOALS

Within the scope and time frame of this dissertation, three goals were set as deliverables.

1. To develop Confidentiality Models for CPSs
2. To develop Information Flow architectures for CPSs
3. To develop EM enforceable event compensation for CPSs

10.3.1. Confidentiality Models. The first phase of developing a CPS confidentiality model was to express the physical flow of the system in terms of a system invariant that describes both the commodity and the information flows, which was done in Chapter 3. In doing so, a generalized set of theories that explain the relationship between connectivity and information flow violation was also presented.

Along with confidentiality leakage analysis, an implementation of EM enforcing event confidentiality required algorithms that could identify possible combinations of event compensations within a finite amount of time. Algorithm (1) fulfilled half of this objective by being able to identify possible candidate solutions for regular networks. Algorithms (2) and (3) fulfilled the other half by providing the ability to search the candidate space for suitable of compensating couples. This was generalized into an integrated flow model that incorporates algorithmic information flow with physical commodity flow with the CEEME framework in Chapter 8.

10.3.2. Information Flow Architectures for CPSs. Once confidentiality models were established, the next step was to develop an information flow security based generalized framework for cyber-physical systems. The compensating automata was proposed as the primary approach towards this goal and was later materialized as a runtime enforcement mechanism in Chapter 8. This was done by incorporating the current implementation of the max-flow based FACTS control [105] in enforcing the Nondeducibility–compensate property for the smart power grid.

10.3.3. EM Enforceable Event Compensation for CPSs. From an operational standpoint, the primary contribution of this work is on integrating cyber control with distributed decision making for CPSs, while maintaining the functional and information flow security requirements. By and large, cyber decisions of a CPS can be due to economical reasons (energy management), functional reasons (avoid cascading failures), control reasons (scheduling), or social reasons (response to a natural disaster). Thus, the proposed concept of event compensation must have the flexibility to address different needs of the system while being a part of the overall distributed cyber algorithm. As an example, Chapter 9 showed how the functional needs of a CPS can be still met while maintaining the event confidentiality.

10.4. FUTURE WORK

There are many exciting ways to extend the work presented in this dissertation. Some of the key areas of future research are enumerated as follows.

10.4.1. Theoretical Extensions. This is probably the most important and obvious extension. Current theoretical basis only covers regular networks. For a more thorough and comprehensive understanding of practical CPS, this necessarily has to be extended to cover irregular structures. There are two approaches. One is the use the same methodology as in this dissertation by considering simple building blocks to build complex interconnections. The other option is to use series-parallel reduction techniques to reduce irregular networks into regular structures.

By definition, the compensating automata is used to enforce $\mathscr{P}$-compensate property where $\mathscr{P}$ could be a multitude of other interesting system properties such as reliability, stability, or even availability through fault tolerance. It would be also interesting to see how the proposed event compensation based framework could be used to enforce some of these other notable properties.

Another theoretical aspect not handled under the current work is the possibility of multiple simultaneous $\mathcal{D}_H$ actions and their effect on enforceability. There is every possibility that the current single $\mathcal{D}_H$ action confidentiality models would hold well against multiple counterparts.

Currently, System Behaviors and Replacement Solutions are considered in isolation. It would be interesting to see how they function when considered in combinations. For example, questions such as how system behavior on replacement solutions different from replacement solutions on system behaviors, or in what situations can and should the combinations considered may are worth exploring. Chapter 9 provides certain amount of preliminary evidence of success for such combinations.

10.4.2. Operational Limitations. Current work is based on a dense deployment[18] of IQCs. A more practical approach would be to consider sparse deployment, meaning the number of IQCs available is less than the number of edges in the network, or there are edges without smart control capability. Such a limitation would naturally pave the way for partially nondeducible (or deducible) systems. It would be interesting to determine an optimal placement strategy that maximizes Nondeducibility under limited resources.

Another aspect would be optimal nondeducible solutions in system behavior and replacement solution schemes. Hypothetical concepts such as partial upstream or downstream replacement schemes can be used to strengthen the proposed CPS security model making it more practical.

10.4.3. Framework Improvement. The overhead associated with the compensator module can be reduced by improving the Selection and Feasibility modules in the CEEME architecture with grid intelligence. In other words, a pervasive

[18] see Chapter 8 for a description

approach to system engineering. Such improvements may be driven by economical, social, or resource availability reasons. In line with this extension is the runtime analysis of the proposed work in a real CPS setup.

10.5. CONCLUDING REMARKS

It is the Author's belief that the work in this dissertation will spearhead a healthy discussion and a fresh push towards developing security models that account for all aspects of CPSs. By all means, the work in this dissertation does not present a comprehensive CPS security model. Yet, it does serve as a first stepping stone towards a new CPS security paradigm that can be vastly improved with further research, analysis, and understanding of CPS dynamics.

ACRONYMS

BIBLIOGRAPHY

[1] J. Sztipanovits, "Composition of Cyber-Physical Systems," in *Procs of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pp. 3–6, March 2007.

[2] E. A. Lee, "Position Paper: Cyber-Physical Systems - Are Computing Foundations Adequate?," in *NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, October 2006.

[3] N. Adam, "Workshop on Future Directions on Cyber-Physical Systems Security," tech. rep., Department of Homeland Security, January 2010. Final Report.

[4] T. T. Gamage and B. M. McMillin, "Observing for Changes: Nondeducibility Based Analysis of Cyber-Physical Systems," in *Proceedings of the 3rd International Federation for Information Processing Conference (IFIP WG 11.10)*, (Hanover, NH), pp. 169–183, Springer Boston, April 2009.

[5] C. Neuman, "Challenges in Security for Cyber-Physical Systems." DHS Workshop on Future Directions in Cyber-Physical Systems Security, July 2009. http://cimic.rutgers.edu/positionPapers/CPS-Neuman.pdf.

[6] N. I. of Standards and Technology, "Introduction to NISTIR 7628 Guidelines for Smart Grid Cyber Security," tech. rep., National Institute of Standards and Technology, September 2010. http://nist.gov/smartgrid/upload/nistir-7628_total.pdf.

[7] C. P. Pfleeger and S. L. Pfleeger, *Security in Computing (4th Edition)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2006.

[8] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks Against Process Control Systems: Risk Assessment, Detection, and Response," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, (New York, NY, USA), pp. 355–366, ACM, 2011.

[9] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier." Symantec Security, November 2010. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[10] A. Huang, "Renewable Energy System Research and Education at the NSF FREEDM Systems Center," in *IEEE Power & Energy Society General Meeting (PES)*, pp. 1–6, July 2009.

[11] S. Massoud Amin and B. Wollenberg, "Toward a Smart Grid: Power Delivery for the 21st Century," *IEEE Power & and Energy Magazine*, vol. 3, pp. 34–41, Sep 2005.

[12] A. Z. Faza, S. Sedigh, and B. M. McMillin, "Reliability Analysis for the Advanced Electric Power Grid: From Cyber Control and Communication to Physical Manifestations of Failure," in *Proceedings of the 28th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, (Berlin, Heidelberg), pp. 257–269, Springer-Verlag, 2009.

[13] K. Zetter, "Security Pros Question Deployment of Smart Meters," March 2010. http://www.wired.com/threatlevel/2010/03/smart-grids-done-smartly/.

[14] C. Gunter and et. al., "Trustworthy Cyber-Infrastructure for Power," in *TAS Workshop on Research Directions for Security and Networking in Critical Real-Time and Embedded Systems (CRTES)*, (San Jose, CA, USA), April 2006.

[15] R. Akella and B. M. McMillin, "Information Flow Analysis of Energy Management in a Smart Grid," in *Proceedings of the 29th international conference on Computer safety, reliability, and security*, SAFECOMP'10, (Berlin, Heidelberg), pp. 263–276, Springer-Verlag, 2010.

[16] H. Tang and B. McMillin, "Analysis of the security of information flow in the Advanced Electric Power Grid using Flexible Alternating Current Transmission System (FACTS)," in *Critical Infrastructure Protection*, pp. 43–56, Springer, 2008.

[17] H. Tang and B. McMillin, "Security Property Violation in CPS through Timing," in *Proceedings of the 1st Workshop on Cyber-Physical Systems (part of ICDCS)*, IEEE Computer Society Press, 2008.

[18] D. Ditch and B. McMillin, "The Security Implication of Multiple Observers in a Distributed System," in *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 341–346, July 2009.

[19] V. Srinivasan, J. Stankovic, and K. Whitehouse, "Protecting Your Daily in-home Activity Information From a Wireless Snooping Attack," in *Proceedings of the 10th international conference on Ubiquitous computing (UbiComp)*, (New York, NY, USA), pp. 202–211, ACM, 2008.

[20] C. Davis, J. Tate, H. Okhravi, C. Grier, T. Overbye, and D. Nicol, "SCADA Cyber Security Testbed Development," in *38th North American Power Symposium (NAPS)*, pp. 483–488, September 2006.

[21] D.-J. Kang, J.-J. Lee, S.-J. Kim, and J.-H. Park, "Analysis on Cyber Threats to SCADA Systems," in *Transmission Distribution Conference Exposition: Asia and Pacific*, pp. 1–4, October 2009.

[22] J. Stamp, J. Dillinger, W. Young, and J. DePoy, "Common Vulnerabilities in Critical Infrastructure Control Systems," tech. rep., Sandia National Laboratories, 2003.

[23] A. McIntyre, B. Becker, and R. Halbgewachs, "Security Metrics for Process Control Systems," Tech. Rep. SAND2008-2070P, Sandia National Laboratories, 2007.

[24] C.-C. Liu, C.-W. Ten, and M. Govindarasu, "Cybersecurity of SCADA Systems: Vulnerability Assessment and Mitigation," in *IEEE/PES Power Systems Conference and Exposition (PSCE)*, pp. 1–3, March 2009.

[25] C.-W. Ten, C.-C. Liu, and M. Govindarasu, "Vulnerability Assessment of Cybersecurity for SCADA Systems Using Attack Trees," in *IEEE Power Engineering Society General Meeting*, pp. 1–8, June 2007.

[26] E. Johansson, T. Sommestad, and M. Ekstedt, "Issues of Cyber Security in SCADA-systems – On the Importance of Awareness," in *20th International Conference and Exhibition on Electricity Distribution*, pp. 1–4, June 2009.

[27] C. Zimmer and F. Mueller, "Time-Based Intrusion Detection in Cyber-Physical Systems," in *International Conference on Cyber-Physical Systems*, April 2010.

[28] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems," *Communications of the ACM*, vol. 19, no. 8, pp. 461–471, 1976.

[29] R. J. Lipton and L. Snyder, "A Linear Time Algorithm for Deciding Subject Security," *Journal of the ACM*, vol. 24, no. 3, pp. 455–464, 1977.

[30] D. E. Bell and L. J. Lapadula, "Secure Computer System: Unified Exposition and Multics Interpretation," Tech. Rep. ESD-TR-75-306, The MITRE Corporation, 1976.

[31] K. J. Biba, "Integrity Considerations for Secure Computer Systems," Tech. Rep. ESD-TR-76-372, MITRE Corporation, Bedford, MA, April 1977.

[32] A. Sabelfeld and A. Myers, "Language-Based Information-Flow Security," *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 5–19, January 2003.

[33] D. Bell and L. LaPadula, "Secure Computer Systems: Mathematical Foundations," Tech. Rep. MTR-2547, MITRE Corporation, March 1973.

[34] A. Myers and B. Liskov, "Protecting Privacy in a Decentralized Environment," in *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, vol. 1, pp. 266–277, 2000.

[35] J. Wittbold and D. Johnson, "Information Flow in Nondeterministic Systems," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 144–161, May 1990.

[36] J. McLean, "Security Models and Information Flow," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 180–187, May 1990.

[37] J. A. Goguen and J. Meseguer, "Unwinding and Inference Control," *IEEE Symposium on Security and Privacy*, vol. 0, p. 75, 1984.

[38] N. Nagatou and T. Watanabe, "Run-Time Detection of Covert Channels," in *Proceedings of the First International Conference on Availability, Reliability and Security (ARES)*, pp. 577–584, 2006.

[39] S. McCamant and M. D. Ernst, "Quantitative Information Flow As Network Flow Capacity," in *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation (PLDI)*, (New York, NY, USA), pp. 193–205, ACM, 2008.

[40] C. Bryce, J. Banatre, and D. Le Metayer, "An Approach to Information Security in Distributed Systems," in *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pp. 384–394, August 1995.

[41] J. Goguen and J. Meseguer, "Security Policies and Security Models," in *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, 1982.

[42] C. O'Halloran, "A Calculus of Information Flow," in *Proceedings of the European Symposium on Research in Computer Security*, (Toulouse, France), 1990.

[43] D. Sutherland, "A Model of Information," in *Proceeding of the 9th National Computer Security Conference*, (Baltimore, MD), pp. 175–183, September 1986.

[44] S. K. Nair, P. N. D. Simpson, B. Crispo, and A. S. Tanenbaum, "A Virtual Machine Based Information Flow Control System for Policy Enforcement," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 197, pp. 3–16, February 2008.

[45] K. W. Hamlen, G. Morrisett, and F. B. Schneider, "Computability Classes for Enforcement Mechanisms," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 28, no. 1, pp. 175–205, 2006.

[46] G. Lowe, "Quantifying Information Flow," *IEEE Computer Security Foundations Workshop*, vol. 0, p. 18, 2002.

[47] R. Focardi and R. Gorrieri, "Classification of Security Properties (Part I: Information Flow)," in *Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design (FOSAD)*, (London, UK), pp. 331–396, Springer-Verlag, 2001.

[48] S. Son, R. Mukkamala, and R. David, "Integrating Security and Real-Time Requirements Using Covert Channel Capacity," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, pp. 865–879, Nov/Dec 2000.

[49] B. Alpern and F. B. Schneider, "Defining Liveness," tech. rep., Cornell University, Ithaca, NY, USA, 1984.

[50] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Transactions on Software Engineering*, vol. SE-3, pp. 125–143, March 1977.

[51] F. B. Schneider, "Enforceable Security Policies," *ACM Transactions on Information and System Security TISSEC*, vol. 3, no. 1, pp. 30–50, 2000.

[52] J. McLean, "A General Theory of Composition for a Class of "Possibilistic" Properties," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 53–67, 1996.

[53] D. M. Volpano, "Safety versus Secrecy," in *Proceedings of the 6th International Symposium on Static Analysis (SAS)*, (London, UK), pp. 303–311, Springer-Verlag, 1999.

[54] J. A. Ligatti, *Policy Enforcement via Program Monitoring*. PhD thesis, Princeton University, Princeton, NJ, USA, 2006. Adviser-Walker, David P.

[55] H. Mantel, "Possibilistic Definitions of Security–An Assembly Kit," in *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW-13)*, pp. 185–199, 2000.

[56] H. Mantel, "Unwinding Possibilistic Security Properties," in *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS)*, (London, UK), pp. 238–254, Springer-Verlag, 2000.

[57] N. Delgado, A. Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," *IEEE Transactions on Software Engineering*, vol. 30, pp. 859–872, December 2004.

[58] N. Ravi, M. Gruteser, and L. Iftode, "Non-Inference: An Information Flow Control Model for Location-based Services," *Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services*, pp. 1–10, July 2006.

[59] H. Mantel, "The Framework of Selective Interleaving Functions and the Modular Assembly Kit," in *Proceedings of the ACM workshop on Formal methods in security engineering (FMSE)*, (New York, NY, USA), pp. 53–62, ACM, 2005.

[60] S. Zdancewic and A. C. Myers, "Secure Information Flow and CPS," in *Proceedings of the 10th European Symposium on Programming Languages and Systems (ESOP)*, (London, UK), pp. 46–61, Springer-Verlag, 2001.

[61] W. Li, R. Wu, and H. Huang, "Colored Petri Nets Based Modeling of Information Flow Security," in *Second International Workshop on Knowledge Discovery and Data Mining (WKDD)*, pp. 681–684, January 2009.

[62] V. Varadharajan, "Petri Net Based Modelling of Information Flow Security Requirements," in *Proceedings Computer Security Foundations Workshop III*, pp. 51–61, June 1990.

[63] V. Varadharajan, "Hook-Up Property for Information Flow Secure Nets," in *Computer Security Foundations Workshop IV, 1991. Proceedings*, pp. 154–175, June 1991.

[64] P. Ryan and S. Schneider, "Process Algebra and Non-Interference," in *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pp. 214–227, 1999.

[65] L. Beringer and M. Hofmann, "Secure Information Flow and Program Logics," in *20th IEEE Computer Security Foundations Symposium (CSF)*, pp. 233–248, July 2007.

[66] W. Hunt, R. Krug, S. Ray, and W. Young, "Mechanized Information Flow Analysis through Inductive Assertions," in *Formal Methods in Computer-Aided Design (FMCAD)*, pp. 1–4, November 2008.

[67] K. O'Neill, M. Clarkson, and S. Chong, "Information-Flow Security for Interactive Programs," in *19th IEEE Computer Security Foundations Workshop*, pp. 199–201, 2006.

[68] F. Pottier, "A Simple View of Type-Secure Information Flow in the π-Calculus," in *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pp. 320–330, 2002.

[69] F. Martinelli and I. Matteucci, "An Approach for the Specification, Verification and Synthesis of Secure Systems," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 168, pp. 29–43, February 2007.

[70] F. Martinell and I. Matteucci, "Through Modeling to Synthesis of Security Automata," *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 179, pp. 31–46, July 2007.

[71] W. Masri, A. Podgurski, and D. Leon, "Detecting and Debugging Insecure Information Flows," in *15th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 198–209, November 2004.

[72] A. Sen and V. Garg, "Formal Verification of Simulation Traces Using Computation Slicing," *IEEE Transactions on Computers*, vol. 56, pp. 511–527, April 2007.

[73] N. Mittal, A. Sen, and V. Garg, "Solving Computation Slicing Using Predicate Detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, pp. 1700–1713, December 2007.

[74] R. Meyden and C. Zhang, "Information Flow in Systems with Schedulers," in *21st IEEE Computer Security Foundations Symposium (CSF)*, pp. 301–312, June 2008.

[75] R. Focardi, R. Gorrieri, and F. Martinelli, "Real-Time Information Flow Analysis," *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 20–35, January 2003.

[76] G. Smith, "Probabilistic Noninterference through Weak Probabilistic Bisimulation," in *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pp. 3–13, June 2003.

[77] C. A. R. Hoare, *Communicating Sequential Processes*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1985.

[78] T. Amtoft and A. Banerjee, "A Logic for Information Flow Analysis with an Application to Forward Slicing of Simple Imperative Programs," *Science of Computer Programming*, vol. 64, no. 1, pp. 3–28, 2007. Special issue on the 11th Static Analysis Symposium (SAS).

[79] A. Zakinthinos and E. S. Lee, "A General Theory of Security Properties," in *Proceedings of the IEEE Symposium on Security and Privacy*, (Washington, DC, USA), p. 94, IEEE Computer Society, 1997.

[80] J. Ligatti, L. Bauer, and D. Walker, "Edit Automata: Enforcement Mechanisms for Run-time Security Policies," *International Journal of Information Security*, vol. 4, pp. 2–16, February 2005.

[81] L. Bauer, J. Ligatti, and D. Walker, "More Enforceable Security Policies," in *Proceedings of the Workshop on Foundations of Computer Security (FLoC)*, (Copenhagen, Denmark), pp. 95–104, DIKU Technical Report, July 2002.

[82] D. Beauquier, J. Cohen, and R. Lanotte, "Security Policies Enforcement Using Finite Edit Automata," *Electronic Notes in Theoretical Computer Science*, vol. 229, no. 3, pp. 19–35, 2009. Proceedings of the First Interaction and Concurrency Experiences Workshop (ICE).

[83] J. Ligatti, L. Bauer, and D. Walker, "Run-time Enforcement of Nonsafety Policies," *ACM Transactions on Information and System Security*, vol. 12, january 2009.

[84] M. Bishop, *Computer Security: Art and Science*, ch. 8, pp. 191–197. Addison Wesley, 2003.

[85] L. Bauer, J. Ligatti, and D. Walker, "More Enforceable Security Policies," Tech. Rep. TR-649-02, Princeton University, July 2002.

[86] P. W. L. Fong, "Access Control by Tracking Shallow Execution History," *IEEE Symposium on Security and Privacy*, vol. 0, p. 43, 2004.

[87] C. Talhi, N. Tawbi, and M. Debbabi, "Execution Monitoring Enforcement Under Memory-Limitation Constraints," *Information and Computation*, vol. 206, no. 2-4, pp. 158–184, 2008.

[88] J. McLean, "A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 79–93, May 1994.

[89] A. Kovacevic, *The Impact of the Russia-Ukraine Gas Crisis in South-eastern Europe*. Oxford Institute for Energy Studies, March 2009.

[90] A. Armbruster, M. Gosnell, B. McMillin, and M. L. Crow, "Power Transmission Control Using Distributed Max-Flow," in *29th Annual International Computer Software and Applications Conference (COMPSAC)*, vol. 2, (Edinburgh, Scotland), pp. 256–263, IEEE Computer Society, July 2005.

[91] B. Schneier, "Stuxnet." Schneider on Security, October 2010. `http://www.schneier.com/blog/archives/2010/10/stuxnet.html`.

[92] M. Crow, *Computational Methods for Electric Power Systems*. CRC Press, 1 ed., December 2002.

[93] T. T. Gamage, T. P. Roth, and B. M. McMillin, "Confidentiality Preserving Security Properties for Cyber-Physical Systems," in *IEEE 35th Annual Computer Software and Applications Conference (COMPSAC '11)*, IEEE Computer Society, July 2011.

[94] J. Ligatti and S. Reddy, "A Theory of Runtime Enforcement, with Results," in *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, September 2010.

[95] A. W. Roscoe, J. C. P. Woodcock, and L. Wulf, "Non-Interference through Determinism," in *Proceedings of the Third European Symposium on Research in Computer Security (ESORICS)*, (London, UK), pp. 33–53, Springer-Verlag, 1994.

[96] T. T. Gamage, B. M. McMillin, and T. P. Roth, "Enforcing Information Flow Security Properties in Cyber-Physical Systems: A Generalized Framework Based on Compensation," in *IEEE 34th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, pp. 158–163, July 2010.

[97] T. T. Gamage and B. M. McMillin, "EM Enforcing Information Flow Properties using Compensating Events," in *Proceedings of the 42nd Hawaii International Conference on Systems Science (HICSS-42)*, pp. 1–7, IEEE Computer Society, January 2009.

[98] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, ch. 26, pp. 748–766. The MIT Press, 3 ed., 2009.

[99] "Final Report on the August 14th Blackout in the United States and Canada." Power System Outage Task Force U.S. and Canada, 2004. `https://reports.energy.gov/BlackoutFinal-Web.pdf`.

[100] J. Valdes, R. E. Tarjan, and E. L. Lawler, "The Recognition of Series Parallel Digraphs," in *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, STOC '79, (New York, NY, USA), pp. 1–12, ACM, 1979.

[101] L. A. Schoenmakers, "A New Algorithm for the Recognition of Series Parallel Graphs," tech. rep., Amsterdam, The Netherlands, The Netherlands, 1995.

[102] M. Mitchell, "Creating Minimal Vertex Series Parallel Graphs from Directed Acyclic Graphs," in *Proceedings of the 2004 Australasian Symposium on Information Visualization*, vol. 35 of *APVis '04*, (Darlinghurst, Australia, Australia), pp. 133–139, Australian Computer Society, Inc., 2004.

[103] B. Chowdhury and S. Baravc, "Creating Cascading Failure Scenarios in Interconnected Power Systems," in *Power Engineering Society General Meeting, 2006. IEEE*, pp. 18–22, 2006.

[104] A. Lininger, B. McMillin, M. Crow, and B. Chowdhury, "Use of Max-Flow on FACTS Devices," in *39th North American Power Symposium, (NAPS)*, pp. 288–294, October 2007.

[105] J. Chaloupek, D. R. Tauritz, B. M. McMillin, and M. L. Crow, "Evolutionary Optimization of Flexible AC Transmission System Device Placement for Increasing Power Grid Reliability," in *Proceedings of the 6th International Workshop on Frontiers in Evolutionary Algorithms (FEA)*, (Salt Lake City, Utah), pp. 516–519, July 2005.

VITA

Thoshitha Thanushka Gamage was born in the city of Badulla, Sri Lanka. He attended Passara Central College for his grade 1 – 11 education, and completed the General Certificate of Education (GCE) Ordinary Level (O/L) exam with all high distinctions in 1997. During 1998 to 2000, Thoshitha attended the Dharmaraja College, Kandy, Sri Lanka for his grade 12 – 13 education specialized in Mathematics. He completed the GCE Advanced Level (A/L) exam in 2000 that earned him the opportunity to attend the Engineering School at the University of Peradeniya, Sri Lanka starting from October 2001. Thoshitha completed his undergraduate degree in Computer Engineering in January 2006 with a 2^{nd} class upper division honors distinction.

In Fall 2006, he enrolled in the Master's program in the department of Computer Science at the St. Cloud State University in St. Cloud, Minnesota. Thoshitha graduated with his Master's degree in Spring 2008 and joined the Ph.D. program in Computer Science at the Missouri University of Science and Technology in Rolla, Missouri. His graduate research advisor is Dr. Bruce McMillin. Thoshitha's primary research interest is in the area of Computer Security and its intersection with Formal Methods, Distributed Systems, and Cross-Disciplinary Architectures. He successfully defended his Ph.D. dissertation entitled "CEEME: Compensating Events based Execution Monitoring Enforcement for Cyber-Physical Systems" on October 28, 2011.