Qualitative Fault Modeling in Safety Critical Cyber Physical Systems

Ajay Chhokra, Nagabhushan Mahadevan, Abhishek Dubey, Gabor Karsai
{ajay.d.chhokra,nag.mahadevan,abhishek.dubey,gabor.karsai}@vanderbilt.edu
Institute of Software Integrated Systems, Vanderbilt University
Nashville, TN, USA

ABSTRACT

One of the key requirements for designing safety critical cyber physical systems (CPS) is to ensure resiliency. Typically, the cyber sub-system in a CPS is empowered with protection devices that quickly detect and isolate faulty components to avoid failures. However, these protection devices can have internal faults that can cause cascading failures, leading to system collapse. Thus, to guarantee the resiliency of the system, it is necessary to identify the root cause(s) of a given system disturbance to take appropriate control actions. Correct failure diagnosis in such systems depends upon an integrated fault model of the system that captures the effect of faults in CPS as well as nominal and faulty operation of protection devices, sensors, and actuators.

In this paper, we propose a novel graph based qualitative fault modeling formalism for CPS, called, Temporal Causal Diagrams (TCDs) that allow system designers to effectively represent faults and their effects in both physical and cyber sub-systems. The paper also discusses in detail the fault propagation and execution semantics of a TCD model by translating to timed automata and thus allowing an efficient means to quickly analyze, validate and verify the fault model. In the end, we show the efficacy of the modeling approach with the help of a case study from energy system.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; • **Theory of computation** → *Formalisms*; *Verification by model checking*.

KEYWORDS

Cyber Physical Systems, Fault Modeling, Energy System, Timed Automata

ACM Reference Format:
Ajay Chhokra, Nagabhushan Mahadevan, Abhishek Dubey, Gabor Karsai. 2020. Qualitative Fault Modeling in Safety Critical Cyber Physical Systems. In *12th System Analysis and Modelling Conference (SAM '20), October 19–20, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3419804.3420273

1 INTRODUCTION

The last decade has seen a significant research activity in the development of Cyber Physical Systems (CPS) as a result of advancement in communication technologies and embedded systems [18]. A CPS consists of 1) a *physical* or engineered system containing mechanical, chemical, or biological processes (realized without digital computers), and 2) a *cyber* system comprising of a network of computing devices that manage the physical system with the help of sensors and actuators. One of the key requirements for designing safety critical CPS such as Cyber Physical Energy Systems (CPES) is to ensure resiliency. To achieve this requirement, the cyber sub-system in a safety critical CPS is empowered with dedicated protection devices that detect anomalies in physical plant and mitigate their effects by taking appropriate remedial actions. Typically, the protection devices are arranged redundantly to allow multiple devices to detect and mitigate fault in a section of the system. However, the protection devices along with sensors and actuators can have internal faults that can alter the anticipated fault trajectory, leading to increased instability in the system resulting in cascading failures. According to *North American Electric Reliability Corporation*, nearly all major system failures in the past decades including recent blackouts, excluding those caused by severe weather, include misoperations of protection system as a factor contributing to the propagation of the events [22]. Thus, a thorough analysis of fault effect propagation including the behavioral verification of protection devices in the cyber system becomes imperative to assure resiliency in any safety critical CPS.

In this paper, we propose a novel graph based qualitative fault modeling formalism, Temporal Causal Diagrams (TCDs), that allows system designers to easily model the effects of fault propagation in the physical system and at the same time analyze and verify the response of protection devices to such faults while considering both nominal and faulty behaviors. Specifically, we make the following technical contributions, ① Discuss various elements of TCD modeling language with the help of a simple example. ② Describe the fault propagation and execution semantics of a TCD model by translating it to a network of Timed Automata [5]. ③ Show the efficacy of the approach with the help of a case study from CPES.

The rest of the paper is organized as follows: Section 2 discusses the existing CPS fault modeling techniques. Section 3 describes the modeling formalism with the help of an exemplar. The same section also provides a detailed description of fault propagation and execution semantics of a TCD model with the help of timed automata followed by translation rules to transform a TCD model to a corresponding timed automata. Section 4 presents the case study that describes a TCD model for capturing the interactions between the progression of faults in CPES (transmission lines) and the behavior of protection devices (distance relays) along with simulation and verification results. Finally, section 5 discusses the concluding remarks and future work.

2 RELATED RESEARCH

Typically, the analysis of a CPS is performed using *Hybrid Automata* [4]. There exist several tools such as *Ptolemy* [8], Stateflow [16], *SpaceEx* [12], *PHaVer* [11], etc that are widely used to model, simulate and verify hybrid systems. However, simulating and verifying hybrid automata is a computationally expensive process that can take a large amount of time. Moreover, in safety critical CPS such as the energy system, describing the change in continuous variables in different modes of the automaton as a result of fault and protection system operation is very difficult. The quantitative modeling approaches, such as hybrid automata, in general, suffer from poor scalability of validation and verification times with the increase of continuous variables due to expensive solvers and huge state space involving real valued state variables respectively.

Another widely used approach for fault analysis in dynamic systems is based on qualitative modeling techniques such as *Signed Directed Graphs* [20], *Temporal Causal Graphs* [7], *Fault Trees* [17], *Temporal Fault Propagation Graphs* [1]. A qualitative fault model is a surrogate model that explicitly represents the effect of faults on the measurements associated with state variables. The relation between fault and its effect on measurements is expressed using qualitative functions instead of the mathematical function used in analytical models. Qualitative models significantly reduce the state space and are easier to analyze. However, these techniques are not suited for modeling the behavior of the discrete protection system as they lack constructs to represent discrete states and transition functions. There are well studied approaches based on *Automata theory* [21], *Discrete Event System Specification* [19], and *Petri-nets* [13] that model the behavior of the protection system as a discrete event based system.

A better approach for analyzing fault propagation in safety critical CPS should be a combination of the qualitative fault modeling methodology to capture the dynamics of fault propagation in the physical system and discrete event based systems to represent the response of the protection system including sensors and actuators in the absence or presence of cyber faults.

3 APPROACH

Our approach for modeling faults and their propagation is based on Temporal Fault Propagation Graphs (TFPG) [2] that captures the causality and temporal characteristics of fault effect propagation in dynamic systems. The classical TFPG model is a qualitative discrete-event model that captures the causal and temporal relationships between faults (causes) and discrepancies (effects) in a system, thereby modeling the fault cascades while taking into account propagation constraints imposed by operating modes and timing delays.

The TFPG model is generic and has been applied to represent faults and their propagation in various physical domains [3]. However, it is incapable of modeling nominal and faulty behavior of discrete systems such as fault-protection devices. These fault protection components are designed to mask the effect of physical faults by isolating the faulty component from the system. Additionally, the fault protection components can introduce their own failure modes such as missed and spurious detection faults that can alter the trajectory of fault effect propagation in the system.

We propose a new graphical formalism, *Temporal Causal Diagram* (TCD), that is based on a discrete-event abstraction of the system which can model the interplay of protection system behavior and the physical fault trajectories. It contains TFPG as an embedded sub-model along with one or more *Time Triggered Automata* (TTA) [15] to represent the faulty and nominal behavior of protection system components. TCD formalism is a super-set of TFPG language as it contains an extended set of nodes and edges to model the behavior of protection system components.

3.1 Modeling Formalism

A TCD model is a behavior augmented fault propagation graph where behavior of protection system components is explicitly modeled using TTA. Formally, a TCD model, $\mathcal{G}$, can be defined as a tuple,

$$\mathcal{G} = (F,\ D,\ E,\ M,\ ET,\ EM,\ ND,\ Q,\ Q_0,\ \Sigma,\ \Psi_{act},\ \Psi_{ina},\ \Phi,\ \Omega,\ T)$$

where

- ✠ F is a nonempty set of faults in the system. F is partitioned into two disjoint sets, F_{phy} and F_{cyb}, where the first set represents the faults in the physical components and the later shows faults associated with cyber or protection system.
- ✠ D is a nonempty set of observable discrepancies. It is a combination of two disjoint sets, D_{phy} and D_{cyb}, where the set D_{phy} represents fault effects related to F_{phy} and D_{cyb} are discrepancies related to cyber faults.
- ✠ $E \subseteq V \times V$ is the set of directional fault propagation edges connecting two nodes, where $V \subseteq F \cup D$ such that ① there are no self loops, ② a physical node, $v_i \in F_{phy} \cup D_{phy}$, is not connected to cyber node, $v_j \in F_{cyb} \cup D_{cyb}$ and ③ fault nodes cannot be destination node.
- ✠ M is a nonempty set of system modes. At each time instant, t, the system can be in only one mode.
- ✠ $ET: \ E \rightarrow \mathbb{R}^2_{\geq 0}$ is a map that associates every edge, $e \in E$ a time interval $[t_{min}, t_{max}] \in \mathbb{R}^2_{\geq 0}$, such that $t_{max} \geq t_{min}$, where t_{min} and t_{max} are the minimum and maximum time for fault propagation to occur over the edge.
- ✠ $EM : E \rightarrow \{M^q \cup \emptyset\}$ is a map that associates every edge $e \in E$ with a set of modes, when the edge is active. For mode independent edges i.e. active in all system modes, $EM(e) = \emptyset$.
- ✠ $ND: \ E \rightarrow \{\top, \bot\}$ is a map that associates an edge, $e \in E$ to $\top$(True) or $\bot$(False), where $\bot$ implies the propagation along the edge, e **will** happen, whereas $\top$ implies the propagation is uncertain and **can** happen.
- ✠ Σ is a finite set of event labels. We categorize events into two types, observable (Σ_{obs}) and unobservable (Σ_{unobs}). Observable events include alarms related to observable discrepancies, commands or messages exchanged between cyber components etc. Whereas, unobservable events are related to injection of faults in the system.

Table 1: Example TCD Model

TCD Element	Example Model
Faults (F)	$\{F1, Fmiss1, Fmiss2, Fstuck\}$
Discrepancies (D)	$\{D1, D2\}$
Edges (E)	$\{(F1, D1),\ (D1, D2)\}$
System modes (M)	$\{m1, m2\}$
Fault propagation duration (ET)	$ET(F1, D1) = [2, 5], ET(D1, D2) = [1, 3]$
Edge-Mode Map (EM)	$EM(F1, D1) = m1,\ EM(D1, D2) = m1$
Edge Uncertainty (ND)	$ND(F1,\ D1) = \bot, ND(D1,\ D2) = \top$
Events (Σ)	$\{d1, d2, d1', d2', fstuck, fmiss1,$ $fmiss2, fm1, c, sc, reset\}$
Automaton locations (Q)	$\{S1, S2, S3, S4, S5, S6, S7, S8, S9, S10\}$
Intial Locations (Q_0)	$\{S1, S5, S7\}$
Location-Mode map (Ω)	$\Omega(m1) : (S1 \lor S3), \Omega(m2) : (S2 \lor S4)$
Activation event map (Ψ)	$\Psi(D1) : d1, \Psi(D2) : d2,$ $\Psi(F1) : f1, \Psi(Fmiss^1) : fmiss1,$ $\Psi(Fmiss2) : fmiss2,$ $\Psi(Fstuck) : f_{stuck}$
De-activation event map ($\overline{\Psi}$)	$\overline{\Psi}(D1) : d1', \overline{\Psi}(D2) : d2'$
Timing constraints (Φ)	$\{(r)\}$
Transitions (T)	Illustrated in Figure 1

- Q is a finite set of locations associated with time triggered automata, and Q_0, is the initial location set.
- $\Omega :\ M \rightarrow f(Q^n)$ is a map that relates a system mode $m \in M$ with a boolean function defined over locations $q \in Q$. A boolean function, $f\ :\ Q^n\ \rightarrow \{\top,\ \bot\}$ can be viewed as a constraint on the locations of cyber sub-system automata. At any time t_1, a system mode m represents the actual operating conditions if the corresponding boolean constraint is satisfied by the locations of cyber sub-system automata, i.e. $\Omega(m)\big|_{t=t_1} == \top$.
- $\Psi_{act}\ :\ \ F \cup D\ \rightarrow\ \Sigma$ is a map that relates activation of nodes, $v \in F \cup D$, in TFPG sub-model with events labels, $\sigma \in \Sigma$, in TTA.
- $\Psi_{ina}\ :\ \ F \cup D \rightarrow \Sigma$ is a map that relates de-activation of nodes, $v \in D$, in TFPG sub-model with events labels, $\sigma \in \Sigma$, in TTA[1].
- Φ is a set of timing constraints, $\Phi = [n], (n)|n \in \mathbb{N}_{+}$, where $[n]$ denotes instantaneous constraints and (n) represents periodic constraints. The timing constraints specify a pattern of time points at which the automaton checks for the presence of events.
- $T \subset Q \times \Sigma \times \Phi \times \Sigma^n \times Q$ is a finite set of transitions between any two locations. Each transition of the time triggered automaton is labeled with an event request, a timing constraint and output event(s). For example the tuple, ($q_1,\ \sigma_1,\ [n],\ \sigma_2,\ q_2$) represents a transition from location $q_1 \in Q$ to $q_2 \in Q$ where $\sigma_1,\ \sigma_2 \in \Sigma$ are input and output events respectively and $[n]$ is a instantaneous time constraint. The transition is enabled only iff the event σ_1 is valid [see Definition 1 in §6] at time, $t\ =\ t_1 + n$, where t_1 is the time when automaton entered location q_1. A transition is represented syntactically, as an edge between two locations with a label of the form `Event(time constraint)/Event` or `Event[time constraint]/Event`.

The TCD model of an arbitrary CPS is described in Table 1. The example system consists of two protection devices (PD1, PD2) that detect anomalous behavior in the underlying physical processes and an actuator (ACT) to mitigate or isolate the fault effects. The TCD model consists of four fault nodes, $F = \{F1, Fmiss1, Fmiss2, Fstuck\}$, where $F1$ is a physical fault and $Fmiss1, Fmiss2,\ Fstuck$ are cyber faults related to PD1, PD2 and ACT respectively. The fault, $F1$ leads to an aberrant behavior indicated by the observable discrepancies, $D = \{D1, D2\}$. The fault effect from $F1$ manifests as $D1$ after the activation of $F1$, indicated by the event $f1$ in system mode $m1$. The manifestation of $D1$ is signaled by the event $d1$ that succeeds $f1$ by a duration [2,5] as highlighted by the markers associated with the edge between $F1$ and $D1$ in Figure 1(a). The system mode $m1$ implies the location of the actuator, ACT to be either $S1$ or $S3$. Under the same system mode, the fault effect propagates from $D1$ to $D2$ in [1,3] units of time, producing an event $d2$.

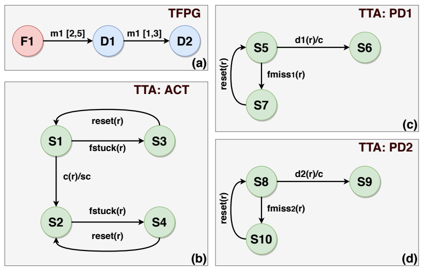

Figure 1: Example TCD Model

The TTA associated with PD1 consists of three locations with S5 being the initial location. The automaton models both nominal and faulty operation of the protection device. A missed detection fault, $Fmiss1$ affects the operation, by forcing the automaton to skip the detection of anomalous behavior indicated by the event $d1$. While in S5, the automaton checks for the presence of events $d1$ and $fmiss1$ every r units of time. The periodic checking of events is enforced by the timing constraint associated with all outgoing transitions from S5. If $fmiss1$ is present then the automaton transitions to S7. On the other hand, the presence of the event $d1$ causes the automaton to jump to S6 and generates an actuation command, indicated by the event c. Similarly, in S7, the automaton checks for the presence of the *reset* event that takes the automaton back to the initial location as highlighted in Figure 1(c).

The TTA, ACT, consists of four locations, with S1 being the initial location. The automaton models the operation of an abstract actuator that changes the state of the physical process after receiving commands from protection devices. The change in actuator location, signaled by the event, sc leads to change in the system mode affecting the fault propagation. The automaton also captures the behavior of the actuator under the influence of the stuck fault, $Fstuck$ that forces an actuator to ignore commands from the protection devices. While in S1, the automaton periodically checks the presence of events, c or $fstuck$. The event, $fstuck$, indicates the presence of the stuck fault. The presence of c forces the actuator to move to S2 and generate sc state change event. On the other hand, the automaton transitions to S3 from S1 or to S4 from S2 if $fstuck$ is observed as shown in Figure 1(b).

3.2 Fault Propagation Semantics

In this section, we describe how fault effect propagates in a TCD model $\mathcal{G}$, while adhering to modal and temporal constraints. To define these constraints, we use two maps $\mathcal{N}\ :\ I \times F \cup D\ \rightarrow \{ON, OFF\} \times \mathbb{R}_{+}$, and $\mathcal{E}\ :\ I \times E\ \rightarrow\ \{ON, OFF\} \times \mathbb{R}_{+}$ to store the state of node and edges, where I is the indexed set of all timestamped events [see Definition 1 in §6], $F \in \mathcal{G}$ is the set of fault nodes, $D \in \mathcal{G}$ is the set of discrepancies, $E \in \mathcal{G}$ is the set of fault propagation edges. The state of a node is deemed ON, if the fault effect has reached the node, otherwise, remains OFF. Similarly, the

[1]Since the mappings, Ψ_{act} and Ψ_{ina} are bijective in nature, we use Ψ^{-1}_{act} and Ψ^{-1}_{ina} to map events to nodes.

state of an edge, $e \in E$ is considered ON if $m \in EM(e)$, where m is the current system mode and EM is the edge mode map. These maps also track the time at which the state of each node and edge was changed. For improving the readability, we refer to the state and time attributes associated with a node, n after k^{th} event as $\mathcal{N}_k(n)$.State and $\mathcal{N}_k(n)$.time respectively. Similarly for an edge e, $\mathcal{E}_k(e)$.State and $\mathcal{E}_k(e)$.time denote state of edge after event k and the time it was last changed.

Every fault effect propagation trace in a TCD model starts with a node activation event, k such that $\Psi_{act}^{-1}(\mathcal{L}(k)) \in F$, where $\mathcal{L}$ is the label associated with the time-stamped event $k \in I$ [see Definition 1 in §6]. If the associated fault node is cyber in nature, i.e., $\Psi_{act}^{-1}(\mathcal{L}(k)) \in F_{cyb}$ then it can cause transition in one or more protection system automaton. A transition in a protection device can result into generation of synchronizing events that can cause state transitions in other protection devices. These synchronizing events can also act as actuation commands that change the location of one or more actuators. An event related to an actuation command, $l \in I$ can alter the current system mode from m_i to m_j such that $\Omega(m_i) \rightarrow \perp\big|_{t=\mathcal{T}(l)} \wedge \Omega(m_j) \rightarrow \top\big|_{t=\mathcal{T}(l)}$. The change in mode can enable or disable an edge $e \in E$ after l^{th} event as per Equations 1, 2 respectively.

$$\mathcal{E}_l(e) \leftarrow (ON, \mathcal{T}(l)) \mid s.t.\ \mathcal{E}_{l-1}(e) == OFF \wedge m_l \in EM(e) \tag{1}$$

$$\mathcal{E}_l(e) \leftarrow (OFF, \mathcal{T}(l)) \mid s.t.\ \mathcal{E}_{l-1}(e) == ON \wedge m_l \notin EM(e) \tag{2}$$

On the other hand, if the initiating event is related to a physical fault node i.e. $\Psi_{act}^{-1}(\mathcal{L}(k)) \in F_{phy}$ then discrepancy node activation events can take place. The fault effect can propagate from fault node, $n = \Psi_{act}^{-1}(\mathcal{L}(k))$ to a discrepancy node $d \in D$ if the constraint specified in Equation 3 holds true

$$\mathcal{N}_k(n).State == ON \wedge \mathcal{N}_k(d) == OFF\ \wedge (n,d) \in E \wedge \mathcal{E}_k((n,d)).State == ON \tag{3}$$

Equation 3 ensures the fault effect will propagate to destination node only if the edge between the source and destination nodes is active and the state of the destination node is inactive. A discrepancy activation event p will[2] happen in the interval defined in Equation 4

$$\begin{aligned}\mathcal{T}(p) \leftarrow [ET(n,d).t_{min}\,,\ ET(n,d).t_{max}] +\\ \max(\mathcal{E}_k(n,d).time\,,\ \mathcal{N}_k(n).time)\end{aligned} \tag{4}$$

The activation of discrepancy node can lead to further activation of other discrepancy nodes as per the constraints mentioned in Equations 3 and 4. It can also cause transitions in one or more protection device automata that can generate new events leading to more transitions in protection devices and actuators. As stated previously, actuator location change can alter the system mode, which can disable existing active edges or enable new edges according to Equations 1 and 2. Finally, the change in the state of edges can alter the future activation of discrepancy nodes as per Equation 4.

3.3 Execution Semantics

In this section, we describe the execution semantics of a TCD model using a discrete model of time, i.e., the time advances in discrete steps. To define the execution rules, we translate TFPG sub model and all TTA to a common model of computation, Timed Automata [5], such that a TCD model, $\mathcal{G}$ is transformed into a network

[2]If $ND(n,d)$ is $\perp$ then p is guaranteed to be observed within the duration mentioned in Equation 4 otherwise its uncertain

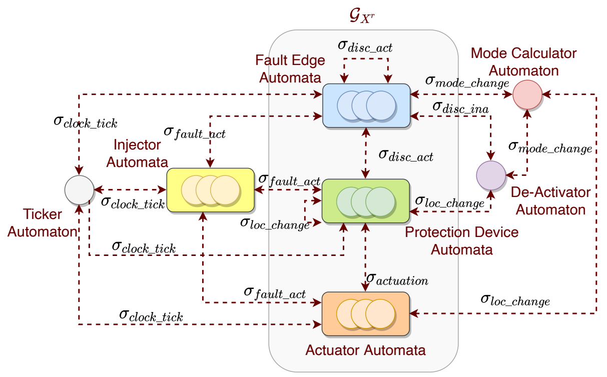

Figure 2: Structure of translated TCD model and its interfacings

of timed automata, $\mathcal{G}_{X^r}$, where each automaton is synchronized to an external clock source, *Ticker*.

A network of timed automata is the parallel composition, $A_1 \mid A_2 ... \mid A_n$ of a set of timed automata, A_1, A_2, ..., A_n. Communication between the individual automaton occurs in two ways via ① handshake synchronization using actions and ② shared variables. To model handshake synchronization between automata, an event $\sigma \in \Sigma$ associated with a transition is replaced by an action pair $(\sigma?, \sigma!)$ where $\sigma?$ implies event generation action and $\sigma!$ denotes consumption action. The synchronization is achieved by forcing generation and consumption actions to occur simultaneously i.e transitions in different automata with action labeled $\sigma!$ and $\sigma?$ are taken simultaneously. The second method uses two functions, *register_occurrence(event_id)* and *check_occurrence(event_id)*, where the former explicitly stores the presence of an event while the later checks for the presence of an event with a unique identifier, *event_id* [see Definition 1 in §6]. We classify handshake synchronization as *strict* since it happens instantaneously while communication via shared variables as *loose* as the update can be relayed in the next cycle depending upon the order of execution.

Apart from clock source, transformed TCD model, $\mathcal{G}_{X^r}$ requires *Injector* automaton to inject external events such as fault activation events, *Mode Calculator* automaton to implement Ω and finally node *De-activator* automaton to signal the discrepancies that should no longer be active in the current system mode. Figure 2 shows the structure of the translated TCD model, $\mathcal{G}_{X^r}$ and its interfacing with external automata. The external clock source uses strict communication mode to synchronize time with $\mathcal{G}_{X^r}$ and injector automata. A fault edge automaton uses handshake synchronization to convey activation of discrepancies to other fault edge automata but relies on shared variable to relay the same update to protection system. Similarly, protection system automata (protection device and actuator) uses loose communication mode to send and receive updates among each other but synchronizes with De-activator and Mode Calculator through handshake actions. The following sub-sections describe the UPPAAL [6] timed automata templates for a fault edge, clock source, mode calculator and de-activation automata along with translation procedure for converting an arbitrary TTA model to UPPAAL TA. An UPPAAL TA is an extended implementation of TA model of computation that allows transition guards to be defined on discrete variables as well as updating state variables when a transition is taken from one location to other. The UPPAAL transition tr from location q_i to q_j is a tuple, $<q_i, g, \sigma_k!$ or $\sigma_k?, r, q_j>$ where g is a boolean constraint defined over discrete variables and (or) clock variables, $\sigma_k!$ or $\sigma_k?$ is the synchronization action (generation or consumption) and r is the set of assignment statements over discrete and (or) clock variables.

3.3.1 Fault Edge Automaton. : The TFPG sub-model is a specification that put modal and timing constraints on the propagation of fault effect. To actually simulate or verify the fault effect propagation, we represent each edge as an UPPAAL TA. Figure 3(a) shows the timed automaton template of an arbitrary fault propagation edge, $e \in E$ with $EM(e)$ be the set of system modes in which the edge is active and $(t\_max, t\_min) = ET(e)$ is the duration of upper and lower bound on the propagation time. The corresponding node activation and de-activation events for source and destination nodes are (src_act, src_ina) and (dst_act, dst_ina) respectively. Other template arguments include a unique edge identifier, $edge\_id$, boolean parameter ND to capture uncertainty associated with the edge and lastly, an event identifier associated with activation of destination node, dst_id.

The fault edge automaton consists of 11 locations with `S3` being the initial location, based on the assumption that the initial system mode belongs to the set $EM(e)$, i.e., the edge is active and state of all TFPG nodes is inactive[3]. The automaton can transition to locations `S7` or `S4` or `S1` depending upon the event observed as shown in Figure 3(a). The transition, `S3` ⟶ `S7` is taken if the destination node becomes active i.e. the event dst_act is observed, whereas if source node becomes active, the automaton moves to `S4`. The location `S1` is selected if the edge becomes inactive as a result of mode change event, $mode\_change$. Whenever, the current system mode is changed, `Mode Calculator` generates a $mode\_change$ event and every fault edge automaton responds to the event by calling a function $check\_mode(edge\_id)$ to ascertain if the edge remains active in the new system mode. The function return true if the current mode is listed in $EM(e)$ otherwise false.

Similarly rest of the locations in the automaton, except `S9` and `S10` reacts to these events and transition to different locations as highlighted in Figure 3(a). Table 2 summarises the physical meaning of each location based on four conditions, ① Is destination node active?, ② Is edge active?, ③ Is source node active? and ④ Has the edge fired? We have assumed persistent faults in this study, which implies a fault edge can fire only once. As a result of this assumption, the locations `S9` and `S10` have no outgoing transitions. According to Table 2, both locations represent the edge has fired. However, `S9` denotes the edge has fired while signalling the activation of destination node, whereas `S10` does not.

While in `S4`(`S8`), the automaton counts the number of ticks, $global\_clock\_tick$ received from external clock source, $Ticker$ using a bounded local integer variable, $tick\_counter$ as shown in Figure 3(a). While the value of $tick\_counter$ is less than t_min, the automaton takes the self transition and increments the counter at every tick. However, after $tick\_counter$ becomes equal to t_min then the transition `S4` ⟶ `S4Temp`(`S8` ⟶ `S10`) also becomes enabled and automaton randomly decides whether to take the self transition or move to `S4Temp`(`S10`) at every clock tick. The self transition is

[3]If the assumption is not valid for system, then an appropriate location can be selected based on Table 2

Table 2: Fault edge automaton location interpretation

Location	Edge Fired?	Dest. Active?	Edge Active?	Source Active?
S1	False	False	False	False
S2	False	False	False	True
S3	False	False	True	False
S4	False	False	True	True
S5	False	True	False	False
S6	False	True	False	True
S7	False	True	True	False
S8	False	True	True	True
S9	True	X	X	X
S10	True	X	X	X

feasible till the location invariant, $tick\_counter < t\_max$, associated with `S4`(`S8`) is valid, i.e., $tick\_counter = t\_max - 1$. At the next tick, the automaton has to take the transition to `S4Temp`(`S10`). The location `S4Temp` is an intermediate committed location, that is used to check certainty parameter before generating dst_act event i.e either transition to `S9` or `S10`.

3.3.2 TTA to TA Translation. : The central idea of translating a TTA to an UPPAAL TA is to add a set of intermediate locations to allow the automaton to check the enabling condition of an outgoing transition, at discrete time steps adhering to the timing constraints associated with that transition. Algorithm 1 outlines sequence of steps required to translate a location s in TTA model to its equivalent UPPAAL TA representation. The algorithm expects the label of the location, s along with a set of all outgoing transitions T, from s. The output of the algorithm is a tuple, where the first element is a set of locations, S that contains the original location label, s and $p+1$ intermediate locations where p is the number of unique timing constraints associated with transitions in T. We use `tr.destination`, `tr.constraint`, `tr.ip_event`, `tr.op_event` to refer to destination node label, timing constraint, input synchronization event label and output synchronization event label associated with a TTA transition, $tr \in T$. Similarly, `k.type` and `k.value` is used to allude to type and value attributes of a timing constraint, k. We also define a function `Transition()` to create an UPPAAL transition that takes src location, guard condition, synchronization action, update statements and destination location as input arguments.

During the initialization, the algorithm adds s to S and creates p committed locations using function `Location()`. These committed locations are stored in a map, Loc that relates a timing constraint to the location label. Apart from Loc, two more maps are created, Var to map timing constraints to tick counters (bounded integer variables) and $Checked\_once$ to relate instantaneous timing constraints to boolean variables [Line 1]. After initializing, the algorithm creates an urgent location, $temp$ and add two transitions between s and $temp$[Lines 3-5]. The transition, $s \longrightarrow temp$ is taken after receiving clock synchronization event and all the tick counters are incremented to capture the advancement of time. The other transition, $temp \longrightarrow s$ has a guard condition which evaluates to true if none of the timing constraints, associated with all outgoing transitions from s, are satisfied. A pair of transitions are added between $temp$ and every location in Loc [Lines 8-16] such that the transition from $temp \longrightarrow Loc[k]$ implies the timing constraint k has satisfied whereas the transition in reverse direction captures the un-satifiability of transition $tr \in T$ with constraint k . Finally, $|T|$ transitions are added from $Loc[tr]$ to $tr.destination$ with guard condition `check_occurrence(tr.ip_event)`) function call as shown

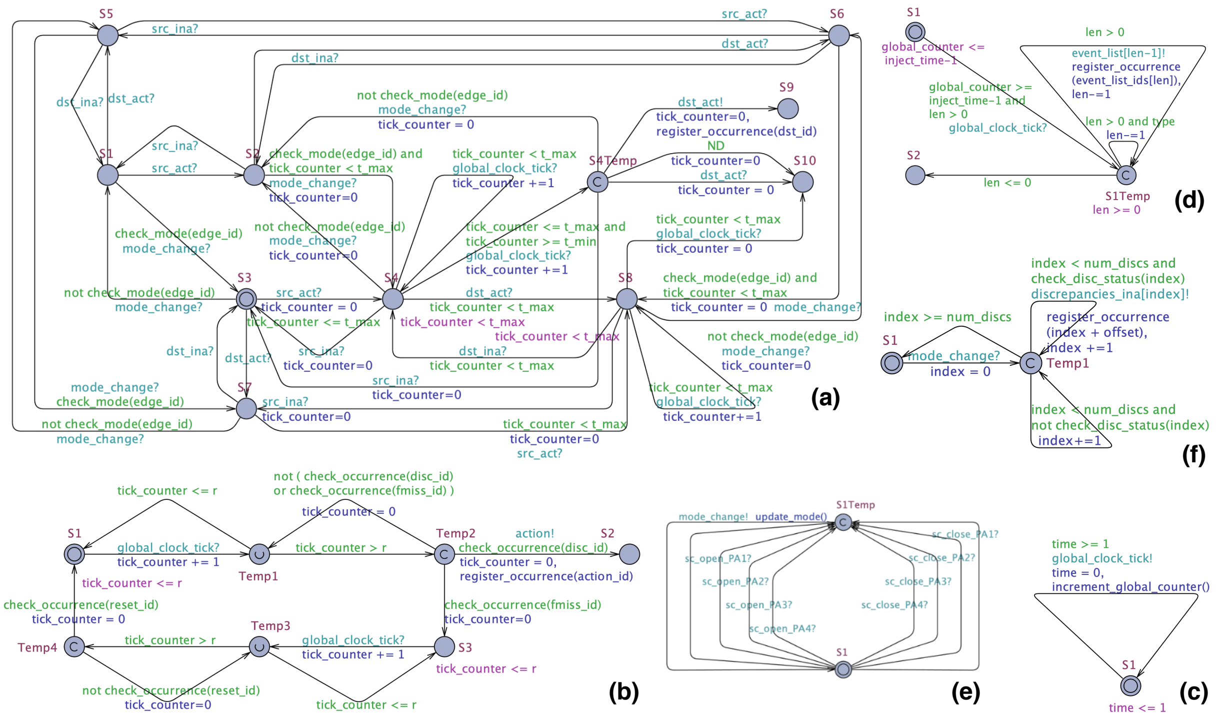

Figure 3: UPPAAL Timed automaton templates for Fault Edge (a), Protection Device (b), Ticker (c), Injector (d), Mode Calculator (e) and Deactivator (f). Locations marked with double circles are initial locations

in Lines [19-25]. Figure 3(b) shows the translated UPPAAL TA associated with TTA model of protection device (example TCD model) described in previous section.

Algorithm 1: TTA to UPPAAL TA translation

```
   Input: $s, T$   Output: $S, T'$
1  Initialize: $Var[tr.constraint] = 0\ \forall tr \in T,\quad S \leftarrow s$
   $Loc[k]$ = Location("committed") $\forall k \in Var$
   $Checked\_once[k] = \bot\ \forall k \in Var$ If $k.type$ == "Instantaneous"
2  begin
3      $temp \leftarrow$ Location("urgent")
4      $S \leftarrow S \cup temp$
5      $T' \leftarrow T' \cup$ Transition($s, \emptyset, global\_clock\_tick!, [Var[k]+ = 1\ \forall k \in Var], temp$)
6      $T \leftarrow T' \cup$ Transition($temp, \bigwedge_k^{Var} var[k] \le k.value, \emptyset, \emptyset, s$)
7      foreach $k \in Var$ do
8          $S \leftarrow S \cup Loc[k]$
9          $T' \leftarrow T' \cup$ Transition($temp, Var[k] > k.value, \emptyset, \emptyset, Loc[k]$)
10         $c \leftarrow \neg(\bigvee_{tr}^{T} check\_occurrence(tr.ip\_event)$ If $tr.constraint == k$ )
11         $u \leftarrow [Var[k] = 0]$
12         if $k.type$ == "Instantaneous" then
13             $c \leftarrow c \cup Checked\_once[k]$
14             $u \leftarrow u \cup Checked\_once[k] = \top$
15         end
16         $T' \leftarrow T' \cup$ Transition($Lock[k], c, \emptyset, u, temp$)
17     end
18     foreach $tr \in T$ do
19         $\phi \leftarrow tr.constraint$
20         $u \leftarrow [Var[k] = 0 \forall k \in Var] \cup register\_occurrence(tr.op\_event)$
21         $c \leftarrow check\_occurrence(tr.ip\_event)$
22         if $\phi.type$ == "Instantaneous" then
23             $c \leftarrow c \cup \neg Checked\_once[\phi]$
24         end
25         $T' \leftarrow T' \cup$ Transition($Loc[\phi], tr.op\_event!, c, u, tr.destination$)
26     end
27 end
```

3.3.3 *Clock Source Automaton.*

: Figure 3(c) shows an UPPAAL TA of clock source, *Ticker* with a single location S1. Ticker periodically resets clock variable, *time*, and broadcasts a synchronizing event, *global_clock_tick*. The automaton stays in the location till, $time < 1$ and at $time = 1$, the self transition is enabled and the location invariant, $time \le 1$ enforces the automaton to take the enabled transition resulting in a broadcasting event, resetting clock variable and increasing the clock counter, *global_counter* by calling function `increment_global_counter()`.

3.3.4 *Injector Automaton.*

: This automaton is responsible for introducing external events such as fault node activation in $\mathcal{G}_{X^r}$. Figures 3(d) shows an *Injector* automaton that injects events in the system at a given instant of time. The input parameters include reference to list of event labels and their associated identifiers, *event_list*, *event_list_ids*, size of the list, *len*, time of injection, *inject_time* and mode of the automaton, *mode*. The automaton has two modes, in first mode, $mode = \top$, the automaton injects all events in the list at specified time. However, in second mode, $mode = \bot$, the automaton randomly injects upto *len* faults. The first mode is predominantly used for concrete simulation while the later is utilized from symbolic simulation and model checking. As shown in Figure 3(d), the automaton consists of three locations with location S1 being the initial location. The Injector automaton stays in the initial location up to *event_time*-1 clock ticks. At *event_time*th clock tick, the transition to committed location S1Temp is enabled and the location invariant associated with S1 forces the automaton to jump to S1Temp. While in S1Temp, the automaton iterates over the list, *event_list* and generates events depending on the parameter *mode*.

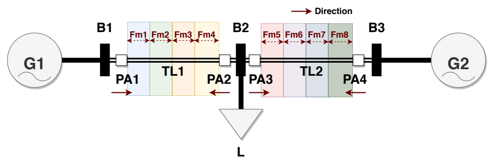

Figure 4: Two transmission line system

If the value of *mode* is ⊥, then in each iteration the automaton can randomly take either of the two self transitions as shown in Figure 3(d) and non-deterministically create events.

3.3.5 Mode Calculator Automaton. : It responds to the events related to change in the location of the actuators. Figure 3(e) shows a mode calculator automaton consisting of two locations with, `S1` being the initial location. When the automaton receives an actuator state change event, (sc_open_PAk, sc_close_PAk) it moves from `S1` to `S1Temp`. The automaton transitions back to `S1` after updating the system mode variable by calling method, `update_mode()` and generating *mode_change* event. The function `update_mode()` iterates over every possible system mode and selects the mode, m for which $\Omega(m)$ evaluates to true.

3.3.6 Discrepancy De-activator Automaton. : The protection devices cause the actuators to change their location in response to the observed fault effects. The resulting mode change as a result of actuator location change can mask the fault effects leading to de-activation of discrepancy nodes. Figure 3(f) shows a discrepancy de-activator automaton consisting of two locations with `S1` being the initial location. The automaton jumps to committed location, `S1Temp` after observing a mode change event. While in `S1Temp`, the automaton iterates over every discrepancy node and generates de-activation event, σ, if all fired fault edges with destination node, $\Psi_{ina}^{-1}(\sigma)$ have become in-active. The condition is checked by the function call, `check_disc_status()` as shown in Figure 3(f).

4 EVALUATION

In this section, we evaluate the TCD based fault modeling approach with the help of a case study from power transmission system. Following sub-sections describe ① the physical and cyber components in CPES, ② the TCD fault model that captures effects of faults in transmission lines and nominal as well as the faulty response of protection systems, ③ simulation and verification results based on the conversion of TCD model to the UPPAAL TA based on the semantics discussed in previous section.

4.1 Case Study: Cyber Physical Energy System

The physical components in CPES such as buses, transmission lines, transformers and generators are protected from faults (short-circuit) with the help of relays and breakers. Figure 4 shows a two transmission line network where two generators, G1, G2 are supplying power to a load, L through lines TL1 and TL2 respectively. The transmission lines are connected to generators and loads via buses (B1, B2, B3) and protection assemblies (PA1, PA2, PA3, PA4). A protection assembly is a collection of relays and breakers that collectively detect and mitigate faults.

Modern relays such as SEL421 [14] offer a wide variety of protection functions. However, we are limiting the scope of this case study to distance protection only as it is the primary protection function used for grounding faults in transmission lines. When a fault is introduced in a transmission line, the current flowing through the conductor increases and the voltage at the bus terminals drops, causing a decrease in the apparent impedance measured by the distance relay. A distance relay issues a trip command to the breaker in the same assembly to clear the fault depending upon the location of the fault. Since apparatus protection in power systems is performed with a high degree of redundancy, a distance relay consists of three relay elements that work in parallel and referred to as zone 1, 2 and 3 elements. A zone 1 relay element detects a fault when the measured impedance falls less the $0.8z_l$, where z_l is the impedance of the primary transmission line connected directly to the protection assembly. It's an under-reaching element that detects a fault in 80% of the line and sends the trip command immediately. A zone 2 element is a time-delayed element, also called an over-reaching element. It covers the primary transmission line along with 50% of the neighboring transmission line such that the impedance threshold is $z_l + 0.5z_{l'}$ where $z_{l'}$ is the impedance of the neighboring line. Zone 2 relay element waits for a specified amount of time, $z2wt$ before issuing the breaker trip command so that the zone 1 element of the neighboring line is able to clear the fault first. The delay is introduced to make sure the minimum number of power system equipment are removed from the system. However, if the fault is close to one end of the transmission line then the relay on the other end will have to wait for $z2wt$ before issuing a trip command. To reduce this wait time, a number of pilot trip protocols are used, one such protocol is pilot under-reaching trip transfer (PUTT) where zone 1 element of one relay sends a trip transfer signal to the zone 2 relay element on the other end. After receiving the PUTT signal, the relay skips the wait time and issues a trip command to the associated breaker. Similar to zone 2 element, the zone 3 element is a time delayed element ($z3wt$) that covers the primary and neighboring transmission lines completely such that the impedance threshold becomes $z_l + z_{l'}$. Typical values of $z2wt$ and $z3wt$ are (0.1-0.25) and (1-2) seconds respectively [14].

Table 3: TCD model: Two line transmission syste,

TCD Element	Two line transmission system TCD model
Faults (F)	$\{Fm1, Fm2, ...Fm8, Fmiss^{PAk}, Fstuck^{PAk}$ $\forall k \in (1, 2, 3, 4)\}$
Discrepancies (D)	$\{D1^{PAk}, D2^{PAk}, D3^{PAk} \forall k \in (1:4)\}$
Edges (E)	Illustrated in Figure 5
System modes (M)	$\{m0, m1, .., m15\}$
Fault propagation (ET)	$[0.016, 0.032] \forall e \in E$
Edge-Mode Map (EM)	Illustrated in Figure 5
Edge Uncertainty (ND)	$\perp \forall e \in E$
Events (Σ)	$\{fm1, ..., fm8, d1^{PAk}, d2^{PAk}, d3^{PAk}, d1'^{PAk}, d2'^{PAk}, d3'^{PAk}, fmiss^{PAk}, fstuck^{PAk}, sc\_open^{PAk}, sc\_close^{PAk}, cmd\_open^{PAk}, cmd\_close^{PAk} \forall k \in (1:4)\}$
Automaton locations (Q)	{ (IDLE, MISSED, TRIPPED) × 12, (WAIT) × 8, (OPEN, CLOSE, OPENING, CLOSING, STUCK_CLOSE, STUCK_OPEN)× 4 }
Initial Locations (Q_0)	{(IDLE) × 12, (CLOSE)× 4}
Location-Mode map (Ω)	Illustrated in Equations 5
Activation event map (Ψ_{act})	$\Psi_{act}(D1^{PAk}) : d1^{PAk}, \Psi_{act}(D2^{PAk}) : d2^{PAk}, \Psi_{act}(D3^{PAk}) : d3^{PAk}, \Psi_{act}(Fmiss^{PAk}) : fmiss^{PAk}, \Psi_{act}(Fstuck^{PAk}) : fstuck^{PAk} \ \forall k \in (1:4), \Psi_{act}(Fmi) : fmi, \forall i \in (1:8)$
De-activation event map (Ψ_{ina})	$\Psi_{ina}(D1^{PAk}) : d1'^{PAk}, \Psi_{ina}(D2^{PAk}) : d2'^{PAk}, \Psi_{ina}(D3^{PAk}) : d3'^{PAk} \ \forall k \in (1:4),$
Timing constraints (Φ)	$\{(r^{PAk}), [z2wt^{PAk}], [z3wt^{PAk}], [tto^{PAk}], [ttc^{PAk}] \forall k \in (1:4)\}$
Transitions (T)	Illustrated in Figure 5

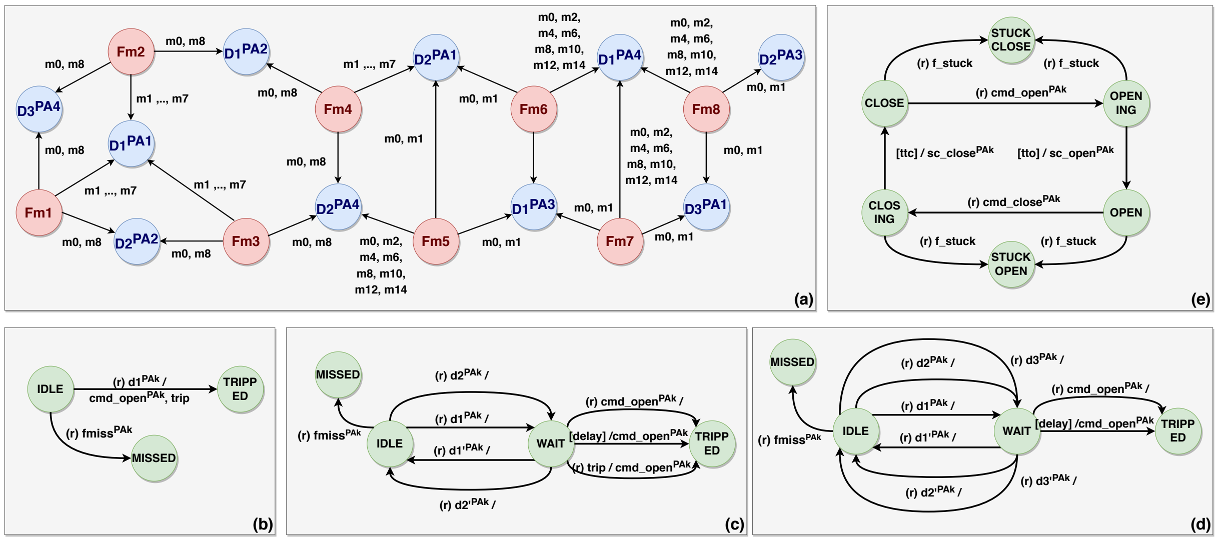

Figure 5: TFPG fault propagation edges (a) and TTA templates for protection system: zone 1 element(b), zone 2 element (c), zone 3 element(d), breaker (e)

4.2 Fault Models

Based on the behavior and redundant arrangement of distance relays, a signature of a fault in the transmission line can be created in terms of the relay responses as described in [10]. A transmission line can be divided into sections such that fault anywhere in a section produces same response from all protection assemblies as shown in [9]. For instance, fault anywhere in the left-most section of transmission line $TL1$, labeled as $Fm1$ causes (zone 1, 2, 3), (zone 2, 3) and (zone 3) elements associated with protection assemblies $PA1$, $PA2$ and $PA4$ to detect the reduction in impedance. Thus, a TFPG graph can be created where fault nodes ($Fm1$, $Fm2$, ..., $Fm8$) are related to faults in transmission lines and the discrepancies ($D1^{PA1}$, $D2^{PA1}$, $D3^{PA1}$, ..., $D1^{PA4}$, $D2^{PA4}$, $D3^{PA4}$) signify reduction in impedance as shown in Figure 5(a). There are three discrepancies associated with a protection relay, PAk, ① $D1^{PAk}$ implies the apparent impedance measured by the relay is less than $0.8z_l$ and every element of the relay detects it, ② $D2^{PAk}$ signifies the impedance is greater than $0.8z_l$ but less than $z_l + 0.5z_{l'}$ such that zone 2 and 3 elements detect it and ③ $D3^{PAk}$ denotes the impedance is between $z_l + (0.5z_{l'}, z_{l'})$ such that only zone 3 element detects it. The propagation time for each edge is in the range [0.016 - 0.032] secs [14] and the mode condition depends upon the state of the breakers such that there exists a path for power to flow between the location of fault and generators. For instance, in order for relay elements in $PA2$ to detect fault conditions due to $Fm1$, the breakers, $PA2\_br$, $PA3\_br$, $PA4\_br$ must be closed. We identify 16 system modes, (m0-m15) based on combinations of breaker states (open or close) given by Equations 5.

$$\Omega(m0) = \bigwedge_{i=1}^{4} \overbrace{(CLOSE \vee STUCK\_CLOSE)}^{PAi\_br} \quad \Omega(m15) = \bigwedge_{i=1}^{4} \overbrace{(OPEN \vee STUCK\_OPEN)}^{PAi\_br} \tag{5}$$

The complete TCD model for two transmission line system is illustrated in Table 3. Figure 5(b) shows TTA template of a zone 1 relay element that consists of three locations with IDLE being the initial location. While in IDLE, the automaton periodically checks for $fmiss^{PAk}$ and $d1^{PAk}$ events . These events are related to the activation of $Fmiss^{PAk}$ and $D1^{PAk}$ nodes respectively. If the missed detection fault is present, then the automaton transitions to MISSED. On the other hand, if reduction in impedance is detected, then the automaton moves to TRIPPED and generates a pilot trip event, $trip$ and actuation command, cmd_open^{PAk} for breaker, PAk_br. Figure 5(c) shows the TTA template for a zone 2 relay element with an additional location, WAIT. While in initial location, IDLE, the automaton checks for events related to missed detection fault, $fmiss^{PAk}$ and reduction in impedance, ($d1^{PAk}$ or $d2^{PAk}$). If discrepancy activation event is observed, the automaton jumps to WAIT. In WAIT, the automaton waits for $delay$ number of clock ticks and then jumps to TRIPPED while generating event cmd_open^{PAk}. The wait time is skipped if a $trip$ event is observed from a protection assembly connected to the other end, or the zone 1 element of the same assembly had issued the cmd_open^{PAk} event before the wait period expires. In case, the fault is cleared by some other protection assembly, a discrepancy de-activation event, $d1'^{PA_k}$ or $d2'^{PA_k}$ can force the automaton back to IDLE . The zone 3 element automaton is similar to zone 2 with two major differences, ① zone 3 automaton responds to three discrepancy activation and de-activation events, $(di^{PAk}, di'^{PAk})\ \forall i \in \{1, 2, 3\}$, ② zone 3 wait time cannot be skipped by pilot trips as shown in Figure 5(d).

4.3 Simulation Results

Tables 4, 5 and 6 show the trace of three simulation scenarios. In the first scenario, a physical fault is introduced in the leftmost section of the transmission line, $TL1$ by injecting an external event through Injector automaton, $fm1\_injector$ at time t = 1. The zone 1, 2 and 3 elements of protection assembly, $PA1$, followed by zone 2, 3 elements of $PA2$ and zone 3 element of $PA4$ detect the fault. The zone 1 element of $PA1$ sends a trip command to the breaker, $PA1\_br$ and sends a PUTT signal to $PA2$. The zone 2 element of $PA2$ picks up the trip signal and commands the breaker $PA2\_br$ to open, thereby shortening the zone 2 wait time of 4 clock ticks. After both breakers have cleared the fault, the system mode changes which leads to the de-activation of all discrepancy nodes. The de-activation of discrepancy $D3^{PA4}$, forces zone 3 element of $PA4$ to return back to `IDLE` as illustrated in Table 4. The second scenario extends the first one by introducing a missed detection fault in all relay elements associated with $PA2$ at t = 1 with the help of an additional injector automaton, $missed\_fault\_injector$. The relay elements in $PA1$ detect the fault and zone 1 element instructs the breaker to trip. However, relay elements in $PA2$ do not detect the fault, causing the backup protection assembly $PA4$, to issue an open command to the breaker $PA4\_br$ after waiting for 5 clock ticks as summarized in Table 5. The third scenario also extends the first one by injecting a stuck fault in the breaker associated with $PA2$ at t=1. In this scenario, protection relays in $PA1$ and $PA2$ behaves exactly the same as in first scenario. However, the breaker $PA2\_br$ fails to open because of the stuck fault, causing the backup protection assembly $PA4$, to issue an open command to the breaker $PA4\_br$ as summarized in Table 6. To reduce the length of traces, the value of periodic timing constraint, r, and zone wait times, $z2wt$, $z3wt$ are assumed to be 0, 4, 5 respectively. Moreover, tables 4, 5 and 6 show partial traces of the scenarios, and do not list the intermediate locations different automaton go through. The UPPAAL model and the complete simulation trace for above mentioned scenarios can be downloaded from https://github.com/chhokrad/sam2020.git.

4.4 Verification Results

We verify the translated TCD model, $\mathcal{G}_{X^r}$, using UPPAAL's symbolic model checker to guarantee the correctness of the underlying TCD model, $\mathcal{G}$. The safety of the system can be evaluated against a list of properties encoded in Real-Time Computation Tree Logic (RCTL). Table 7 lists the description of these properties and their corresponding RCTL formulae. We used the same system parameters as described in previous section except ① added a reference real valued clock variable, abs_time to verify the RCTL formulae w.r.t absolute time and ② changed the mode of stuck and missed fault injectors to random, to check the properties in different CPS fault configurations.

5 CONCLUSION

In this paper, we presented a new qualitative fault modeling technique, TCD, that utilizes TFPG to model fault propagation in physical sub-system and TTA to capture the behavior of discrete protection devices. We also described the fault propagation and execution semantics of a TCD model by translating it to a network of UPPAAL TA. In the end, with the help of a case study from CPES, we illustrated the procedure to model, simulate and verify a fault model. In the current work, we restricted the TFPG sub-model to include only one kind of discrepancy, i.e. observable *OR* type discrepancy, which can be activated if at-least one of the parent nodes is active. However, in future work, we would like to include, *AND* type discrepancies that can be activated if all the parent nodes are active.

6 APPENDICES

Definition 1. A timestamped event, k is a tuple, (σ, t_1) such that $\sigma \in \Sigma$ is the event label and $t_1 \in \mathbb{R}_+$ is the time instant at which the event occurs. We define two functions, $\mathcal{T} : I \rightarrow \mathbb{R}_+$, $\mathcal{L} : I \rightarrow \Sigma$, that relates an event $k \in I$ to its time of occurrence and event label. The event k is considered to be valid at time t if $t - \mathcal{T}(k) \leq \epsilon$, where ϵ is a system parameter and depends upon the different timing constraints used in the TCD specification.

ACKNOWLEDGMENTS

This work is funded in part by the National Science Foundation under the award numbers 1329803 and 1840052.

REFERENCES

[1] Sherif Abdelwahed, Sherif Abdelwahed, Gabor Karsai, and Gautam Biswas. 2005. A Consistency-based Robust Diagnosis Approach for Temporal Causal Systems. *IN THE 16TH INTERNATIONAL WORKSHOP ON PRINCIPLES OF DIAGNOSIS* (2005), 73–79.

[2] Sherif Abdelwahed and Gabor Karsai. 2007. Notions of diagnosability for timed failure propagation graphs. In *AUTOTESTCON (Proceedings)*. 643–648.

[3] S. Abdelwahed, G. Karsai, N. Mahadevan, and S.C. Ofsthun. 2009. Practical Implementation of Diagnosis Systems Using Timed Failure Propagation Graph Models. *IEEE Transactions on Instrumentation and Measurement* 58 (2009), 240–247.

[4] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. 1992. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*. Springer, 209–229.

[5] Rajeev Alur and David L. Dill. 1994. A theory of timed automata. *Theoretical Computer Science* 126, 2 (4 1994), 183–235.

[6] Gerd Behrmann, Alexandre David, and Kim G. Larsen. 2004. A Tutorial on UPPAAL. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3185 (2004), 200–236.

[7] Anibal Bregon, Belarmino Pulido, Gautam Biswas, and Xenofon D Koutsoukos. 2009. Generating Possible Conflicts From Bond Graphs Using Temporal Causal Graphs.. In *ECMS*. 675–682.

[8] Christopher Brooks. 2016. Ptolemy II: An open-source platform for experimenting with actor-oriented design.

[9] A. Chhokra, A. Dubey, N. Mahadevan, and G. Karsai. 2015. A component-based approach for modeling failure propagations in power systems. In *2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*. 1–6.

[10] Ajay Chhokra, Nagabhushan Mahadevan, Abhishek Dubey, Daniel Balasubramanian, and Gabor Karsai. 2017. Towards Diagnosing Cascading Outages in Cyber Physical Energy Systems using Temporal Causal Models. In *2017 Annual Conference of the Prognostics and Health Management Society (PHM17)*.

[11] Goran Frehse. 2005. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *International workshop on hybrid systems: computation and control*. Springer, 258–273.

[12] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 379–395.

[13] Hui Ren, Zengqiang Mi, Hongshan Zhao, and Qixun Yang. [n.d.]. Fault diagnosis for substation automation based on Petri nets and coding theory. In *IEEE Power Engineering Society General Meeting, 2004.*, Vol. 2. IEEE, 1038–1042.

[14] Schweitzer Engineering Laboratories Inc. [n.d.]. *High-Speed Line Protection, Automation, and Control System Major Features and Benefits SEL-421 Protection and Automation System.* Technical Report.

[15] H. Kopetz and G. Bauer. 2003. The time-triggered architecture. *Proc. IEEE* 91, 1 (1 2003), 112–126.

[16] Mathworks. R2020a. Stateflow Documentation.

[17] Yiannis Papadopoulos. 2003. Model-based system monitoring and diagnosis of failures using statecharts and fault trees. *Reliability Engineering & System Safety* 81, 3 (9 2003), 325–341.

[18] Vasso Reppa, Marios Polycarpou, and Christos Panayiotou. 2015. Distributed Sensor Fault Diagnosis for a Network of Interconnected Cyber-Physical Systems. *Control of Network Systems, IEEE Transactions on* 2 (03 2015), 11–23.

[19] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. 1996. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology* 4, 2 (3 1996), 105–124.

[20] J. Shiozaki, B. Shibata, H. Matsuyama, and E. O'shima. 1989. Fault diagnosis of chemical processes utilizing signed directed graphs-improvement by using temporal information. *IEEE Transactions on Industrial Electronics* (1989), 469–474.

[21] Stavros Tripakis. 2002. Fault diagnosis for timed automata. In *International symposium on formal techniques in real-time and fault-tolerant systems*. Springer, 205–221.

[22] U.S.-Canada Power System Outage Task Force. 2003. *Causes of the August 14th Blackout in the United States and Canada*. Technical Report. NERC. 134 pages.

Table 4: Simulation trace for scenario 1: Only physical fault

global_ counter	system mode	fm1_ injector	fm1_d1_ pa1_edge	fm1_d2_ pa2_edge	fm1_d3_ pa4_edge	pa1_z1_ element	pa1_z2_ element	pa1_z3_ element	pa2_z2_ element	pa2_z3_ element	pa3_z3_ element	pa1_br	pa2_br2	pa4_br
0	m0	S1	S3	S3	S3	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	CLOSE	CLOSE	CLOSE
1	m0	S2	S4	S4	S4	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	CLOSE	CLOSE	CLOSE
2	m0	S2	S4	S4	S4	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	CLOSE	CLOSE	CLOSE
3	m0	S2	S9	S9	S9	TRIPPED	WAIT	WAIT	WAIT	WAIT	WAIT	OPENING	CLOSE	CLOSE
4	m0	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	WAIT	OPENING	OPENING	CLOSE
5	m8	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	WAIT	OPEN	OPENING	CLOSE
6	m14	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	IDLE	OPEN	OPEN	CLOSE

Table 5: Simulation trace for scenario 2: Physical and missed detection faults

global_ counter	system mode	fm1_ injector	missed_ fault_ injector	fm1_d1_ pa1_edge	fm1_d2_ pa2_edge	fm1_d3_ pa4_edge	pa1_z1_ element	pa1_z2_ element	pa1_z3_ element	pa2_z2_ element	pa2_z3_ element	pa3_z3_ element	pa1_br	pa2_br2	pa4_br
0	m0	S1	S1	S3	S3	S3	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	CLOSE	CLOSE	CLOSE
1	m0	S2	S2	S4	S4	S4	IDLE	IDLE	IDLE	MISSED	MISSED	IDLE	CLOSE	CLOSE	CLOSE
2	m0	S2	S2	S4	S4	S4	IDLE	IDLE	IDLE	MISSED	MISSED	IDLE	CLOSE	CLOSE	CLOSE
3	m0	S2	S2	S9	S9	S9	TRIPPED	WAIT	WAIT	MISSED	MISSED	WAIT	OPENING	CLOSE	CLOSE
4	m0	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	MISSED	MISSED	WAIT	OPENING	CLOSE	CLOSE
5	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	MISSED	MISSED	WAIT	OPEN	CLOSE	CLOSE
6	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	MISSED	MISSED	WAIT	OPEN	CLOSE	CLOSE
7	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	MISSED	MISSED	WAIT	OPEN	CLOSE	CLOSE
8	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	MISSED	MISSED	WAIT	OPEN	CLOSE	CLOSE
9	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	MISSED	MISSED	TRIPPED	OPEN	CLOSE	OPENING
10	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	MISSED	MISSED	TRIPPED	OPEN	CLOSE	OPENING
11	m9	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	MISSED	MISSED	TRIPPED	OPEN	CLOSE	OPEN

Table 6: Simulation trace for scenario 3: Physical and breaker stuck faults

global_ counter	system mode	fm1_ injector	stuck_ fault_ injector	fm1_d1 _pa1_ edge	fm1_d2 _pa2_ edge	fm1_d3 _pa4_ edge	pa1_z1_ element	pa1_z2_ element	pa1_z3_ element	pa2_z2_ element	pa2_z3_ element	pa3_z3_ element	pa1_br	pa2_br2	pa4_br
0	m0	S1	S1	S3	S3	S3	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	CLOSE	CLOSE	CLOSE
1	m0	S2	S2	S4	S4	S4	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	CLOSE	STUCK_CLOSE	CLOSE
2	m0	S2	S2	S4	S4	S4	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	CLOSE	STUCK_CLOSE	CLOSE
3	m0	S2	S2	S9	S9	S9	TRIPPED	WAIT	WAIT	WAIT	WAIT	WAIT	OPENING	STUCK_CLOSE	CLOSE
4	m0	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	WAIT	OPENING	STUCK_CLOSE	CLOSE
5	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	WAIT	OPEN	STUCK_CLOSE	CLOSE
6	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	WAIT	OPEN	STUCK_CLOSE	CLOSE
7	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	WAIT	OPEN	STUCK_CLOSE	CLOSE
8	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	WAIT	OPEN	STUCK_CLOSE	CLOSE
9	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	OPEN	STUCK_CLOSE	OPENING
10	m8	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	OPEN	STUCK_CLOSE	OPENING
11	m9	S2	S2	S9	S9	S9	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	TRIPPED	OPEN	STUCK_CLOSE	OPEN

Table 7: Real-Time Computation Tree Logic Properties

ID	Property	Description
1	`A[] not deadlock`	This property ensures there are no deadlocks in the system and time always evolve.
2	`A[] (time <=1 and time >= 0)`	This property ensures that the clock variable, time, is bounded between [0,1].
3	`A[] abs_time > fm1_injector.inject_time imply fm1_injector.S2 and f_physical_state[0]`	This property ensures that the physical fault, FM1, is injected at appropriate time, where f_physical_state is boolean array that is used for storing the state of physical fault node, where index 0 implies FM1.
4	`A[] (abs_time > (fm1_injector.inject_time + fm1_d1_pa1.t_max) imply fm1_d1_pa1.S9)`	This property ensures that the edge between FM1 and discrepancy $D1^{PA1}$ is fired by (`fm1_injector.inject_time` + `fm1_d1_pa1.t_max`) ticks.
5	`A[] (abs_time > (fm1_injector.inject_time + fm1_d2_pa2.t_max) imply fm1_d2_pa2.S9)`	This property ensures that the edge between FM1 and discrepancy $D2^{PA2}$ is fired by (`fm1_injector.inject_time` + `fm1_d2_pa2.t_max`) ticks.
6	`A[] (abs_time > (fm1_injector.inject_time + fm1_d3_pa4.t_max) imply fm1_d3_pa4.S9)`	This property ensures that the edge between FM1 and discrepancy $D3^{PA4}$ is fired by (`fm1_injector.inject_time` + `fm1_d3_pa4.t_max`) ticks.
7	`A[] (abs_time > (fm1_injector.inject_time + fm1_d1_pa1.t_max ) and not f_missed_state[0] imply pa1_z1_element.TRIPPED)`	This property ensures that if missed detection fault is not present in protection assembly, PA1, then its zone 1 element will trip by `fm1_injector.inject_time + fm1_d1_pa1.t_max` ticks. Array, `f_missed_state` is used to store the state of missed detection faults, where indices 0,1,2 implies protection assemblies PA1, PA2, and PA4 respectively.
8	`A[] (abs_time > (fm1_injector.inject_time + fm1_d2.t_max + pa2_z2_element.delay) and f_missed_state[1] imply pa2_z2_element.TRIPPED)`	This property ensures that if missed detection fault is absent in protection assembly, PA2, then its zone 2 element will trip by `fm1_injector.inject_time + fm1_d2.t_max + pa2_z2_element.delay` ticks.
9	`A[] (abs_time > (fm1_injector.inject_time + fm1_d3.t_max + pa4_z3_element.delay) and not (f_missed_state[2] or f_stuck_state[2]) and (f_missed_state[1] or f_stuck_state[1]) imply pa4_z3_element.TRIPPED)`	This property ensures that if missed detection fault is absent protection assembly, PA4 and cyber fault is present in PA2 (either missed detection in relay or stuck fault in breaker), then zone 3 element of PA4 will trip by `fm1_injector.inject_time + fm1_d3.t_max + pa4_z3_element.delay` ticks. Array, `f_stuck_state` is used to store the state of stuck faults, where indices 0,1,2 implies protection assemblies PA1, PA2, and PA4 respectively.
10	`A[] (abs_time > (fm1_injector.inject_time + fm1_d1.t_max + pa2_z2_element.delay) and not (f_missed_state[0] or f_stuck_state[0] or f_missed_state[1] or f_stuck_state[1] or f_missed_state[3] or f_stuck_state[3])) imply not (discrepancy_state[0] or discrepancy_state[1] or discrepancy_state[2]))`	This property ensures that if no cyber fault is present the all discrepancies should stop signaling by `fm1_injector.inject_time + fm1_d1.t_max + pa2_z2_element.delay` ticks. Array `discrepancy_state` is used to store the state of discrepancy nodes, where indices 0,1,2 signifies discrepancies $D1^{PA1}$, $D2^{PA2}$ and $D3^{PA4}$ respectively