DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Reference herein to any social initiative (including but not limited to Diversity, Equity, and Inclusion (DEI); Community Benefits Plans (CBP); Justice 40; etc.) is made by the Author independent of any current requirement by the United States Government and does not constitute or imply endorsement, recommendation, or support by the United States Government or any agency thereof.

ORNL/TM-2025/3840

Oak Ridge National Laboratory Smart Contracts for Power Grid Applications Using the Advanced DLT Cyber Grid Guard Testbed

Emilio C. Piesciorovsky
Gary Hahn
Raymond Borges Hink
Aaron Werth

June 2025

OAK RIDGE National Laboratory

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2025/3840

Electrification and Energy Infrastructures Division

SMART CONTRACTS FOR POWER GRID APPLICATIONS USING THE ADVANCED DLT CYBER GRID GUARD TESTBED

Emilio C. Piesciorovsky
Gary Hahn
Raymond Borges Hink
Aaron Werth

June 2025

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT-BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

3LG	three-line-to-ground
ANSI	American National Standards Institute
APF	average power factor
BK	breaker
CAST	Center for Alternative Synchronization and Timing
CGG	Cyber Grid Guard
CGGS	Cyber Grid Guard System
DER	distributed energy resource
DLT	distributed ledger technology
DOE	US Department of Energy
GOOSE	Generic Object-Oriented Substation Event
HLF	Hyperledger Fabric
IEC	International Electrotechnical Commission
IED	intelligent electronic devices
IEEE	Institute of Electrical and Electronics Engineers
IRIG-B	inter-range instrumentation group time code B
LG	line-to-ground
LL	line-to-line
MVAR	MegaVolt-Ampere Reactive
ORNL	Oak Ridge National Laboratory
POI	point of interconnection
PTP	precision-time protocol
SC	smart contract
SCADA	supervisory control and data acquisition
SEL	Schweitzer Engineering Laboratories
SLG	single-line-to-ground
TPF	total power factor
TSC	time source clock
VAR	Volt-Ampere Reactive
VDC	Volts Direct Current
VSL	voltage service limit

ABSTRACT

This study presents two power system applications with distributed ledger technology (DLT) and smart contracts (SC) that were assessed in a Cyber Grid Guard System (CGGS) advanced testbed, with protective relays, power meters, communication devices, DLT devices, synchronized time source, clock displays, and real-time simulator. The first application is an SC to define and control the allowable total power factor (TPF) of the distributed energy resource (DER) output (e.g., wind farm), and the terms of the SC are implemented using DLT with a CGGS for a customer-owned DER [1]. The TPF SC was implemented by the CGGS using DLT. The experimental model was performed with a real-time simulator using a CGGS and relay in-the-loop. Data collected from the CGGS were used to execute the TPF SC. The TPF limits were between +0.9 and +1.0, and the breakers' operation in the point of interconnection (POI) was controlled by the relay using the SC. The events were collected from the real-time simulator, CGGS, and SEL 700GT relay (from Schweitzer Engineering Laboratories) to validate a successful application of the TPF SC using DLT. The second application is an SC to measure and control the allowable voltage service limits (VSLs) by the CGGS using DLT. The tests were performed using a real-time simulator, CGGS, and relay in-the-loop. The data were collected from the CGGS that executed the SC. The main constraints were defined based on American National Standards Institute (ANSI) C84.1 service voltage limits and on the operation of the breakers in the POI. The events were collected from the CGGS and an SEL 700GT relay to assess a successful operation of the VSL SC using DLT.

1. INTRODUCTION

1.1 POWER SYSTEM AND SMART CONTRACT APPLICATIONS

The number of distributed energy resources (DERs) interconnected to utility systems and the complexity of the configurations of these interconnections have both significantly increased [2]. The characteristics of energy produced by customer-owned DERs using renewable energy sources are often not as stable as that from other sources. DER-produced power may show variations from nominal voltages, frequency, and power factor. One approach to safely incorporate DERs into a power system is to install a relay at the point of interconnection (POI) that can isolate the DER from the power grid if characteristics of the DER's electricity production become out of tolerance for power quality ranges, based on measuring the power factor and voltage magnitudes. This report defines the rules of engagement for the DER (i.e., a customer-owned wind farm) to connect at the electrical utility grid through a smart contract (SC). The DER transmits the operating and status data securely using distributed ledger technology (DLT) implemented with a Cyber Grid Guard System (CGGS), using a relay at the POI when the DER remains disconnected to maintain power system stability.

SCs are digital contracts stored on a distributed ledger that are automatically executed when predetermined terms and conditions are met [3] between a power grid utility and customer-owned DERs. The increased number of customer-owned DERs is accompanied by relays deployed at the POIs. Each renewable DER and relay require communications and data management at the POI, and the integrity of the data communicated at the POI—between the DERs and the bulk power system—is vital. DLTs are decentralized storage platforms that maintain data integrity without requiring mutual trust among participants [4]. Security, governance, and scalability are prominent subjects of recent advancements in DLT and SCs.

Security is still a critical issue, with ongoing efforts to improve SC safety using libraries like OpenZeppelin and formal verification methods [5]. For example, composite SCs, which are SCs that call

other SCs, introduce additional security challenges. These could be addressed using finite state machine models and checking techniques. Other relevant formal methods include the Observe-based Statistical Model Checking framework, which is designed for verification of complex software systems [6]. The BlockASP framework leverages aspect-oriented programming to address the dynamic nature of blockchain systems, including SCs, by enabling flexible instrumentation for monitoring runtime events and facilitating analysis [7]. Scalability is addressed through techniques like sharding [8, 9] and rollups [10, 11], particularly in the context of public DLTs and off-chain storage strategies [12]. Transitioning to consensus mechanisms like Proof-of-Stake from Proof-of-Work reduces resource consumption and enhances scalability [13], which is a significant trend pertinent to incorporating DLT into the regulated energy sector to meet regulatory standards [14]; the possibility also exists for use in the deregulated merchant generation bid DER generation environment.

Traditional cybersecurity solutions in protective relays and centralized monitoring systems managing DERs lack the decentralization, transparency, and immutability of SCs and DLT [15]. SCs execute predefined agreements automatically and independently [16]. Cryptographic primitives such as hashing and digital signatures are used by DLTs to protect data integrity and secure sensitive information against unauthorized access [17]. The use of DLT to solve cybersecurity problems and improve resilience within power grids has been examined in recent studies, especially in distributed generation systems [18]. For example, one study implemented a prototype using Hyperledger Fabric DLT and IoT for metering and billing small consumers, emphasizing its scalability, low energy requirements, and reduced vulnerability to cyberattacks owing to data encryption and decentralized architecture [19]. A fault location and traceability system based on DLT was developed to protect IoT sensor data integrity, which is vital for grid security and reliability [20]. A realistic electric grid substation testbed was developed for researching fault detection and cyberattack scenarios involving DERs [21], which was used for executing the total power factor experiments described in this study.

Most potential energy applications of DLT have been based on software simulations [22–26]. However, software simulation algorithms sometimes could not be directly integrated with intelligent electronic devices (IEDs) like protective relays or power meters. For example, problems could arise when IEDs have different measurement condition behaviors. When the circuit breakers are opened, IEDs could measure a frequency of 60 Hz and/or a power factor of 1.00 [27] or have different power factor sign conventions depending on Institute of Electrical and Electronics Engineers (IEEE) and International Electrotechnical Commission (IEC) standards [28]. This study used a real-time simulator with a CGGS using DLT and SC for a Schweitzer Engineering Laboratories (SEL) 700GT relay. This testbed was established at Oak Ridge National Laboratory (ORNL) [29–31]. The CGGS with DLT used SCs to study the control application for improving the total power factor (TPF) and voltage service limits (VSLs) in the POI between the electrical utility grid and the customer-owned wind farm.

In this report, the CGGS with DLT and SC has been shown as a viable methodology to improve the integrity of data received from the breakers at the grid and customer-owned DER (wind farm) sides. The TPF and VSL SC were activated with a time window of 450 s because voltage and frequency protection elements could have a maximum time setting of 120 s and 400 s [27], respectively, and traditional power quality improvement methods (e.g., load shedding, capacitor banks, or transformer/load tap changers) could be applied on the utility grid side, too. Also, the 450 s time window avoids having the TPF and VSL SC operating breakers during anomaly events like disturbances or electrical faults. The TPF SC flowcharts were developed by using the power factor explanations [32] and setting the condition of good power factor between +0.90 and +1.00; electrical utilities usually adjust customer bills for a power factor smaller than +0.90 [33]. However, the VSL SC flowchart was developed by using the ANSI C84-1 Standard [34]. Considering a nominal voltage of 7,200 V, the phase voltages measured at the grid side must be between the 0.95 and 1.058 of nominal voltage limits or between 6,840 and 7,620 V. The reasons

for this are that the SEL 700GT relay is not located at the end of the user load and the nominal voltage is greater than 600 V, according to ANSI C84-1 Standard [34].

2. ELECTRICAL SUBSTATION GRID TESTBED WITH CYBER GRID GUARD

2.1 ELECTRICAL SUBSTATION AND GRID WITH CUSTOMER-OWNED DER

The TPF and VSL SCs were applied in a simulated power grid using real IEDs that monitored, managed, and controlled the SC–based applications. This testbed used a software model–simulated power grid. The testbed has a trigger-event signal system and a synchronized time source system. The CGGS, relays, and meters were synchronized with the time source system. The trigger-event signal system was created for recording the relay's events at the desired time. The power grid diagram and equipment are in Figure 1a and Figure 1b, respectively.

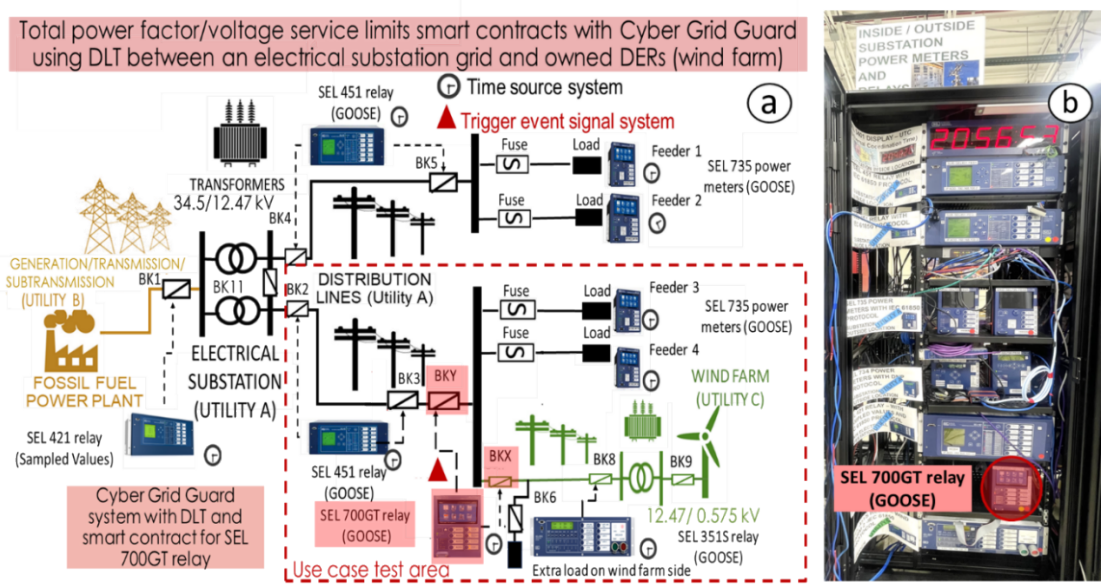
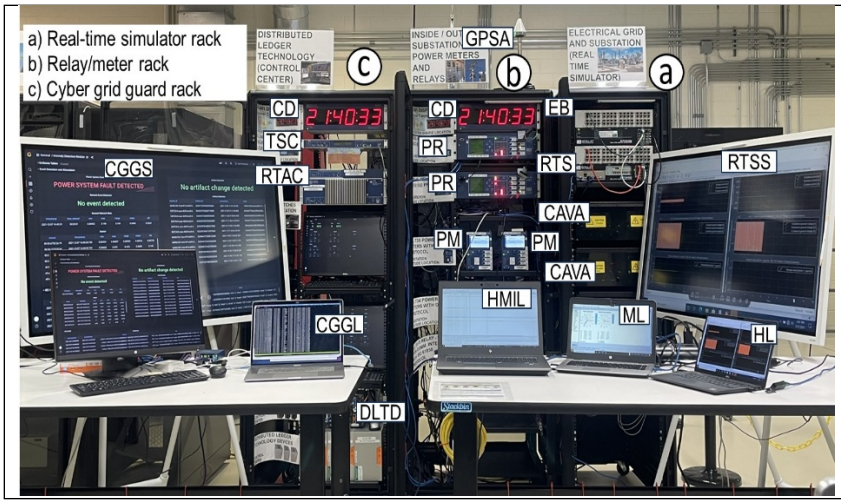


Figure 1. (a) Power grid diagram and (b) photo of the equipment. (GOOSE = Generic Object-Oriented Substation Event)

The main goal of this testbed was to assess the DLT architecture in executing the TPF and VSL SCs in the POI. The sectionalized bus substation configuration [35] and power grid with one customer-owned DER (wind farm) are presented in Figure 1a. Utility A has one substation, using primary and secondary voltages of 34.5 and 12.47 kV, respectively, and the power lines were connected to load feeders. The loads could be fed by the wind farm (Utility B) or substation (Utility A). The customer-owned DER has six 1.5 MW wind turbines (Utility B) and a fossil fuel power plant in Utility C. The rack shown in Figure 1b has the meters [36] and relays [37–39], all of which use the Generic Object-Oriented Substation Event (GOOSE) IEC 61850 protocol. In this experimental model, the test scenarios were run into the red dashed-line square area (Figure 1a) to evaluate the CGGS for the TPF and VSL SCs in the POI, using an SEL 700GT relay.

2.2 TESTBED PLATFORM

The testbed had the CGGS using DLT and SCs, and the IEDs (meters and relays) were wired to a real-time simulator. This testbed has a real-time simulator rack, a relay/meter rack, and a CGGS rack (Figure 2). The four laptops in Figure 2 were used to run and control the use case scenarios. The synchronized time system was connected to the CGG, meters, and relays.



CAVA = current and voltage amplifier; CD = clock display; CGGL = Cyber Grid Guard laptop; CGGS = Cyber Grid Guard System screen; DLTD = distributed ledger technology device; EB = expansion box; GPSA = global positioning system antenna; HL = host laptop; HMIL = human–machine interface laptop; ML = monitoring laptop; PM = power meter; PR = protective relay; RTAC = real-time automation controller; RTS = real-time simulator; RTSS = real-time simulation screen; and TSC = time source clock.

Figure 2. Testbed with relays, meters, and workstation computers.

The architecture of the testbed, shown in Figure 3, has four layers. The first layer—physical—includes the circuit breakers, power lines, and other grid elements simulated by the real-time simulator. The second layer—protection and metering—has the hardware-in-the-loop, represented by the relays and meters. The third layer—automation—has the Ethernet switches and remote terminal units. The fourth layer includes the CGGS, synchronized time system, trigger-event system, supervisory control and data acquisition (SCADA), and human–machine interface. In this testbed, the SEL 700GT relay transmitted IEC 61850 GOOSE messages. The relay recorded the test events by receiving a digital signal from the trigger-event system. The implemented synchronized-time protocol was based on the inter-range instrumentation group time code B (IRIG-B) signals for relays and meters. However, the CGGS through the Ethernet network used a precision-time protocol (PTP) communication.

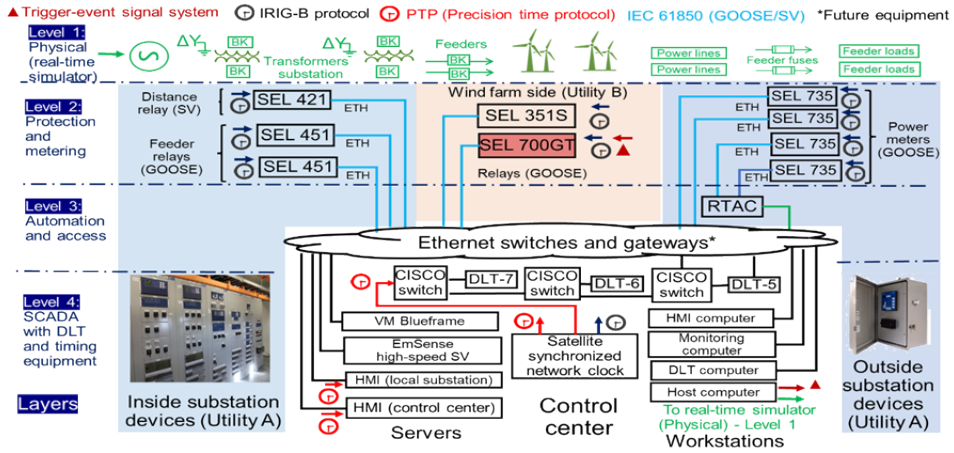


Figure 3. Architecture of testbed. (BK = breaker; ETH = Ethernet; RTAC = real-time automation controller; SV = sampled value; VM = virtual machine).

2.3 TRIGGER-EVENT SIGNAL SYSTEM

The trigger-event signal system (Figure 4) is set on the electrical substation grid testbed with the CGGS (DLT using smart contracts). The trigger-event signal system has the function of recording events of the relays and meters at a specific time selected on the switch connected to the clock (Figure 4a) set in the RT-LAB project before running the simulation. The relays and meters receive the trigger event signal (Figure 4b) from a digital output card (Figure 4c) without delay time from the real-time simulator; this occurs because the signal is provided by an analog cable (rather than by a communication cable) connected from the real-time simulator to the relays and meters (Figure 4d). The use of analog cables instead of communication cables enables recording events in multiple relays and meters at same time because the signal is not affected by the communication latency. The relays and meters receive 24 voltages direct current (VDC) signals for the SEL 421, SEL 451, and SEL 700GT relays and SEL 735 meters and 120 VDC signals for the SEL 351S relay signals on their IN206, IN106, and IN304 relay control inputs and their IN101 meter control inputs (Figure 4d). The logic equation of the trigger events for the relays and meters is set with their control inputs to enable recording the events when the trigger-event signal is received. Next, the COMTRADE and CEV files of events from relays and meters can be collected during normal operation situations that usually are not recorded in devices installed on real power grids but that are required in a testing platform to assess power system smart contract applications using DLT.

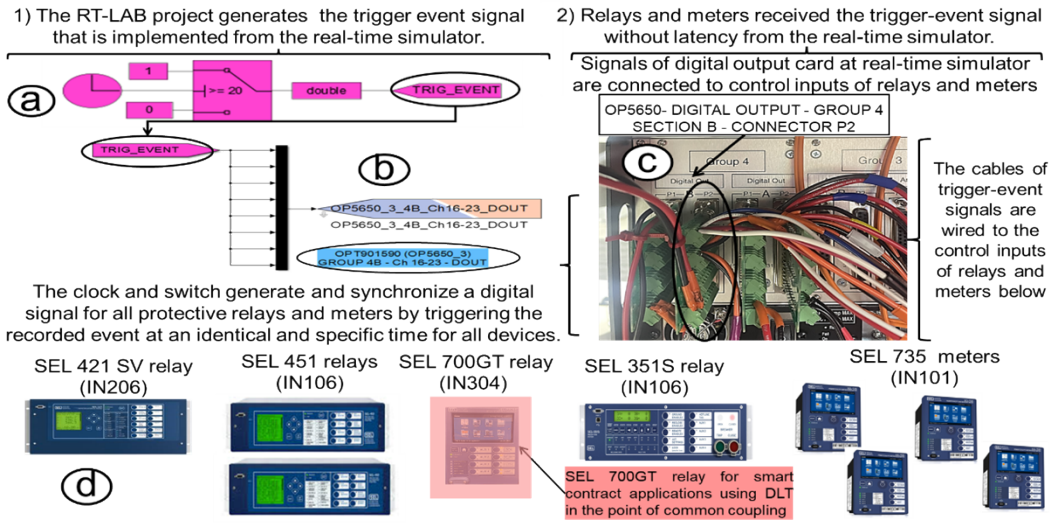


Figure 4. Trigger-event signal system: (a) the specific time is selected on the clock and switch, and (b) a trigger signal is sent from (c) a digital output card to (d) the relays and meters.

2.4 SYNCHRONIZED TIME SOURCE SYSTEM

The synchronized time source system (Figure 5) is set on the electrical substation grid testbed with the CGGS (DLT using smart contracts). The system synchronizes the CGGS and relays/meters with the same time stamps to compare events from the DLT system and relays/meters for the performed tests during the smart contract experiments. The synchronized time source system receives the time signal from the Center for Alternative Synchronization and Timing (CAST) with a PTP and a network-time protocol. In the synchronized time source system (Figure 5), the CAST time signal is received into the Boundary DarkNet CAST clock and then sent to the CISCO Ethernet switches (Figure 5a) of the CGGS as PTP signals. However, the relays/meters (Figure 5b) receive the time signals as IRIG-B protocol through the SEL 3401 clock displays also connected to the Boundary DarkNet CAST clock. The synchronized time source system also has a backup time source provided by an SEL 2488 clock, connected in case the

CAST signal is interrupted. The CAST and SEL 2488 clocks provide a secure and reliable synchronized time signal for the CGGS (Figure 5a) and relays/meters (Figure 5b) in the electrical substation grid testbed.

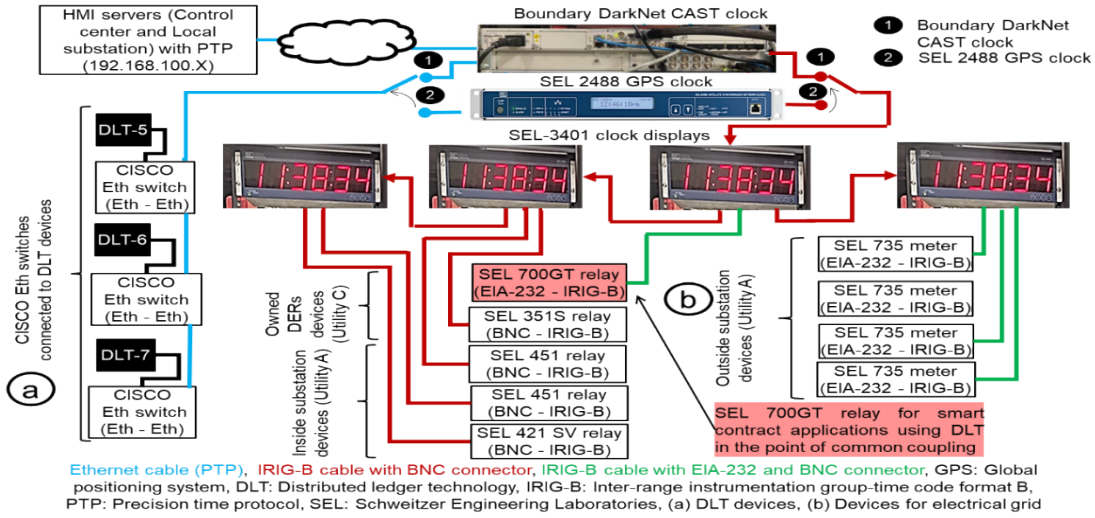


Figure 5. Synchronized time source system diagram: (a) DLT devices and (b) devices for the electrical grid. (BNC = Bayonet Neill–Concelman; EIA = Electronic Industries Alliance; Eth = Ethernet.)

2.5 RT-LAB PROJECT

In the RT-LAB project, a three-phase power system diagram was created with MATLAB/Simulink (2015b version) software models, and this power system was set into a real-time simulator with relays and meters in-the-loop. The RT-LAB software is fully integrated with MATLAB/Simulink to interact with the real-time simulator. The real-time simulator uses the RT-LAB software as the model-simulation platform for the power system simulations. The RT-LAB software provides a flexible solution for power system test scenarios, similar to other research applications [40–43]. This testbed platform performed different use case scenarios, such as normal operation and temporary electrical fault tests. Because electrical faults are not likely to be setting in a real power grid on the field, the proposed real-time simulator platform offer the main advantage of testing electrical fault scenarios in a safe environment. Electrical fault tests are undesired testing situations because they can damage electrical equipment (e.g., power transformers, capacitor banks, power lines, generators) in real power grids. Therefore, the real-time simulator with relays and meters in-the-loop was the best testbed platform to perform the TPF and VSL SCs with the CGGS using DLT.

In shown in Figure 6, the main substation and wind farm were given by 34.5/12.47 kV and 0.575/12.47 kV voltage systems, respectively. The six 1.5 MW wind turbines installed on the DER side had a voltage of 575 V. Two power transformers were connected in parallel in the main substation, and two power lines were connected to the load feeders with 50 T and 100 T fuses based on the maximum load feeder currents [44]. In the POI, the SEL 700GT relay controlled the breaker BKY (grid side) and BKX (wind farm side). The BKY/BKX circuit breaker states, TPF, and phase voltages were measured in the CGGS with DLT and SCs. Then, the BKY/BKX circuit breakers were operated based on the TPF and VSL boundary conditions. In the POI, the TPF and VSL SCs conditions were between +0.90/+1.00 (IEEE Standard power factor convention for the SEL 700GT relay), and 6,840/7,620 V (ANSI C84.1 Standard), respectively. The main goal was based on reducing the reactive power and improving the voltage service on the load feeders, considering a time window operation of 450 s, to permit the application of traditional power quality techniques (e.g., capacitor banks, load shedding, transformer/load tap changers) from the

grid-side utility without use of the TPF or VSL SCs. The CGGS using DLT and SCs were used in the POI between the electrical utility grid and customer-owned wind farm. Figure 6 shows the three-phase power system diagram. The substation grid (Utility A) can be connected to the customer-owned wind farm (Utility B). Utility C is a large fossil power plant with transmission and subtransmission models. Figure 6 **Error! Reference source not found.**shows the descriptions and function of items used in the three-phase power system diagram of the substation and grid with the customer-owned wind farm.

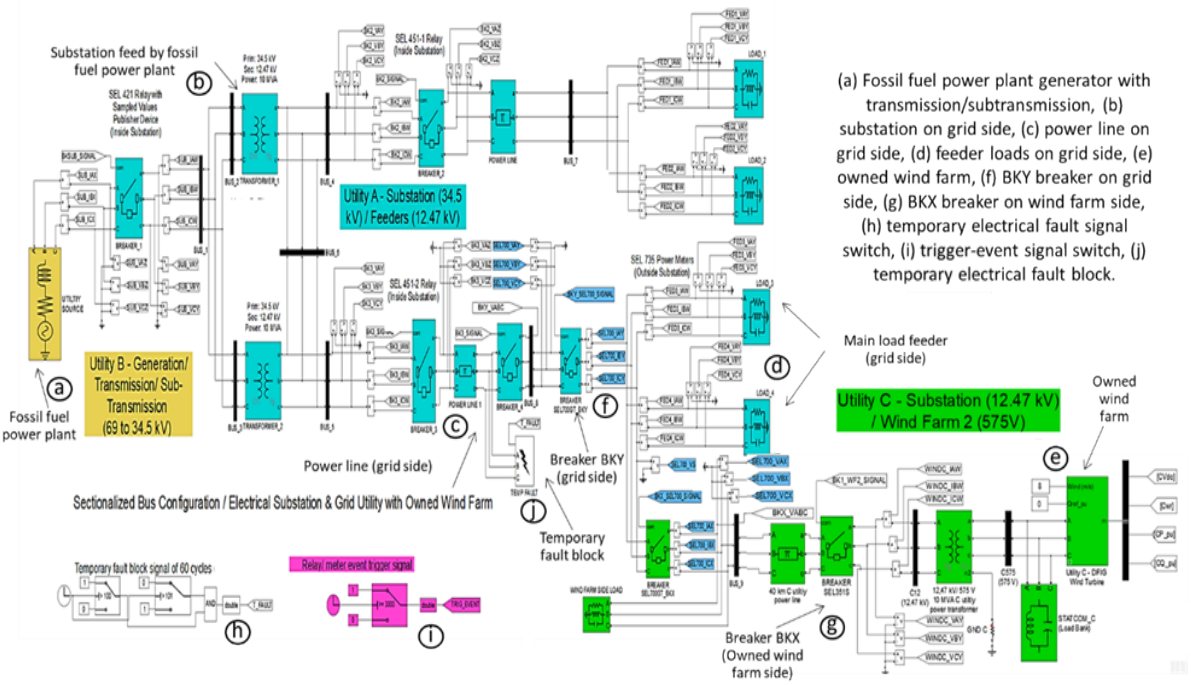


Figure 6. Three-phase power system diagram of the substation and grid with customer-owned wind farm.

Table 1. Components of the three-phase power system diagram.

Component location		Component name	Component function
Area	**Figure**		
Utility C	6a	Fossil fuel power plant	The fossil fuel plant feeds the main substation
Utility A	6b	Main substation	The main substation feeds the power lines
Utility A	6c	Power lines	The power lines feed the main loads
Utility A	6d	Main loads	The main loads are the customers
Utility B	6e	Wind farm customer-owned utility	The wind farm customer-owned utility feeds the main loads
Utility A	6f	Breaker BKY on grid side	The breaker BKY feeds the main loads from the main substation and power lines of Utility A
Utility B	6g	Breaker BKX on wind farm side	The breaker BKX feeds the main loads from the wind farm of Utility B
System	6h	Temporary fault circuit	The temporary fault circuit generates the signal for the temporary electrical faults during 60 cycles on the grid-side power line
	6i	Trigger-event circuit	The trigger-event circuit generates a 24 VDC signal sent to a SEL 700GT relay for recording the events during the simulation tests at a specific time
	6j	Temporary fault block	The block generates the temporary electrical fault block

In this experimental model, a power gui block with a discrete simulation was set in the RT-LAB project. This block uses numerical integration methods for simulating the differential equations of the power system, based on the Tustin (bilinear transform) solver and Backward Euler solver with a time step of 50 µs. The tests were set from the host computer before the simulations were performed. The test properties were set at a target platform with OPAL-RT Linux, real-time simulation mode with hardware synchronized, real-time communication link, time factor of 1.0, and stop/pause time of 600 s.

3. THEORY AND EQUATIONS

3.1 TOTAL POWER FACTOR SMART CONTRACT

In the TPF SC, the theory for the total power factor was considered, with the power factor convention signs based on IEEE or IEC standards [28]. However, the TPF SC was adapted to the total power factor measured by the SEL 700GT relay and its total power factor convention signs.

3.1.1 Power Factor

The power factor [45] is represented by Eq. (1) as the ratio of the true and apparent powers. The product of voltage and current by their cosine angle is denominated as true power. The apparent power is given by the product of current and voltage. A power factor magnitude of less than one means the current and voltage are not in phase. A negative power factor means the true power flows back toward the source.

$$PF = \frac{P}{S} = \frac{V \times I \times cos(\theta_V - \theta_I)}{V \times I} = cos(\theta_V - \theta_I) \text{ ,\#(1)}$$

where PF is the power factor in unity, P is the true power in watts, S is the apparent power in volt-amperes, V is the magnitude of phase voltage in volts, I is the magnitude of phase current in amps, Θ_V is the angle of phase voltage in degrees, and Θ_I is the angle of phase current in degrees.

3.1.2 Average and Total Power Factors

The average of the power factor (APF) is the average of the measured power factor for the A, B, and C phases. The average of the power factor is calculated with the power factor for each phase by Eq. (1), then the phase current/voltage magnitudes and angles for each phase are measured. Finally, the average power factor is given by the sum of the power factor for the A, B, and C phases divided by 3, as shown in Eq. (2)

$$APF = \frac{PF_A + PF_B + PF_C}{3} \text{ ,\#(2)}$$

where APF is the average of the power factor, PF_A is the phase A power factor, PF_B is the phase B power factor, and PF_C is the phase C power factor.

Protective relays and meters can measure the TPF and/or APF. However, the APF does not represent the power factor of a three-phase power system effectively, when the A, B, and C phase loads are unbalanced. For this reason, the TPF represents the best way for measuring the efficiency in three-phase energy systems. The TPF can be calculated as the ratio of the sum for the true power and the sum of apparent power for the A, B, and C phases, as shown in Eq. (3).

$$TPF = \frac{P_A + P_B + P_C}{S_A + S_B + S_C}, \#(3)$$

where *TPF* is the total power factor, P_A is the phase A true power in watts, P_B is the phase B true power in watts, P_C is the phase C true power in watts, S_A is the phase A apparent power in volt-amperes, S_B is the phase B apparent power in volt-amperes, and S_C is the phase C apparent power in volt-amperes.

Some protective relays and/or meters do not provide the measured true and apparent power for each phase (A, B, and C phases), but the TPF can be calculated by Eq. (3) using the measured current/voltage magnitudes and angles for the phases. From Eq. (1) and Eq. (3), the TPF is given by Eq. (4):

$$TPF = \frac{\sum_{m=A}^{C} V_m \times I_m \times cos(\theta_{Vm} - \theta_{Im})}{\sum_{m=A}^{C} V_m \times I_m}, \#(4)$$

where *TPF* is the total power factor, V_m is the generic phase *m* (A, B, C) voltage magnitude in volts, I_m is the generic phase *m* (A, B, C) current magnitude in amperes, Θ_{Vm} is the generic phase *m* (A, B or C) voltage angle in degrees, and Θ_{Im} is the generic phase *m* (A, B or C) current angle in degrees.

3.1.3 Power Factor Convention Signs

The power factor is measured by protective relays and/or meters, and it is in the range of ±1.00. The power factor can be obtained as unity or percentage values, although the unity measurement is the most common method. The efficiency of a power system is represented by the power factor because it represents the losses generated by the reactive energy flowing in a power grid point. The maximum power factor is +1.00, and the minimum power factor limits are usually between +0.80 and +0.98 [45].

Based on an industrial meter manual [28], the measured power factor sign convention could be defined for the IEEE or IEC Standards, and sometimes meters allow setting the power factor sign convention that is used on the display to either IEC or IEEE [28]. The conventional power factor signs based on the IEC and IEEE Standards are shown in Figure 7a and Figure 7b, respectively, based on a meter instruction manual [28].

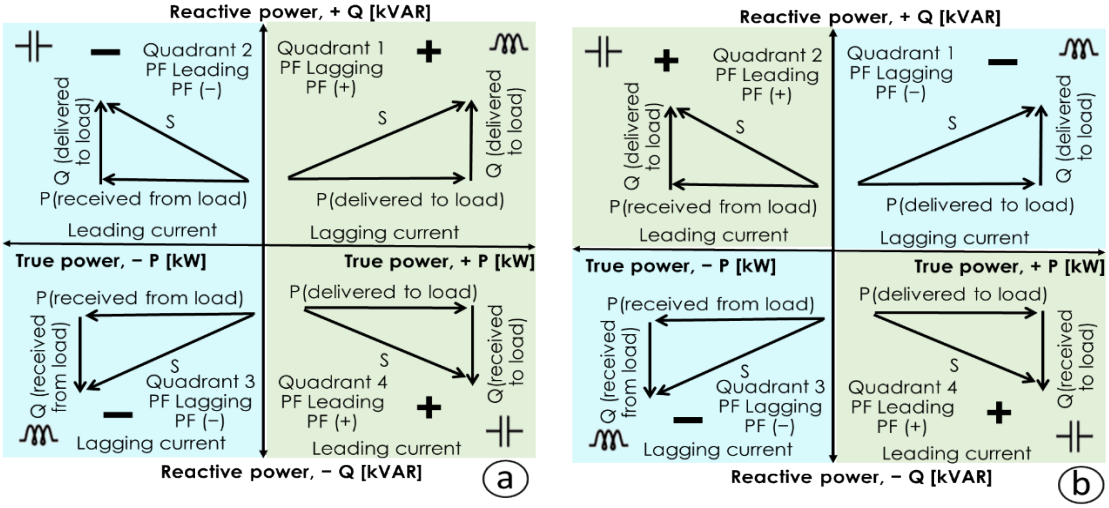


Figure 7. Signs of power factor based on (a) IEC and (b) IEEE standards.

For the IEC standard (Figure 7a), the positive and negative power factors are when the true power is positive and negative, respectively. In the IEEE standard (Figure 7b), the positive and negative power factors are for leading current (capacitive effect) and lagging current (inductive effect), respectively. For the IEC standard [28] in Figure 7a, quadrants 1 and 4 with positive true power represent a positive power factor. But quadrants 2 and 3 with negative true power represent a negative power factor. In the IEEE standard [28] in Figure 7b, quadrants 2 and 4 with capacitive load (power factor leading) represent a positive power factor. But quadrants 1 and 3 with inductive load (power factor lagging) represent a negative power factor. In Figure 7, the green and blue quadrants are the positive and negative power factors, respectively. In this study, the TPF SC with CGGS using DLT for the electrical utility grid with customer-owned wind farm was used with an SEL 700GT relay [27] that uses the power factor sign convention, based on the IEEE standard in Figure 7b.

3.2 VOLTAGE SERVICE LIMIT SMART CONTRACT

In the VSL SC, the voltage service limits were selected based on the grid location for the SEL 700GT relay in the POI between the main grid side and DER side, the nominal voltage on the relay's grid side, and considering the required voltage limits for a service voltage limit greater than 600 V for the ANSI C84.1 standard.

3.2.1 Voltage Service Limits

The voltage boundaries were selected to define a good voltage service limit application. The flow diagram in Figure 8 defines the boundaries for the VSL algorithm. The voltage limits were calculated based on the ANSI C84.1 standard [34]. The steps to select the voltage limits are presented in Figure 8a–Figure 8f. In the first step (Figure 8a), a relay is selected on the grid (SEL 700GT relay in the POI). Then, the relay nominal voltage on the grid side (7,200 V) is greater than 600 V (Figure 8b), and the voltage limits are selected based on the ANSI C84.1 standard (Figure 8c). Because the relay is not located on a user load site (Figure 8d), the SEL 700GT relay is in the POI. The voltage service limit for a more than 600 V option is selected (Figure 8e), based on the ANSI C84.1 standard [34]. Then, the ANSI C84.1 service voltage limits were selected for range B (Figure 8f). In this case, the voltage limits were set between 95% and 105.8% of the nominal limits. Therefore, the undervoltage limit was 6,840 V, and the overvoltage limit was 7,620 V.

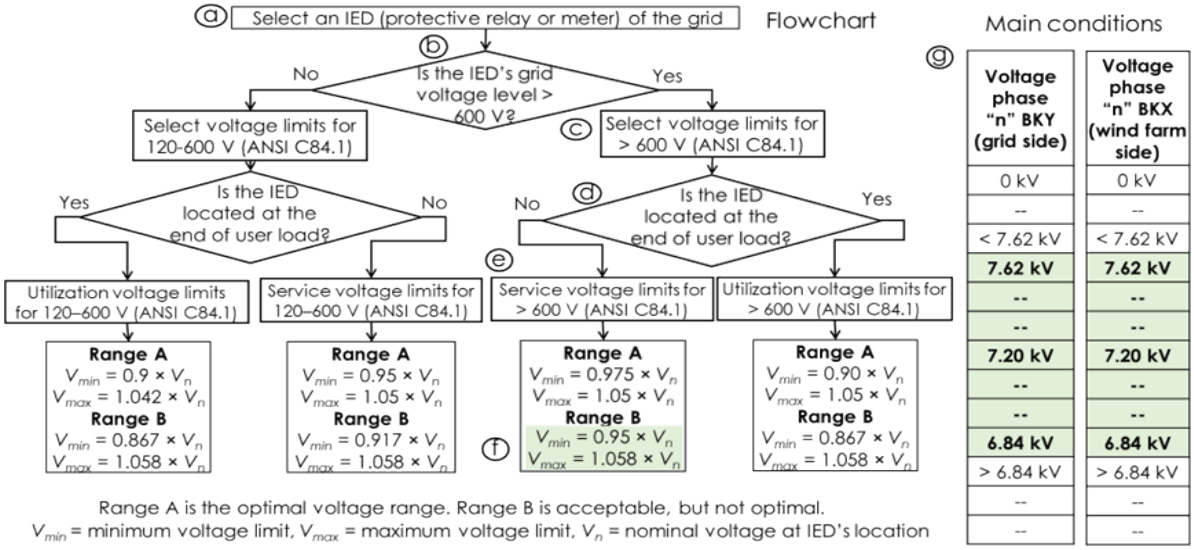


Figure 8. (a–f) Flowchart of voltage ranges of the ANSI C84.1 standard and (g) voltage service limits for SC.

The flowchart selects the voltage service limits based on the ANSI C84.1 standard (Figure 8a–Figure 8f), and Figure 8g shows the range of the selected voltage service limits for the VSL SC, for the breaker BKY (grid side), and breaker BKX (wind farm side).From the flowchart in Figure 8a–Figure 8f, considering the SEL 700GT relay is on a 7,200 kV grid at the POI between the grid side and wind farm side, the IED's grid voltage level is greater than 600 V, and the IED is not located at the end of the user load. Therefore, the Range B for the service voltage limits greater than 600 V is selected for the generic "*n*" phases (A, B, and C) given by Eq. (5):

$$V_n \times 0.95 \leq VSL_n \leq V_n \times 1.058, \text{ (Range B for service voltage limits } > 600 \text{ V, ANSI C84.1)} \#(5)$$

where VSL_n is the voltage service limit for the generic "*n*" phase (A, B, and C) in volts total power factor, and V_n is the nominal voltage in volts.

Because the SEL 700GT relay is on a 7,200 kV grid at the POI, placing 7,200 V as the nominal voltage on (5), the voltage service limits for the SEL 700GT relay in the POI between the grid side and wind farm side are represented by Eq. (6):

$$6{,}840\ volts \leq VSL_n \leq 7{,}620\ volts. \text{ (Range B for service voltage limits } > 600 \text{ V, ANSI C84.1)} \#(6)$$

In the VSL SC, the condition from Eq. (6) represents the main boundary for the measured phases A, B, and C voltages of the SEL 700GT relay in the POI on the grid-side breaker BKY (Figure 1a), considering this voltage range a priority to keep the voltage service limits on the load feeders.

4. METHODOLOGY

4.1 TOTAL POWER FACTOR SMART CONTRACT

The TPF SC defines the permitted TPF from the wind farm side to feed the main loads as well as the terms of the SC to be implemented with the CGGS using DLT for a customer-owned DER (wind farm). The TPF SC was implemented by the CGGS using an SEL 700GT relay located in POI. In this study, the permitted maximum and minimum TPF limits on the grid side were defined between +1.0 and +0.9, respectively. The operation of the breakers in the POI between the power grid and DER sides was controlled by the SEL 700GT relay's measured data with the instructions of the TPF SC.

4.1.1 Total Power Factor Smart Contract

The SC is defined as a digital contract stored on a distributed ledger that is automatically performed when the predetermined terms with conditions are reached. The SC is commonly used to automate the execution of transactions consistent with set rules or according to specific conditions so that all members (electrical utilities) can operate in consensus. The TPF SC was applied to a CGGS with DLT to manage the relay in the POI between the electrical utility grid and customer-owned wind farm, thereby maintaining the integrity of data from a relay at the POI. In addition, TPF SC becomes more difficult as the number of DERs and POI increases. However, DLT could enhance the resilience of modern power grids by effectively securing shared data. The TPF SC was based on using CGGS with DLT and an SEL 700GT relay that measures the TPF at breaker BKY (grid side) and breaker BKX (wind farm side) for the three phases in the POI. The TPF SC application was performed using a real-time simulator with the CGGS and the SEL 700GT relay-in-the-loop. In this study, the main achievement of the TPF SC is to implement the CGGS with DLT to improve power quality at the POIs measuring the TPF at the grid side

that could be connected to a customer-owned DER (wind turbine farm). The CGGS collected data from the SEL 700GT relay used for performing the TPF SC. The permitted maximum and minimum TPF limits on the grid side were set at between +1.0 and +0.9, respectively. The operation of the breakers in the POI was controlled by the CGGS using DLT and the SEL 700GT relay, based on the terms and conditions for the SC. The limits of the TPF SC were defined by the breaker states (BKY and BKX) of the grid side and wind farm side; the TPF values at the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) were measured by keeping the measured TPF on the grid side between +0.9 and +1.0 for the loads on the main feeders.

The TPF SC had another boundary condition based on the period for keeping the TPF from the CGGS. When the measured TPF on the grid side (TPF_{BKY}) was smaller than +0.9 and this condition was measured for more than 400 s, the circuit breakers in the POI controlled by the CGGS and SEL 700GT relay were operated by the TPF SC. In this situation, a time window of 450 s was defined for the TPF SC, and this time will depend on the implementation of the TPF SC for normal operation and electrical fault situations. In normal operation, this time window permits the operation of circuit breakers in the POI by the TPF SC after use of other power factor quality techniques (e.g., capacitor banks, load shedding, transformer/load tap changers) from the grid-side utility, without use of the TPF SC. A common method in electrical distribution substations is to use load tap changers. The load tap changer method can raise the voltage magnitudes but cannot improve the reactive power or power factor [46]. The capacitor banks method is usually selected against overload tap changers at the electrical distribution substations. However, the feeding of industrial loads is achieved with a proper time delay at both controllers. The capacitor bank controller is applied on the high voltage side, and the load tap changers controller is used in the low voltage side of the transformer [46]. In the transformers, the onload tap changers can regulate voltages in steps of 0.625% to 2.5% with a time delay of 1–3 min (60–180 s) for each step operation [46], but switching on a capacitor bank can give the same or larger voltage increase more quickly [46]. Another method for improving the power quality is load shedding, which is usually applied if there is a shortage of electricity supply or to avoid power lines being overloaded by poor power factors. A load shedding program is successfully implemented when the system frequency is recovered to 60 Hz, but this process can take several minutes [47] or hours, depending on the power grid.

The measured TPF can be up to +1.0, and a good TPF was considered between +0.9 and +1.00. In the TPF SC, the time window was set greater than 400 s (6.66 min). The TPF SC condition must not operate the breakers in the POI for a poor measured TPF of less than +0.9 from the grid side ($TPF_{BKY} < +0.9$) during anomaly events such as electrical fault states. The poor measured TPF could be detected during an electrical fault state for short periods of less than 1 s, and the protective relays first need to clear the electrical fault by opening the breakers before the CGGS triggers the low-TPF SC condition. In addition, the time window greater than 400 s was selected based on the maximum time setting limit for the frequency protection elements, defined as 400 s for frequency protection elements [48], based on the relay manual [27] (Table 2).

Table 2. Time range of frequency protection elements [1].

Protection element (number)	Protection function description	Delay time setting	Delay time range (s)
Frequency of X side (81)	81: Frequency protection of breaker BKX side	Frequency delay time	0.00–400.00
Frequency of Y side (81)	81: Frequency protection of breaker BKY side	Frequency delay time	0.00–400.00
V/Hz (24)	24: Overexcitation protection, ratio of the voltage to frequency	Reset time	0.00–400.00

In the SEL 700GT relay, the conventional sign for the power factor is based on the IEEE standard. The conventional sign of the power factor for the IEEE standard is based on when the current is lagging behind the voltage (Figure 9a) or the power factor is lagging (inductive behavior); this is when the power factor has a negative sign. But when the current is leading the voltage (Figure 9b) or the power factor is leading (capacitive behavior), the power factor has a positive sign. Figure 9 presents the phasor diagrams (Figure 9a and Figure 9b) and IEEE standard power factor sign convention (Figure 9c) for the SEL 700GT relay.

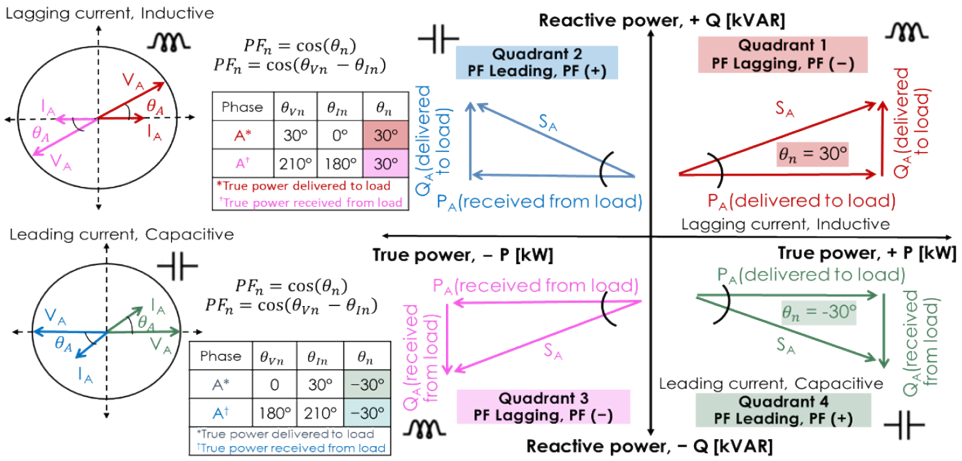


Figure 9. (a and b) Phasor diagrams and (c) IEEE standard power factor signs convention [1].

In the SEL 700GT relay, the power factor at phases in the breakers at the POI for the grid side and the wind farm side were calculated as the cosine of the angle between the phase voltage and phase current by Eq. (7):

$$PF_{mBKn} = \cos(\theta_{VmBKn} - \theta_{ImBKn}) \,, \tag{7}$$

where PF_{mBKn} is the power factor of generic phase m (A, B, or C) for the breaker n (BKY or BKX) measured at ±1.00; Θ_{VmBKn} is the voltage angle of the generic phase m (A, B, or C) for the circuit breaker n (BKY or BKX) measured in degrees; and Θ_{ImBKn} is the current angle of the generic phase m (A, B, or C) for the circuit breaker n (BKY or BKX) measured in degrees.

In the SEL 700GT relay, the TPF or three-phase power factor for the grid side (TPF_{BKY}) and the wind farm side (TPF_{BKX}) depends on the true and apparent power of phases A, B, and C; it was calculated by Eq. (8):

$$TPF_{BKn} = \frac{PA_{BKn} + PB_{BKn} + PC_{BKn}}{SA_{BKn} + SB_{BKn} + SC_{BKn}} \,, \tag{8}$$

where TPF_{BKn} is the total power factor for the n breakers (BKY or BKX); PA_{BKn}, PB_{BKn}, and PC_{BKn} are the true powers of the A, B, and C phases, respectively, for the n breaker (BKY or BKX) in kW; and SA_{BKn}, SB_{BKn}, and SC_{BKn} are the apparent powers of the A, B, and C phases, respectively, for the n breaker (BKY or BKX) in kVA.

In the POI, the SEL 700GT relay was connected to three-phase pole breakers (BKY or BKX) that measure a TPF of +1.00 when they were opened. This condition is taken to avoid indetermined TPF measurements when the breakers (BKY or BKX) are opened (to avoid a theoretical indetermined measuring error by Eq. [9]), and the phase currents are not flowing there. The situation is represented by

the generic expression of the TPF definition in Eq. (9). Basically, when the phase currents are zero, the total true and apparent powers also are zero; consequently, the TPF result is indetermined, and the SEL 700GT relay measures a TPF value of +1.00.

$$If\ TPF = \frac{Total\ true\ power}{Total\ apparent\ power} = \frac{0.00}{0.00} = \nexists \rightarrow Then\ \text{measured TPF} = +1.00\ . \quad (9)$$

This open breaker TPP measurement condition is set by the manufacturer. The measurement of a TPF is +1.00 when a breaker is opened; this condition is defined by the manufacturer to avoid a measurement of an indetermined (swing) TPF. Therefore, the grid-side (BKY) and wind farm side (BKX) breakers in the POI should not be operated. The measured TPF from the real-time simulation versus the SEL 700GT relay are shown in Figure 10. From the real-time simulator (Figure 10a), the measured TPF swings when the breaker BKX is opened. But in the CGGS (Figure 10b), the measured TPF from the SEL 700GT relay is +1.00, validating the condition by Eq. (9).

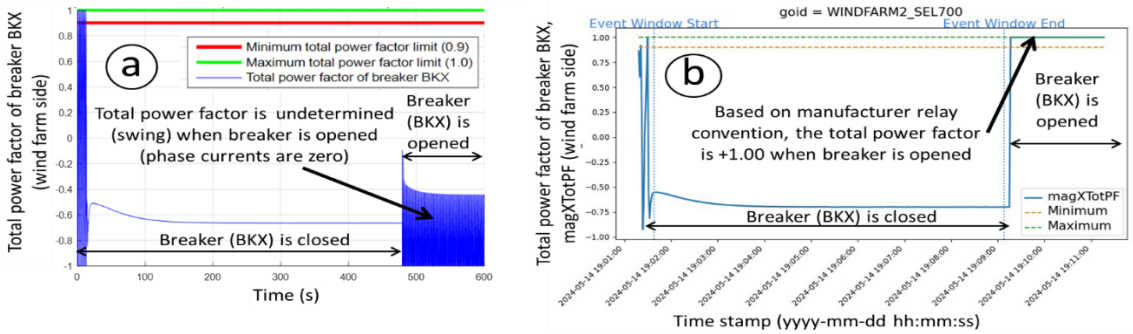


Figure 10. Measured TPF from (a) the real-time simulator and (b) CGGS [1].

The main goal of the TPF SC is to reduce the reactive power circulating on the power line at the grid side when the main electrical utility grid was connected to the customer-owned DER (wind farm); another objective is to keep the data security and reliability from CGGS using DLT and SCs. The power line losses generated by the reactive power need to be kept with a good TPF of +1.00 to +0.90. Figure 11 shows the TPF SC flowchart for the time window operation.

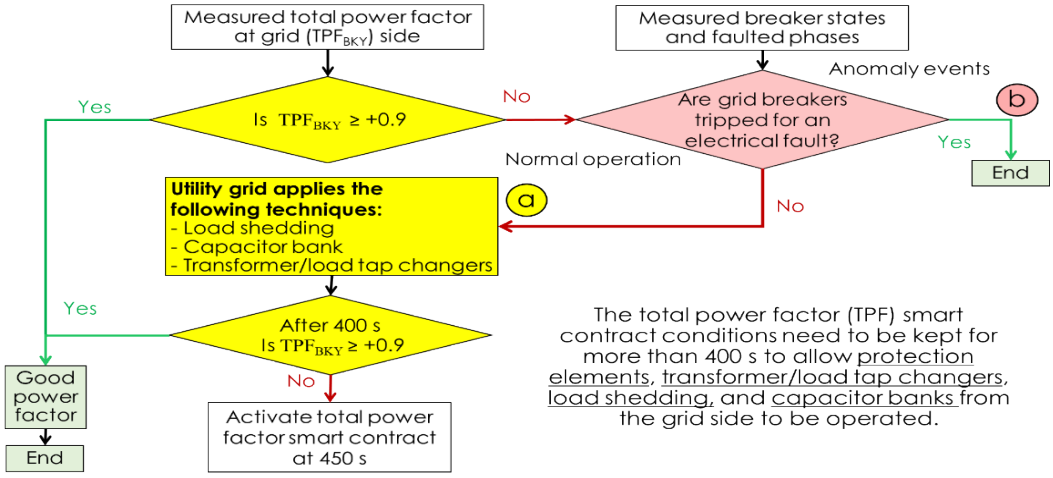


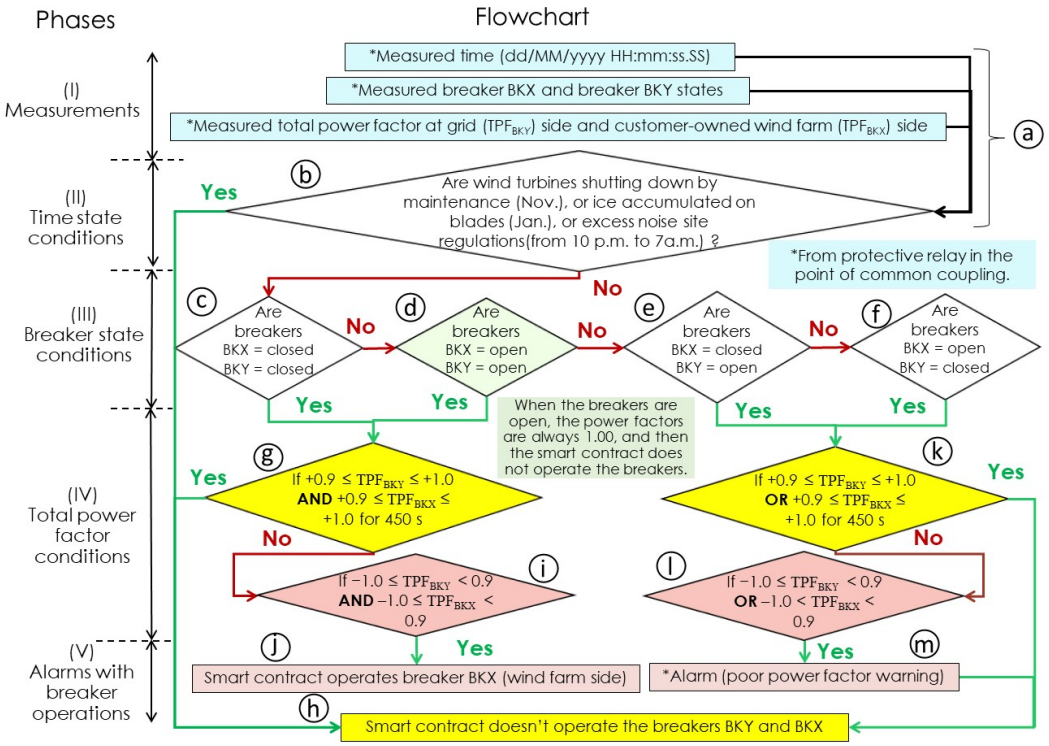
Figure 11. Flowchart of the time window for the TPF SC (a) normal operation and (b) anomaly events.

4.1.2 Flowchart of Time Window for Total Power Factor Smart Contract

The time window to set the TPF SC was defined at 450 s; it was selected to perform normal electrical grid operation. This time window permitted the BKX and BKY breakers in the POI to work under the TPF SC after following the traditional power factor technique applications (e.g., capacitor banks, load shedding, transformer/load tap changers) in the grid side of the main electrical utility (Figure 11a). The time window of 450 s is also defined to not operate the TPF SC below +0.9 for transient anomaly events, which in some cases could have poor TPF value for only a couple of cycles, like electrical faults that were cleared by nearby relays (Figure 11b). In the TPF SC, the conditions for the measured TPF values were held for 450 s to permit the protection elements, transformer/load tap changers, load shedding, and capacitor banks on the grid side to be assessed. This occurred before the BKY/BKX breakers in the POI were operated with the CGGS.

4.1.3 Flowchart of Total Power Factor Smart Contract

In the flowchart of Figure 12, the A, B, and C phases were measured and collected for evaluation with the main conditions of the TPF SC. The flowchart of the TPF SC was defined for the breaker non-operation condition when a good TPF was measured (between +1.00 and +0.90) and for the breaker operation condition when a poor TPF was measured (smaller than +0.90). The definitions for the good and poor power factors limited the TPF boundaries considering a time window of 450 s for the TPF SC. Figure 12 shows the flowchart of the TPF SC created to measure the power factor behavior of the SEL 700GT relay from Eq. (9) and the PF3X/PF3Y three-phase power factors on the POI measured from the SEL 700GT relay using the CGGS with IEC 61850 GOOSE messages. The phases of the TPS SC flowchart (Figure 12) were defined by the (I) measurements, (II) time state conditions, (III) breaker state conditions, (IV) total power factor conditions, and (V) alarms with breaker operations.



TPF BKY (Grid side)	TPF BKX (Wind farm side)
+1.0	+1.0
+0.9	+0.9
+0.8	+0.8
+0.7	+0.7
+0.6	+0.6
+0.5	+0.5
+0.4	+0.4
+0.3	+0.3
+0.2	+0.2
+0.1	+0.1
0	0
−0.1	−0.1
−0.2	−0.2
-0.3	-0.3
−0.4	−0.4
−0.5	−0.5
−0.6	−0.6
−0.7	−0.7
−0.8	−0.8
−0.9	−0.9
−1.0	−1.0

Figure 12. Phases, flowchart, and main conditions of TPF SC [1].

In phase (I) of the flowchart (part Figure 12a), the time states (day, month, year, minutes, and seconds) and TPFs of breakers in the POI are measured. In phase (II) of the flowchart (part Figure 12b), the time state conditions are assessed for the TPF SC. This contract must consider periods when disturbances (e.g., those related to wind farm low-power-quality scenarios and shut-down situations) could affect the utility grid side. The scenarios for maintenance operations (November), possible extreme ice accumulation on the blades (January), and excess noise site regulations (from 10 p.m. to 7 a.m.) are programmed. In phase (III) of the flowchart (Figure 12c–Figure 12f), the breaker state conditions in the POI are assessed. The breaker state situations were defined for the closed and open events of the breakers (BKY/BKX) in the POI on the grid side and wind farm side. In phase (IV) of the flowchart (Figure 12g, 12i, 12k, and 12l), the AND/OR logic is assessed for the TPF conditions of the breakers (BKY/BKX) in the POI on the grid side and wind farm side. In phase (V) of the flowchart (Figure 12j, 12m, and 12h), the TPF SC operates the wind farm side breaker (BKX) in the POI if the measured TPF on the grid side is smaller than +0.90 (TPF_{BKY}). A warning alarm is activated if the TPF SC does not operate the BKX (wind farm side) and BKY (grid side) breakers in the POI.

The main objective of the TPF SC flowchart (Figure 12) is to prioritize operation for the wind farm side (BKX) breaker instead of for the grid side (BKY) breaker. The reasons for this are because the BKY breaker (grid side) feeds the main loads (Figure 1a), keeping a resistive nature (between +0.90 and +1.00) of the TPF at the grid-side is necessary, and the main grid utility does not allow its BKY breaker to be controlled by the TPF SC in the CGGS. Also, the TPF SC does not operate the BKY and BKX breakers for a good TPF (between +0.90 and +1.00). Therefore, the next conditional sequences are defined according to the following parameters (Figure 12):

- When both breakers in the POI are closed (Figure 12c), and the measured TPF on the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) are between +0.90 and +1.00 (Figure 12g) for 450 s, the SC doesn't operate any breaker (Figure 12h). However, if the measured TPFs on the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) are smaller than +0.90 (Figure 12i) after 450 s, the TPF SC operates the wind farm side BKX breaker (Figure 12j).

- When both breakers are open in the POI (Figure 12d) and the measured TPF on the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) are +1.00—because of the SEL 700GT relay condition from Eq. (9) if breakers in the POI are opened—the TPF SC doesn't operate any breakers in the POI (Figure 12h).

- When only one breaker is closed in the POI (Figure 12e and Figure 12f) and the measured TPF on the grid side (TPF_{BKY}) or wind farm side (TPF_{BKX}) is between +0.90 and +1.00 (Figure 12k) for 450 s, the SC doesn't operate any breaker (Figure 12h). However, if the measured TPF on the grid side (TPF_{BKY}) or wind farm side (TPF_{BKX}) is smaller than +0.90 (Figure 12l) after 450 s, a warning alarm is triggered (Figure 12m) to indicate a poor power factor without operating the breakers in the POI.

4.2 SMART CONTRACT OF VOLTAGE SERVICE LIMITS

The VSL SC defines the allowable phase voltage magnitudes of the DER output. The conditions and terms of the VSL SC are performed by using CGGS with DLT for a customer-owned DER (wind farm). The VSL SC controlled by the CGGS using DLT collected the data from the SEL 700GT relay to execute the VSL SC step by step. The main constraints were defined based on ANSI C84.1 [34] service voltage limits, and the operation of the circuit breakers in the power grid and DER sides was controlled by the relay consistent with the provisions of the SC.

4.2.1 Smart Contract of Voltage Service Limit

The SC is described as a stored digital program on a distributed ledger that is performed automatically if the terms and conditions are reached. The SC is performed to automate execution of a transaction list that must agree with rules set according to specific conditions; all members (e.g., electrical utilities) in the SC can then manage implementation according to the consensus of all participants. The VSL SC was applied to a CGGS with DLT to operate the conditions in the POI for an electrical utility grid with a customer-owned wind farm. The VSL SC is focused on maintaining data integrity from the SEL 700GT relay in the POI. The VSL SC was based on using CGGS with DLT and an SEL 700GT relay in the POI that measures phase (*n*) voltages magnitudes at the BKY breaker (grid side) and BKX breaker (wind farm side) for the phases A, B, and C. The application of the VSL SC was made by using a real-time simulator with a CGGS and an SEL 700GT relay in-the-loop.

In this study, the main achievement of the VSL SC is to implement the CGGS with DLT to improve power quality at the POIs measuring phase voltages at the grid side that could be connected to a customer-owned DER (wind turbine farm). The CGGS collected data from the SEL 700GT relay used for performing the VSL SC. The permitted maximum and minimum phase voltage magnitude limits on the grid side were set at between 7,620 and 6,840 V, respectively. Operation of the breakers in the POI was controlled by the CGGS using DLT and the SEL 700GT relay, based on terms and conditions for the SC. Limits of the VSL SC were defined by the breaker states (BKY and BKX) of the grid side and wind farm side. The phase voltage magnitudes at the grid side and wind farm side were measured by keeping the measured phase voltages on the grid side between 7,620 and 6,840 V on the main feeders.

The VSL SC had another boundary condition based on the period for keeping the phase voltage magnitudes from the CGGS. When the measured phase voltage magnitudes on the grid side (*VnBKY*) are not between 6,840 and 7,620 V—and this condition is measured for more than 400 s—the breakers in the POI controlled by the CGGS and SEL 700GT relay are operated by the VSL SC. In this situation, a time window of 450 s was defined for the VSL SC, and this time depends on implementation of the VSL SC for normal operation and electrical fault situations. In normal operation, this time window permits operation of the breakers in the POI by the VSL SC after the use of other power factor quality techniques (e.g., capacitor banks, load shedding, transformer/load tap changers) from the grid-side utility, without use of the VSL SC. A common method in electrical distribution substations is to use load tap changers. This method can raise voltage magnitudes but cannot improve reactive or power factors [46]. Therefore, the capacitor banks method is usually selected against overload tap changers at the electrical distribution substations. However, the feeding of industrial loads is achieved with a proper time delay at both controllers. The capacitor bank controller is applied on the high voltage side, and the load tap changers controller is used on the low voltage side of the transformer [46]. In transformers, the onload tap changers can regulate voltages in steps of 0.625%–2.5% with a time delay of 1–3 min (60–180 s) for each step operation [46]. However, switching on a capacitor bank can give the same or a larger voltage increase much faster [46]. Another method for improving power quality is load shedding, which is usually applied in case of a shortage of electricity supply or to prevent power lines being overloaded by poor power factors. A load shedding program is successfully implemented when the system frequency is recovered to 60 Hz, but that can take several minutes [47] or hours, depending on the power grid.

4.2.2 Flowchart of Time Window for Voltage Service Limits Smart Contract

The time window to set the VSL SC was defined at 450 s; it was selected to perform normal electrical grid operation. This time window enabled the BKX and BKY breakers in the POI to work under the VSL SC after following the traditional power factor technique applications (e.g., capacitor banks, load shedding, transformer/load tap changers) in the grid side of the main electrical utility (Figure 13a). The time window of 450 s is also defined to not operate the VSL SC for transient anomaly events. Some cases

could have phase voltage magnitudes outside 6,840 and 7,620 V for only a couple of cycles, such as electrical faults that were cleared by nearby relays (Figure 13b). In the VSL SC, the conditions for the measured phase voltage magnitudes were held for a period of 450 s to permit the protection elements, transformer/load tap changers, load shedding, and capacitor banks on the grid side to be assessed. After this step, the BKY/BKX breakers in the POI will be operated with the CGGS.

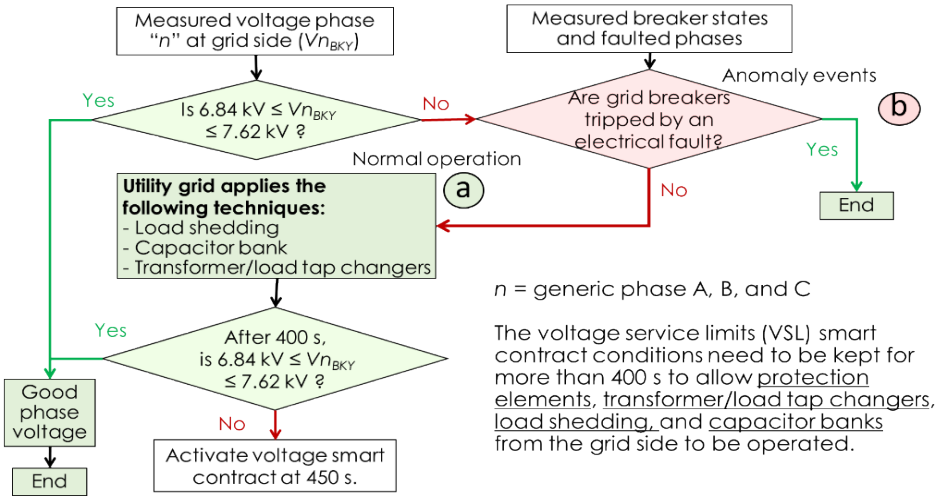


Figure 13. Flowchart of the time window for the VSL SC (a) normal operation and (b) anomaly events.

4.2.3 Voltage Service Limits Smart Contract Flowchart

The phases and main conditions of the VSL SC are shown in Figure 14. The VSL SC was defined by the operation and nonoperation breaker conditions for the maximum and minimum voltage limits of 7,620 and 6840 V, respectively, by using a time window of 450 s. The VSL SC flowchart (Figure 14) was based on measuring the phase voltage magnitudes in the POI with the SEL 700GT relay and CGG. The measured phase voltage magnitudes for phases A, B, and C at the grid side and wind farm side were collected from the SEL 700GT relay using the CGGS with IEC 61850 GOOSE messages. In Figure 14, the phases of the VSL SC flowchart for an *n* generic phase are presented and defined by the (I) measurements, (II) time state conditions, (III) breaker state conditions, (IV) phase voltage conditions, and (V) alarms with breaker operations. In phase (I) of the VSL SC flowchart (Figure 14a), the time (day, month, year, minutes, and seconds), pole states, and phase (*n*) voltages of the breakers (BKY and BKX) in the POI are measured on the grid side and wind farm side. In phase (II) of the VSL SC flowchart (Figure 14b), the time state conditions are assessed at 450 s. This VSL SC must consider periods when disturbances related to wind farm low- or high-voltage scenarios could be available and when possible shut-down situations could affect the utility grid-side. These scenarios for the VSL SC were programmed routines like maintenance and operation tasks (November), possible extreme ice accumulation on the blades (January), and excess noise site regulations (from 10 p.m. to 7 a.m.). In Figure 14, the time stamps are measured in the *dd/MM/yyyy HH:mm:ss.SS* format from the SEL 700GT relay to activate the VSL SC (Figure 14b). In phase (III) of the flowchart (Figure 14c–Figure 14f), the breaker state conditions are assessed. The breaker state situations were defined at possible close and open events of the breakers in POI, located on the wind farm side (BKX) and grid side (BKY) for the conditions of the VSL SC flowchart. In phase (IV) of the VSL SC flowchart (Figure 14g–Figure14n), the AND/OR logic phase *n* voltage conditions for the wind farm side (BKX) and grid-side (BKY) breakers are assessed. In phase (V) of the VSL SC flowchart (Figure 14o), the VSL SC does not operate the breakers in the POI of the SEL 700GT relay on the wind farm side (BKX) and grid side (BKY) and perform a warning alarm.

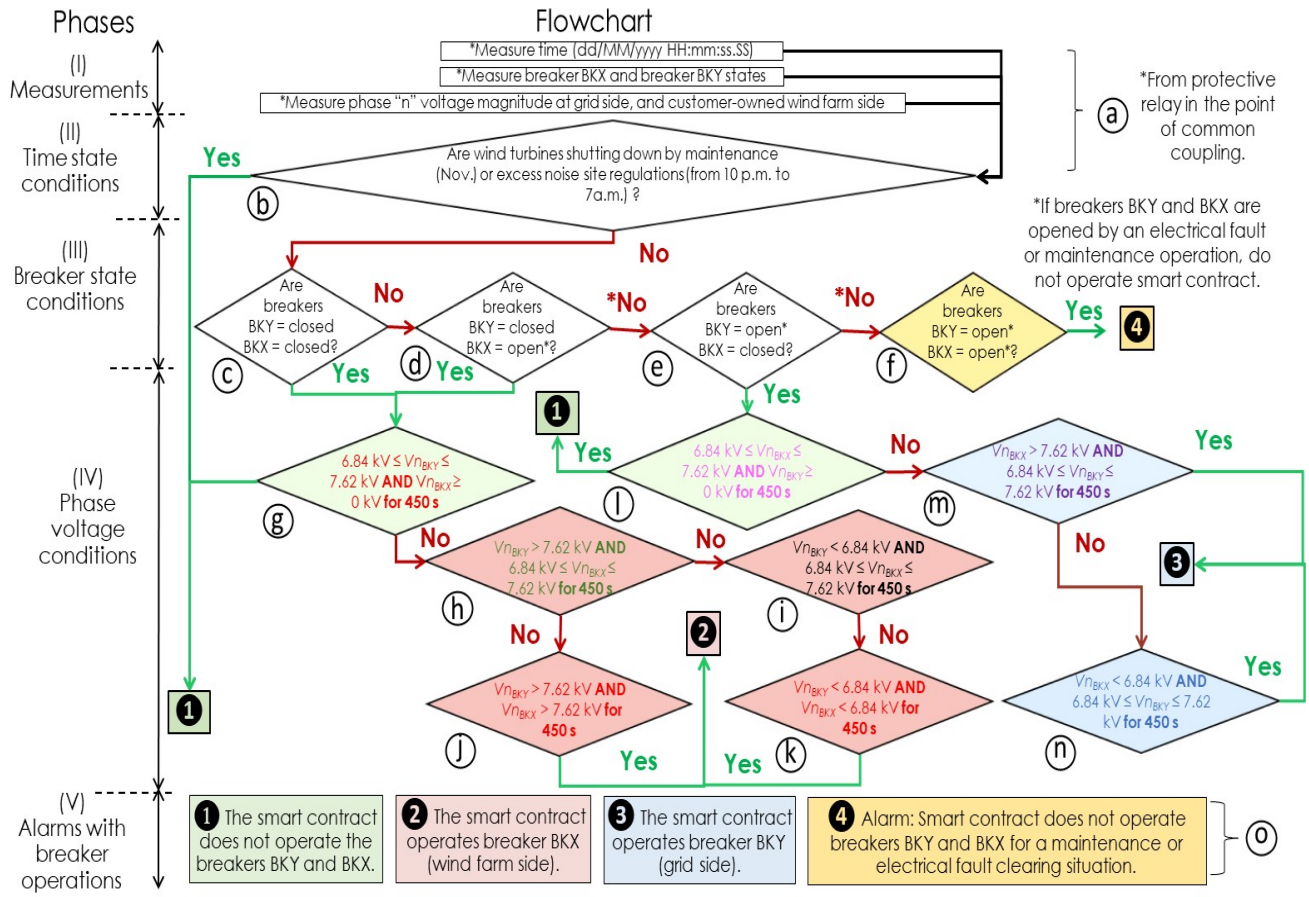


Figure 14. Phases, flowchart, and main conditions of the VSL SC.

5. CYBER GRID GUARD SYSTEM

5.1 CYBER GRID GUARD SYSTEM AND DLT SMART CONTRACTS

The CGGS is the main platform to perform TPF and VSL SCs in the POI between the electrical utility grid-side and customer-owned utility wind farm side. This section presents the CGGS and definitions, architecture of the TPF and VSL SCs, and the boundaries for the TPF and VSL SCs. The conditions for the TPF and VSL SCs were shown in tables that present the TPF values and phase voltage magnitudes for the different breaker states in the POI between the grid side and wind farm side.

5.1.1 Cyber Grid Guard and Definitions

In this study, the CGGS is represented by a modular software platform designed for handling data ingestion and performing tasks related to securing the protective relays/power meters' data and configuration settings. The CGGS is responsible for ingesting data into the storage layers, consisting of data stores formed by on-chain (DLT) and off-chain (SQL) databases. The CGGS then uses these data to enable anomaly detection. At a high level, the modules are configured around specific tasks that include data ingestion, processing, and anomaly detection. The data ingestion is based on collecting data and configuration settings from the protective relays/power meters by the network applying protocols such as

IEC 61850 GOOSE and FTP. The data are stored in on-chain databases (DLT) by sending the transactions as SC functions; the data are stored in the off-chain database (PostgreSQL) with the TimescaleDB extension. In this process, the data steps are included to convert the data from wire formats to JSON for more convenient storage and parsing. Anomaly detection is performed by using methods based on analyzing generated statistics and looking for comparisons between the on-chain and off-chain data to detect anomalous events from the electrical grid. The definitions and terms related to DLT are shown in Table 3.

Table 3. Terms and definitions related to distributed ledger technology.

Term	Definition
CGGS: Cyber Grid Guard System	The CGGS is given by a modular software framework that generates data integrity attestation with a storage layer formed of a DLT and database.
DLT: distributed ledger technology	The DLT is a distributed and decentralized secure database that uses consensus mechanisms to reach agreement on the states. The most common types of DLT are based on blockchain technology.
SCs: smart contracts	The SCs are programs that run on top of the DLTs and enforce the conditions and terms on the data transaction executed in a distributed manner.
On-chain	On-chain refers to the functionality or data in the distributed ledger.
Off-chain	Off-chain refers to the functionality or data outside of the distributed ledger (e.g., in a traditional database).

5.1.2 Smart Contract Architecture

The TPF and VSL SCs were implemented with the CGGS using DLT and the SEL 700GT relay in the POI. The SCs were focused on checking the TPF/VSL conditions in the POI, for the BKY breaker (grid side) and BKX breaker (wind farm side).

The SCs were created using the Hyperledger Fabric (HLF) 2.5, which represents the latest long-term support release of the open-source software with an enterprise-grade blockchain platform. This HLF was a modular, permissioned DLT platform designed for enterprise-grade, general-purpose use test cases. The SCs were defined as self-executing contracts in which the terms of the contracts were defined in the programmed Go codes because the CGG software is implemented with Python and the smart contracts are implemented with Go. The SCs operated on DLT platforms, enabling execution and automatic enforcement without the need for intermediaries. In the HLF DLT platform, the SC programs were referred to as “chaincode” and implemented in various programming languages. In this research, the Go language was used to implement the TPF/VSL chain code. The TPF/VSL chain code was implemented to store the IEC 61850 GOOSE-based measurements from the SEL 700GT relay in the POI, including the states. TPF/VSL values are presented in Table 4.

Table 4. Fields of the GOOSE datasets in the event checks.

Field name	Data type	Data description
breakerXOpen	Boolean	Breaker X status (open = 1, closed = 0)
breakerYOpen	Boolean	Breaker Y status (open = 1, closed = 0)
magXTotPF	Float32	Breaker X total power factor
magYTotPF	Float32	Breaker Y total power factor
MagXVoltagePhaseA	Float64	Phase A voltage magnitude for breaker X
MagXVoltagePhaseB	Float64	Phase B voltage magnitude for breaker X
MagXVoltagePhaseC	Float64	Phase C voltage magnitude for breaker X

Field name	Data type	Data description
MagYVoltagePhaseA	Float64	Phase A voltage magnitude for breaker Y
MagYVoltagePhaseB	Float64	Phase B voltage magnitude for breaker Y
magYVoltagePhaseC	Float64	Phase C voltage magnitude for breaker Y

Figure 15 shows the architecture and software of the CGGS. In this case, the SEL 700GT relay sends the IEC 61850 GOOSE measurements from Table 4, which are published by the WINDFARM2_SEL700 relay and received by the observer program. In this process, the program uses the *libiec61850* library to receive the GOOSE messages and process the data of the SEL 700GT relay in the POI, which includes filtering duplicate messages and formatting them as JSON. The TPF/VSL ledger storage module handles receiving the formatted GOOSE messages of the SEL 700GT relay, and then the DLT transactions are created and sent with the relevant data implemented by the TPF/VSL chaincode AddOrUpdateRecord function.

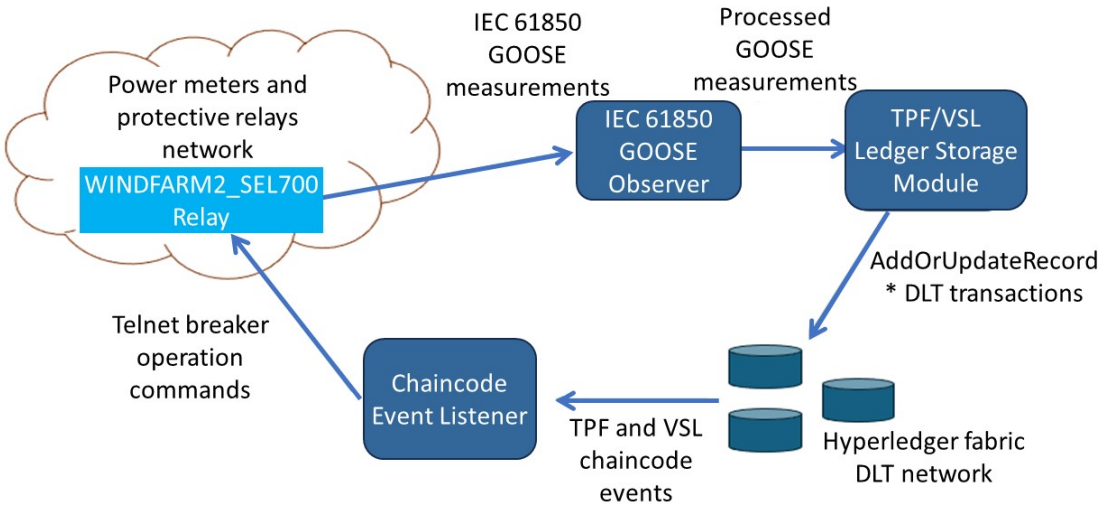


Figure 15. Architecture of the DLT with SC condition chaincode.

5.1.3 Conditions for Total Power Factor Smart Contract

In the CGGS, the conditions for the TPF SC are created based on the flowcharts in Figure 11 and

Figure 12. The TPF SC uses a backup power quality method that could be performed after using the capacitor bank applications, load shedding, and transformer/load tap changers on the grid side. Next, the TPF condition and operation of the BKX breaker (wind farm side) were implemented at a time of 450 s. The TPF conditions depend on the time, states of breaker, and TPF measured in the POI, as shown in Figure 16. If a condition is met for the detailed logic shown in Figure 16, the DLT event is generated. The event is then emitted as a JSON message that is received by the listener application to manage the operation of the BKX breaker (wind farm side). The listener application was generated as a Go program with the HLF software that enables receiving the event from the SC and performing the communication between the SEL 700GT relay and CGGS for operating the BKX breaker (wind farm side) in the POI. This design was made because the SCs are unable to directly interact with external resources, and they

need to be executed in a distributed manner to reach a result that is consistent and deterministic in the CGGS.

Phase	*Measured smart contract conditions (IF)*			*Smart contract decisions (THEN)*
	BKY (grid side)	*BKX (wind farm side)*	*Total power factor for a period of 450 s*	
No breaker operation	closed	closed	+0.90 ≤ TPF BKY ≤ +1.00 **AND** +0.90 ≤ TPF BKX ≤ +1.00	**Do not operate breakers BKY and BKX.**
	closed	open	+0.90 ≤ TPF BKY ≤ +1.00 **OR** +0.90 ≤ TPF BKX ≤ +1.00	
	open	closed		
	open	open	When breakers are opened, then the measured power factor is 1.00; therefore, breakers must not be operated by the smart contract	
Breaker operation	closed	closed	-1.00 ≤ TPF BKY < +0.90 **AND** -1.00 ≤ TPF BKX < +0.90	**Operate breaker BKX.**
	closed	open	-1.00 ≤ TPF BKY < 0.90 **OR** – 1.00 ≤ TPF BKX < 0.90	***Alarm (poor power factor warning).**
	open	closed		
*Do not operate breakers BKY (grid side) and BKX (wind farm side).				

a

TPF BKY (grid side)	TPF BKX (wind farm side)
+1.0	+1.0
+0.9	+0.9
+0.8	+0.8
+0.7	+0.7
+0.6	+0.6
+0.5	+0.5
+0.4	+0.4
+0.3	+0.3
+0.2	+0.2
+0.1	+0.1
0	0
−0.1	−0.1
−0.2	−0.2
−0.3	−0.3
−0.4	−0.4
−0.5	−0.5
−0.6	−0.6
−0.7	−0.7
−0.8	−0.8
−0.9	−0.9
−1.0	−1.0

b

Figure 16. (a) Breaker states and (b) boundaries for TPF SC.

The SC conditions (Figure 16a) are the TPF boundaries for the no breaker operation in yellow and breaker operation in red (Figure 16b). When the measured TPF conditions are reached at 450 s, the BKX breaker (wind farm side) is operated for a measured TPF smaller than 0.90. The breaker states depend on the "AND/OR" logic conditions in Figure 16a. If a checked condition is triggered, a sending transaction from the DLT is performed by using the CheckConditions function of the TPF chaincode. The TPF SC conditions (Figure 16) are evaluated for the past 450 s period based on the provided timestamp of the CGGS that is synchronized with the SEL 700GT relay in the POI. If the condition is met, the chaincode event will be generated, and the chaincode event listener subscribed to these events will initiate the required action. If the event requires a breaker operation, the listener will trigger a code to send the breaker operation command, which will be received by the Telnet protocol port of the SEL 700GT relay in the POI.

5.1.4 Conditions for Voltage Service Limits Smart Contract

In the CGGS, the conditions for the VSL SC are created based on the flowcharts in Figure 13 and Figure 14. The VSL SC is based on using a backup power quality method that could be performed after using the capacitor bank applications, load shedding, and transformer/load tap changers on the grid side. Next, the VSL condition and operation of the BKX breaker (wind farm side) were implemented at a time of 450 s. The VSL conditions depend on the time, states of breaker, and phase voltages measured in the POI, as shown in Figure 17. If a condition is met for the detailed logic shown in Figure 17, the DLT event is generated. The event is then emitted as a JSON message that is received by the listener application to manage the operation of the BKX breaker (wind farm side). The listener application was generated as a Go program with the HLF software that enables receiving the event from the SC and performing the communication between the SEL 700GT relay and CGGS for operating the BKX breaker (wind farm

side) in the POI. This design was made because the SCs are unable to directly interact with external resources, and they need to be executed in a distributed manner to reach a result that is consistent and deterministic in the CGGS.

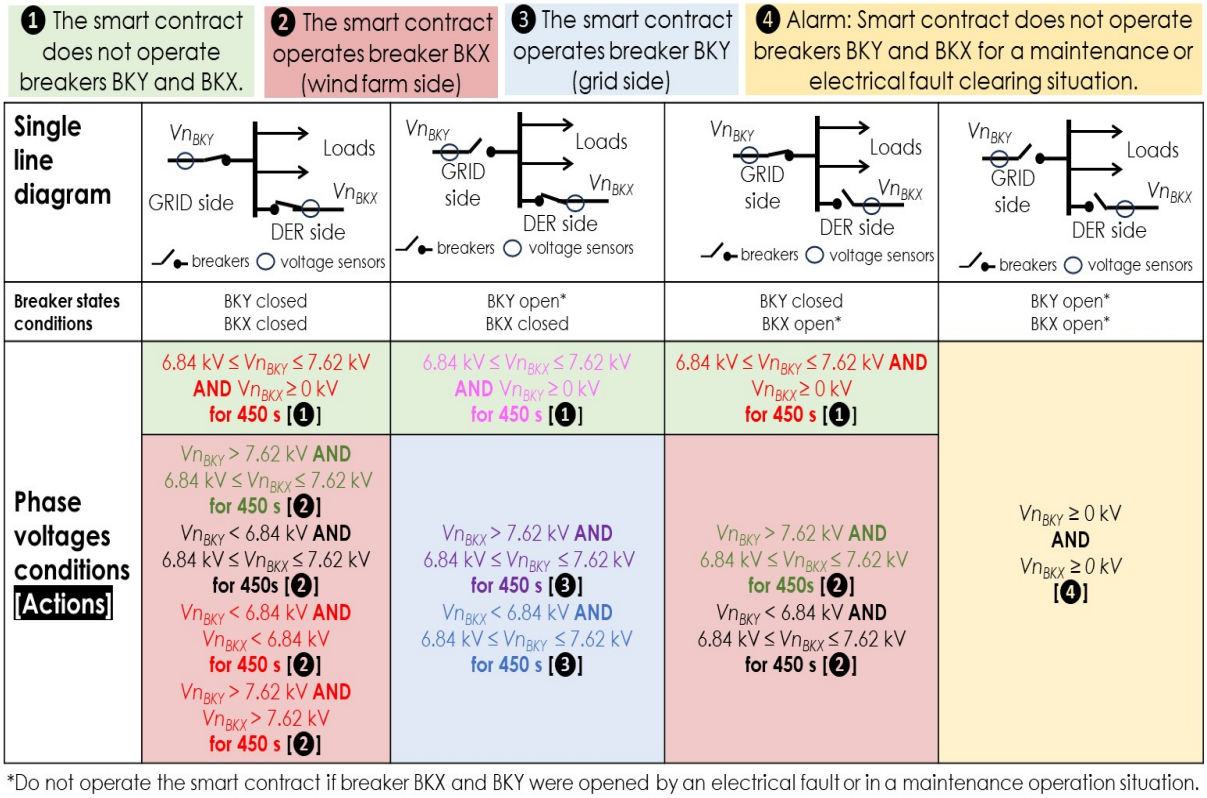


Figure 17. Breaker states and voltages conditions for the VSL SC.

The SC conditions (Figure 17) represent the VSL boundaries. If the measured phase voltage magnitude conditions are reached for a duration of 450 s, the wind farm side breaker is operated for a voltage range outside 6,840–7,620 V. Breaker states and phase voltage conditions are based on the "AND" logic. If a checked condition is triggered, a sending transaction from the DLT is performed by using the CheckConditions function of the VSL chaincode. The VSL SC conditions (Figure 17) are evaluated for the past 450 s period based on the provided timestamp of the CGGS that is synchronized with the SEL 700GT relay in the POI. If the condition is met, the chaincode event will be generated, and the chaincode event listener subscribed to these events will initiate the required action. If the event requires a breaker operation, the listener will trigger a code to send the breaker operation command, which will be received by the Telnet protocol port of the SEL 700GT relay in the POI.

6. USE CASE SCENARIOS

6.1 TESTS FOR TOTAL POWER FACTOR AND VOLTAGE SERVICE LIMITS SMART CONTRACT

This section presents the use case scenarios performed for the TPF and VSL SCs. In the assessment of the TPF and VSL SCs, four and five tests were performed, respectively. The tests were based on studying the

behavior of the SCs using DLT for different scenarios like normal operation and temporary electrical faults, such as where the SCs should not operate the breakers of the SEL 700GT relay in the POI and where extra loads or anomaly events could trigger the conditions for operating the breakers at the SEL 700GT relay in the POI.

6.1.1 Tests of Total Power Factor Smart Contract

In the TPF SC, four tests were run based on the scope of work. The TPF SC was assessed under normal situations with different grid connections (with and without DER) and electrical fault situations. In the normal operation tests, the TPF SC conditions are delayed for 450 s to permit the operation of load shedding, transformer/load tap changers, and/or capacitor banks on the grid side. After 450 s, the CGGS implemented the SC using DLT and the SEL 700GT relay in the POI. In the electrical fault tests, the TPF SC did not operate the breakers in the POI for the SEL 700GT relay because temporary electrical faults of 1 s (60 cycles) were simulated.

Tests 1 and 2 in Table 5 are based on normal electrical grid simulations. For Test 1, the BKY breaker (grid side) feeds the main feeder loads, which measured a TPF between +0.90 and +1.00. Then, the SC did not operate any breaker. For Test 2, the BKY breaker (grid side) and the BKX breaker (wind farm side) feed the main loads. A poor TPF is measured on the BKY breaker (grid side), and the BKX breaker (wind farm side) was opened. Tests 3 and 4 in Table 5 are based on performing temporary electrical fault simulations, like windblown tree branches, lightning, or bird/rodent contact with wires [49]. In these use case scenarios, the TPF SC is not initiated because the simulations for the electrical faults have only a few cycles or seconds [49]. For Test 3, the BKY breaker (grid-side breaker) feeds the main loads, and a good TPF between +0.90 and +1.00 was measured during the prefault state. Then, a temporary three-line-to-ground (3LG) electrical fault of 1 s (60 cycles) at the grid-side power line was run. However, the SC did not operate the breakers in the POI between the grid side and wind farm side because the electrical fault was cleared by itself during the simulation. For Test 4, a temporary single-line-to-ground (SLG) electrical fault of 1 s (60 cycles) at the grid-side power line was performed. Table 5 shows the test scenarios based on the electrical grid in Figure 6.

.

Table 5. Tests scenarios for TPF SC.

***†TEST 1:** Normal situation and no breaker operation.	***Title:*** Main feeder loads are connected only with the grid-side breaker (BKY) that measures a TPF ≥ +0.90; thus, the SC does not operate any breaker. ***Description***: Initially the grid-side breaker (BKY) is closed and the wind farm–side breaker (BKX) is opened. The wind farm–side load has 2.06 MW, +1000 VAR, −1000 VAR. The fossil fuel power plant (Utility C) through the substation (Utility A) is connected to the main feeder loads (2 X [1.25 MW, +0.5 MVAR, −1000 VAR]). The grid-side breaker (BKY) shows a TPF ≥ +0.90, meaning reactive losses are practically none on the grid-side power line. The TPF SC does not operate the breakers at the grid-side (BKY) and wind farm (BKX) for a TPF ≥ +0.90.
***TEST 2:** Normal situation and wind farm-side breaker (BKX) operation.	***Title:*** Main feeder loads are connected with the grid-side breaker (BKY) and wind farm–side breaker (BKY); then the grid-side breaker (BKY) measures a poor TPF, and the wind farm–side breaker (BKX) is opened. ***Description***: Initially the grid-side breaker (BKY) is closed, and the wind farm-side breaker (BKX) is closed. The wind farm-side load has 2.06 MW, +1000 VAR, −1000 VAR. The fossil fuel power plant (Utility C) through the substation (Utility A) and the wind farm are connected to the main feeder loads (2 X [1.25 MW, +0.5 MVAR, −1000 VAR]). The grid-side breaker (BKY) shows a TPF < +0.90, meaning that the TPF in the grid-side power line generates reactive losses. The wind farm–side breaker (BKX) is opened after 400 s, and the TPF is ≥ +0.90 at the grid-side breaker (BKY).
***†TEST 3:** Temporary 3LG electrical fault situation and no breaker operation.	***Title:*** Main feeder loads are connected only with the grid-side breaker (BKY) and measure a TPF > +0.90. Thus, a temporary 3LG electrical fault of 1 s (60 cycles) at the grid-side power line is set, but the SC does not operate any breaker because the electrical fault is cleared. ***Description***: Initially the grid-side breaker (BKY) is closed, and the wind farm–side breaker (BKX) is opened. The wind farm–side load has 2.06 MW, +1000 VAR, −1000 VAR. The fossil fuel power plant (Utility C) through the substation (Utility A) is connected to the main feeder loads (2 X [1.25 MW, +0.5 MVAR, −1000 VAR]). Then a temporary 3LG electrical fault of 1 s (60 cycles) at the end of the grid-side power line is set at 100 s. The grid-side breaker (BKY) shows a swing and poor power factor during the fault state, until the fault is cleared by itself. However, the TPF SC does not operate the grid-side breaker (BKY), and the wind farm–side breaker (BKX) remains open.
***†TEST 4:** Temporary SLG electrical fault situation and no breaker operation.	***Title:*** Main feeder loads are connected only with the grid-side breaker (BKY) and measure a TPF > +0.90. Thus, a temporary SLG electrical fault of 1 s (60 cycles) at the grid-side power line is set, but the SC does not operate any breaker because the electrical fault is cleared. ***Description***: Initially the grid-side breaker (BKY) is closed, and the wind farm-side breaker (BKX) is opened. The wind farm–side load has 2.06 MW, +1000 VAR, −1000 VAR. The fossil fuel power plant (Utility C) through the substation (Utility A) is connected to the main feeder loads (2 X [1.25 MW, +0.5 MVAR, −1000 VAR]). Then a temporary SGL electrical fault of 1 s (60 cycles) at the end of the grid-side power line is set at 100 s. The grid-side breaker (BKY) shows a swing and poor power factor during the fault state, until the fault clears by itself. However, the TPF SC does not operate the grid-side breaker (BKY), and the wind farm–side breaker (BKX) remains open.

*Tests were run for a total time of 600 s. †Tests 1, 3, and 4 started the simulation with all breakers closed; then the breaker BKX (wind farm-side) was tripped manually from the SEL 700GT relay to get the initial condition, 3LG (three lines to ground), SLG (single line to ground).

6.1.2 Tests of Voltage Service Limit Smart Contract

In the VSL SC, five tests were run. The VSL SC was tested under normal operation and temporary electrical fault situations. In the normal operation tests, the VSL SC conditions are delayed for 450 s to permit the operation of load shedding, transformer/load tap changers, and/or capacitor banks on the grid side. After 450 s, the CGGS implemented the SC using DLT and the SEL 700GT relay in the POI. In the electrical fault tests, the VSL SC did not operate the breakers in the POI for the SEL 700GT relay because temporary electrical faults of 1 s (60 cycles) were simulated. Table 6 shows the test scenarios based on the electrical grid (Figure 6).

Table 6. Test scenarios for VSL SC.

***TEST 1:** Normal situation and no breaker operation.	***Title:*** Main loads are feed with the grid-side (BKY) and wind farm-side (BKX) breakers, and both sides measure phase voltages between 6,840 and 7,620 volts, the SC does not operate any breaker. ***Description:*** Initially the grid-side (BKY) and wind farm-side (BKX) breakers are closed. **The wind farm-side load has 2.06 MW, +1000 VAR, -1000 VAR. The main loads (2 X [1.25 MW, +0.5 MVAR, -0.5 MVAR])** are feed through the fossil fuel power plant (Utility B) with the substation (Utility A) and wind farm (Utility C). The grid-side (BKY) and wind farm-side (BK X) breakers show phase voltages between 6,840 and 7,620 volts. Then the VSL SC does not operate the grid-side (BKY) and wind farm-side (BKX) breakers.
***TEST 2:** Normal situation with extra load on wind farm-side and breaker (BKX) operation.	***Title:*** Main loads are feed with the grid-side (BKY) and wind farm-side (BKY) breakers, with an extra inductive load on the wind farm-side and both breakers measure phase voltages below 6,840 volts, then the SC opens the wind farm-side (BKX) breaker. ***Description:*** Initially the grid-side (BKY) wind farm-side (BKX) breakers are closed. **The wind farm-side load has 2.06 MW, +1000 VAR, -1000 VAR** with **an extra load of +1.00 MVAR. The main loads (2 X [1.25 MW, +0.5 MVAR, -1000 VAR])** are feed through the fossil fuel power plant (Utility B) through the substation (Utility A) and wind farm (Utility C). Then the grid-side (BKY) and wind farm-side (BKX) breakers show phase voltages below 6,840 volts. Then, the VSL SC opens the wind farm-side (BKX) breaker after 400 seconds and the phase voltages on the grid-side are set between 6,840 and 7,620 volts.
***TEST 3:** Normal situation with a capacitive extra load on wind farm-side and breaker (BKX) operation.	***Title:*** Main loads are feed with the grid-side (BKY) and wind farm-side (BKY) breakers, with an extra capacitive load on the wind farm-side and both breakers measure phase voltages above 7,620 volts, then the SC opens the wind farm-side (BKX) breaker. ***Description:*** Initially the grid-side (BKY) wind farm-side (BKX) breakers are closed. **The wind farm-side load has 2.06 MW, +1000 VAR, -1000 VAR with an extra load of -3.00 MVAR. The main loads (2 X [1.25 MW, +0.5 MVAR, -1000 VAR])** are feed through the fossil fuel power plant (Utility B) through the substation (Utility A) and wind farm (Utility C). Then the grid-side (BKY) and wind farm-side (BKX) breakers show phase voltages above 7,620 volts. Then, the VSL SC opens the wind farm-side (BKX) breaker after 400 seconds and the phase voltages on the grid-side are set between 6,840 and 7,620 volts.
***TEST 4:** Temporary LL electrical fault situation and no breaker operation.	***Title:*** Main loads are feed with the grid-side (BKY) and wind farm-side (BKX) breakers, then a temporary LL electrical fault of 1 s (60 cycles) at the grid-side power line is set, but the SC does not operate any breaker because the electrical fault is cleared. ***Description:*** Initially the grid-side (BKY) and wind farm-side (BKX) breakers are closed. **The wind farm-side load has 2.06 MW, +1000 VAR, -1000 VAR. The main loads (2 X [1.25 MW, +0.5 MVAR, -0.5 MVAR])** are feed through the fossil fuel power plant (Utility B) with the substation (Utility A) and wind farm (Utility C). **Then, a temporary A-B phase electrical fault of 1 s (60 cycles) at the end of the grid-side power line is set at 100 s**. The grid-side (BKY) and wind farm-side (BKX) breakers show some low voltages during the fault state, up to the fault is cleared by itself. However, the VSL SC does not operate the grid-side (BKY) and wind farm-side (BKX) breakers.
*†**TEST 5:** Temporary SLG electrical fault situation and no breaker operation.	***Title:*** Main loads feed with the grid-side (BKY) breaker, then a temporary SLG electrical fault of 1 s (60 cycles) at the grid-side power line is set, but the SC does not operate any breaker because the electrical fault is cleared. ***Description:*** Initially the grid-side (BKY) breaker is closed, and the wind farm-side (BKX) breaker is opened. **The wind farm-side load has 2.06 MW, +1000 VAR, -1000 VAR**. The fossil fuel power plant (Utility B) through the substation (Utility A) is connected to **the main loads (2 X [1.25 MW, +0.5 MVAR, -0.5 MVAR]). Then, a temporary phase A –ground electrical fault of 1 second (60 cycles) at the end of the grid-side power line is set at 100 s.** The grid-side (BKY) breaker shows a low voltage for phase A during the fault state, up to the fault is cleared by itself. However, the VSL SC does not operate the grid-side (BKY) and wind farm-side (BKX) breakers.

*Tests were run for a total time of 600 seconds, †Test 5 start the simulation with all breakers closed, then the breaker BKX (wind farm-side) is tripped manually from the SEL 700GT relay to get the initial condition, 3LG: three line to ground, SLG: single line to ground

7. RESULTS

7.1 RESULTS FOR TOTAL POWER FACTOR SMART CONTRACT

The TPF SC was evaluated by comparing the measured data from the CGGS, real-time simulator, and SEL 700GT relay in the POI. The TPF SC was operated by the data from the CGGS. In the TPF SC, the TPF limits were between +0.9 and +1.0 at grid side, and the operation of the BKY breaker (grid side) and

BKX breaker (wind farm side) was controlled by the SEL 700GT relay in the POI using the SC of the CGGS. The events from the CGGS, real-time simulator, and SEL 700GT relay demonstrated a successful assessment for the TPF SC, which was assessed for normal operation tests and temporary electrical fault tests. The CGGS using DLT with SC was evaluated, which included collecting and plotting TPF SC tests. The normal operation tests (Figure 18 and Figure 19) and temporary electrical fault tests (Figure 20 and Figure 21) were run for a total simulation time of 600 s, which allowed activation of the SC for a TPF less than +0.9 on the grid side after 400 s. In the electrical fault tests, the temporary electrical faults were simulated for a short time of 60 cycles (1 s), then a measured TPF smaller than +0.9 at fault states was observed. However, the TPF SC did not operate the breakers of the SEL 700GT in the POI because anomaly transient events were simulated. Table 7 shows the tests and relay events for the TPF SC.

Table 7. Tests and plots for the TPF SC.

Type of test	Test number	Real-time simulator source	Cyber Grid Guard source	SEL 700GT relay source	
		Figures	Figures	Figures	Event
Normal situation tests	1	Figures 18a–18b	Figures 18c–18d	Figures 18e–18f	CEV_10706
	2	Figures 19a–19b	Figures 19c–19d	Figures 19e–19f	CEV_10671
Temporary electrical fault tests	3	Figures 20a–20b	Figures 20c–20d	Figures 20e–20f	CEV_10703
	4	Figures 21a–21b	Figures 21c–21d	Figures 21e–21f	CEV_10705

7.1.1 Normal Situation Tests

Test 1 (Table 5) is a normal situation that presents the TPF of the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) from the real-time simulator (Figure 18a and Figure 18b), the CGGS (Figure 18c and Figure 18d), and the SEL 700GT relay (Figure 18e and Figure 18f). The CGGS and real-time simulator show the TPF plots up to 600 s, and the SEL 700GT relay records the TPF plots at 7:28 p.m. Figure 18a/18c/18e show the TPF of the grid side (TPF_{BKY}), and Figure 18b/18d/18f show the TPF of the wind farm side (TPF_{BKX}). In the real-time simulator (Figure 18a and Figure 18b) and CGGS (Figure 18c and Figure 18d), the simulation initially has the breakers closed, and the TPF swing is around ±1.00 for a couple of seconds when the connection of the wind farm is made. The wind farm side breaker (BKX) is then opened to obtain the situation of Test 1. The fossil fuel power plant (Utility C) through the substation (Utility A) is connected to the main feeder loads (Figure 6). The grid-side breaker (BKY) presents a TPF between +0.90 and +1.00 because the reactive losses are nonexistent on the grid-side power line. The TPF SC does not operate the breakers at the grid side (BKY) and wind farm side (BKX) for a TPF between +0.90 and +1.00 in the POI. In the real-time simulator (Figure 18b), the wind farm side TPF swings because the breaker (BKX) is opened and phase currents are zero; an indetermined value is then obtained from Eq. (4). However, from the CGGS, the TPF on the wind farm side (Figure 18d) is +1.00 because the manufacturer condition for the SEL 700GT relay is that when the breaker is opened, the measured TPF is +1.00, and an indetermined value is not measured. However, from the SEL 700GT relay event (Figure 18f) at 7:28 p.m., the TPF on the wind farm side swings because the BKX breaker is opened and phase currents are zero; an indetermined value is then obtained from Eq. (4). This test led to the conclusion that the TPF SC did not operate the SEL 700GT relay's breakers because the TPF of the grid side (TPF_{BKY}) was between the limits of +0.90 and +1.00.

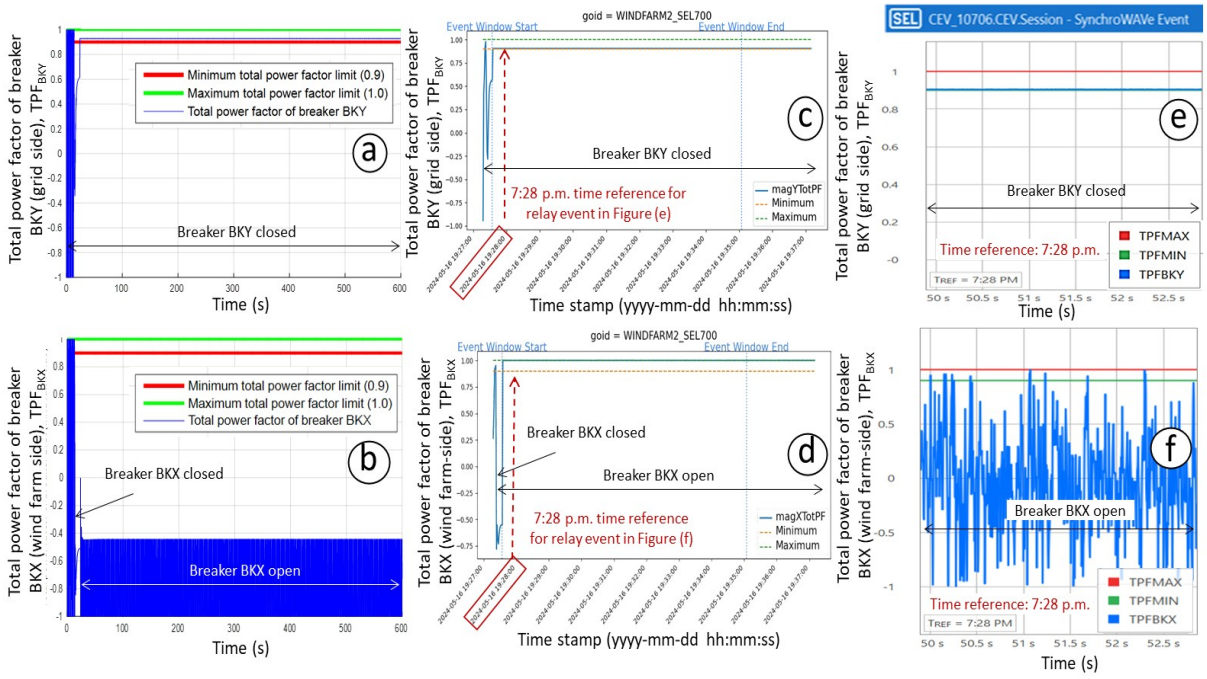


Figure 18. Measured TPF from (a and b) the real-time simulator, (c and d) CGGS, and (e and f) relay in Test 1 for a normal situation and no breaker operation [1].

Test 2 (Table 5) is a normal situation that presents the TPF of the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) from the real-time simulator (Figure 19a and Figure 19b), CGGS (Figure 19c and Figure 19d), and the SEL 700GT relay (Figure 19e and Figure 19f). The CGGS and real-time simulator recorded the TPF up to 600 s, and the SEL 700GT relay recorded the TPF plots at 7:09 p.m. The measured TPFs on the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) are shown in Figure 19a/19c/19e and Figure 19b/19d/19f, respectively. In the real-time simulator (Figure 19a and Figure 19b) and CGGS (Figure 19c and Figure 19d), all breakers initially are closed during the simulation, and the TPF swings around ± 1.00 for a couple of seconds when the wind farm is connected. The TPF on the grid side (TPF_{BKY}) is smaller than +0.90 for the real-time simulator (Figure 19a) and CGGS (Figure 19c) plots for more than 400 s. In this test the fossil fuel power plant (Utility C) through the substation (Utility A) and the wind farm (Utility B) is feeding the main loads (Figure 6). The BKY breaker (grid side) shows a TPF smaller than +0.90, meaning the TPF on the grid-side power line is given by reactive losses. The BKX breaker wind farm side is opened after 400 s, and the TPF is between +0.90 and +1.00 on the grid side. From the real-time simulator (Figure 19b), the TPF on the wind farm side swings because the BKX breaker is opened and phase currents are zero; an indetermined value is obtained from Eq. (4). However, the TPF on the wind farm side from the CGGS (Figure 19d) is +1.00 because a manufacturer condition for the SEL 700GT relay defines that if the breaker is opened, the TPF is +1.00. However, from the SEL 700GT relay event (Figure 19f) at 7:09 p.m., the TPF on the wind farm side swings because the BKX breaker is opened and phase currents are zero; an indetermined value is then obtained from Eq. (4). This normal situation test led to the conclusion that the TPF SC opened the BKX breaker (wind farm side) of the SEL 700GT relay because the TPF on the grid side (TPF_{BKY}) was smaller than +0.90.

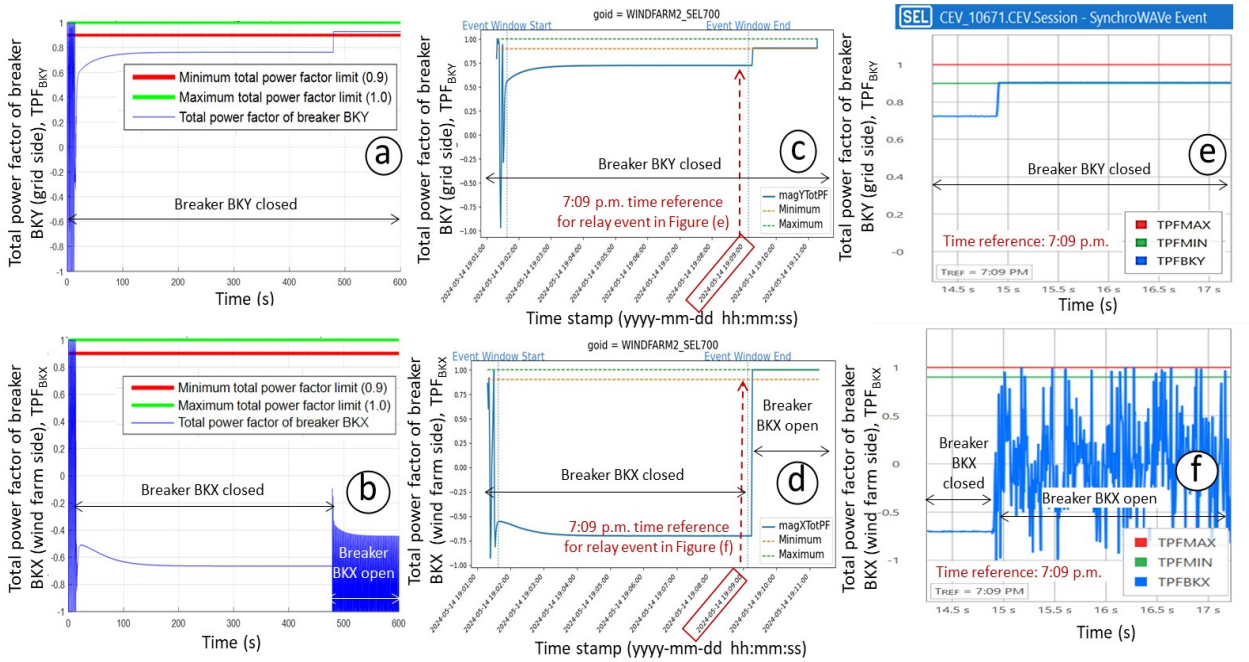


Figure 19. Measured TPF from (a and b) the real-time simulator, (c and d) CGGS, and (e and f) relay in Test 2 for a normal situation and wind farm side breaker BKX operation [1].

7.1.2 Temporary Electrical Fault Situation Tests

The temporary electrical fault situation tests were run to assess that the TPF SCs didn't operate for possible anomaly events. Test 3 (Table 5) is a temporary electrical fault that presents the TPF of the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) from the real-time simulator (Figure 20a and Figure 20b), the CGGS (Figure 20c and Figure 20d), and the SEL 700GT relay (Figure 20e and Figure 20f). The CGGS and real-time simulator record the TPF up to 600 s, and the SEL 700GT relay records the TPF at 6:10 p.m. The TPFs on the grid side (TPF_{BKY}) and the wind farm side (TPF_{BKX}) are shown in Figure 20a/20c/20e and Figure 20b/20d/20f, respectively. In the real-time simulator (Figure 20a and Figure 20b) and CGGS (Figure 20c and Figure 20d), simulation initially has the circuit breakers closed and the TPF swings around ± 1.00 for a couple of seconds when the connection of the wind farm is performed. The BKX breaker (wind farm side) is opened to obtain the initial test condition. The fossil fuel power plant (Utility C) through the substation (Utility A) is connected to the main loads (Figure 6). A temporary 3LG electrical fault of 1 s (60 cycles) at the end of the grid-side power line (near SEL 700GT relay controlled by CGGS) is then set at 100 s. The BKY breaker (grid side) presents a poor power factor during the fault state (Figure 20a) until the temporary electrical fault is cleared by itself. However, the TPF SC does not operate the BKY breaker (grid side) and BKX breaker (wind farm side) because this is a temporary electrical fault situation. In the real-time simulator (Figure 20a), the TPF on the grid side has a peak during the fault state, like the TPF on the grid side from the CGGS (Figure 20c). However, the SEL 700GT relay event (Figure 20f) at 6:10 p.m. shows the TPF on the wind farm side swinging because the BKX breaker is opened and phase currents are zero; an indetermined value is then obtained from Eq. (4). This temporary electrical fault test led to the conclusion that the TPF SC did not operate the SEL 700GT relay's breakers because a poor measured TPF on the grid side (TPF_{BKY}) was available during the temporary fault state.

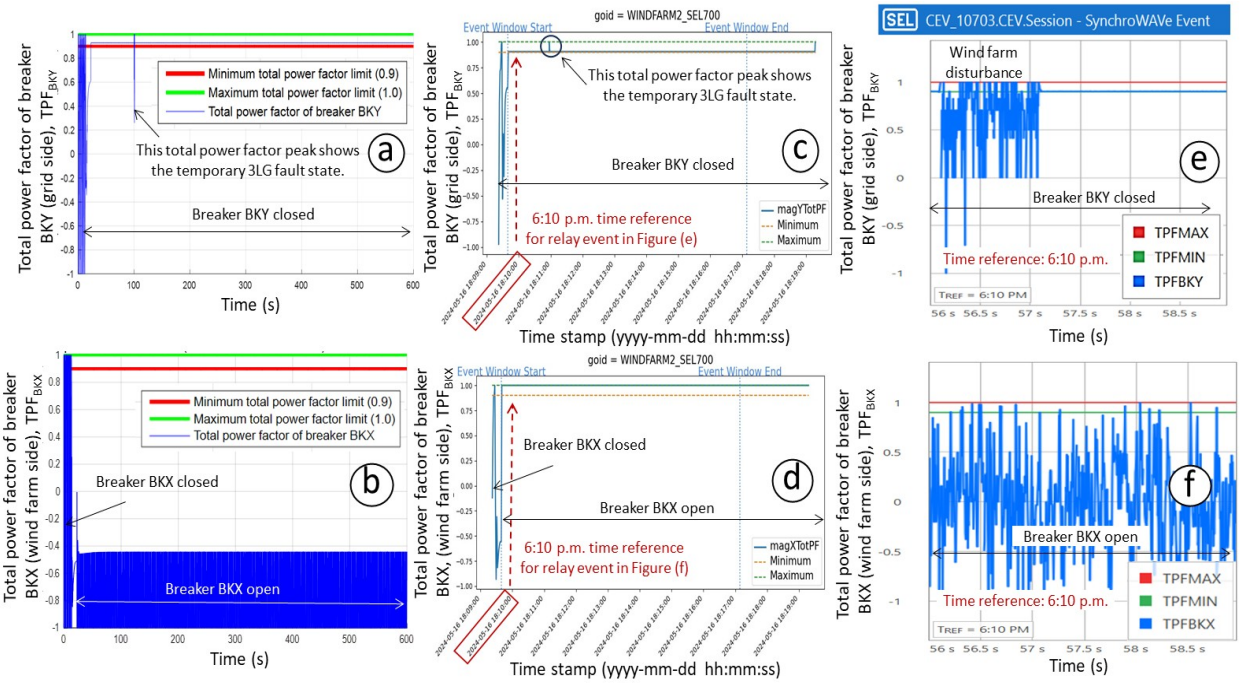


Figure 20. Measured TPF from (a and b) the real-time simulator, (c and d) CGGS, and (e and f) relay in Test 3 for a temporary 3LG electrical fault situation and no breaker operation [1].

Test 4 (Table 5) is a temporary electrical fault that presents the TPF of the grid side (TPF_{BKY}) and wind farm–side (TPF_{BKX}) from the real-time simulator (Figure 21a and Figure 21b), CGGS (Figure 21c and Figure 21d), and SEL 700GT relay (Figure 21e and Figure 21f). The CGGS and real-time simulator recorded the TPF up to 600 s, and the SEL 700GT relay recorded the TPF at 7:01 p.m. The TPF on the grid side (TPF_{BKY}) and TPF on the wind farm side (TPF_{BKX}) are shown in Figure 21a/21c/21e and Figure 21b/21d/21f, respectively. In the real-time simulator (Figure 21a and Figure 21b) and CGGS (Figure 21c and Figure 21d), the breakers initially are closed and the TPF swings around ± 1.00 for a couple of seconds when the connection of the wind farm is performed. The BKX breaker (wind farm side) is opened to obtain the test condition. The fossil fuel power plant (Utility C) through the substation (Utility A) is connected to the main loads (Figure 6). Then a temporary SLG electrical fault of 1 s (60 cycles) at the end of the grid-side power line (near SEL 700GT relay controlled by CGGS) is set at 100 s. The BKY breaker (grid side) shows a small swing of TPF at the fault state (Figure 21a/21c/21e) until the temporary electrical fault is cleared by itself. However, the TPF SC does not operate the BKY breaker (grid side) and BKX breaker (wind farm side) because this is a temporary electrical fault. From the real-time simulator (Figure 21a), the TPF on the grid side shows a little peak during at the fault state, like the TPF on the grid side for the CGGS (Figure 21c). However, from the SEL 700GT relay event (Figure 21f) at 7:01 p.m., the TPF on the wind farm side swings because the BKX breaker is opened and phase currents are zero; an indetermined value is then obtained from Eq. (4). This temporary electrical fault test led to the conclusion that the TPF SC did not operate the SEL 700GT relay's breakers during the temporary fault.

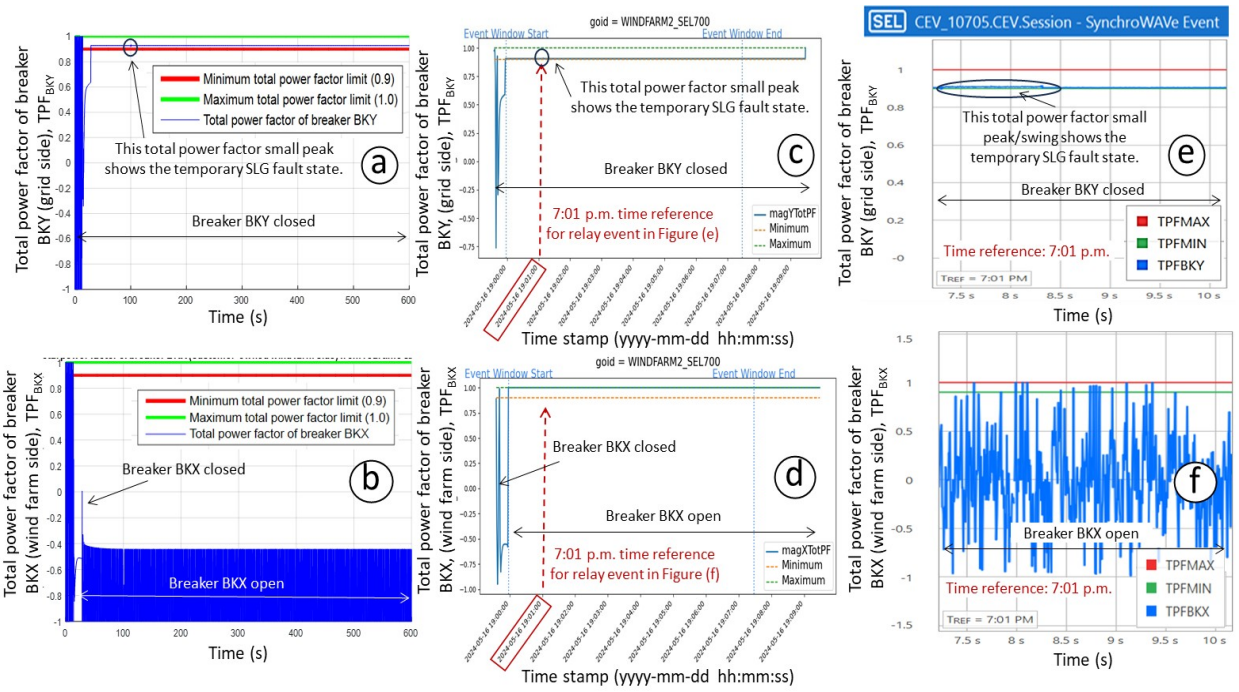


Figure 21. Measured TPF from (a and b) the real-time simulator, (c and d) CGGS, and (e and f) relay in Test 4 for a temporary SLG electrical fault situation and no breaker operation [1].

7.2 RESULTS FOR VOLTAGE SERVICE LIMITS SMART CONTRACT

The VSL SC assessment compared data from the CGGS, SEL 700GT relay, and real-time simulator. The CGGS data were applied to operate the breakers in the POI for the VSL SC. The VSL limits on the grid side were defined between 6,840 and 7,620 V to maintain a good condition. Operation of the breakers in the POI between the grid side and wind farm side was managed by the CGGS and the SEL 700GT relay. The collected events from the CGGS, real-time simulator, and SEL 700GT relay were compared to validate a successful evaluation of the VSL SC. The normal situation tests and temporary electrical fault tests were run in the VSL SC assessment. The CGGS used DLT with the SC for performing the normal situation and anomaly event tests. The normal situation tests (Figure 22–Figure 26) and temporary electrical fault situation tests (Figure 27–Figure 30) were simulated for 600 s to permit application of the SC after 400 s. The temporary electrical fault tests were run for a couple of cycles, and the phase voltages outside the permitted limits were observed. The VSL SC did not operate the breakers in POI controlled by the SEL 700GT and CGGS because the anomaly transient events for a short period of 1 s (60 cycles) were available for the temporary electrical fault tests. Table 8 shows the tests and relay events for the VSL SC.

Table 8. Tests and plots for the VSL SC.

Type of test	Test number	Cyber Grid Guard	SEL 700GT relay	
		Figure	Figure	Event
Normal situation test	1	Figure 22	NA	NA
	2	Figure 23	Figure 24	HR_11136
	3	Figure 25	Figure 26	HR_11295
Temporary electrical fault test	4	Figure 27	Figure 28	HR_11210
	5	Figure 29	Figure 30	HR_11211

NA = not available, HR: high resolution.

7.2.1 Normal Situation Tests

Test 1 for the VSL SC (Table 6) was performed for a normal situation. CGGS data are shown in Figure 22. The phase voltage magnitudes (Figure 22a–Figure 22c) and pole states (Figure 22d) of the BKY breaker for the grid side are presented on the left, whereas the phase voltage magnitudes (Figure 22e–Figure 22g) and pole states (Figure 22h) of the BKX breaker for the wind farm side are presented on the right. The main loads were fed with the BKY breaker (grid side) and BKX breaker (wind farm side), and both sides measured the phase voltages between 6,840 and 7,620 V. The VSL SC did not operate any breaker. In the initial conditions of this test, the BKY breaker (grid side) and BKX breaker (wind farm side) were closed. The wind farm side had an extra load of 2.06 MW; +1,000 VAR; and −1,000 VAR. The main feeder loads (2 X [1.25 MW, +0.5 MVAR, −0.5 MVAR]) were fed with the fossil fuel power plant (Utility B) through the substation (Utility A) and wind farm (Utility C) for the electrical grid in Figure 6. The BKY breaker (grid side) and BKX breaker (wind farm–side) then presented the phase voltage magnitudes between 6,840 and 7,620 V (Figure 22a–Figure 22c and Figure 22e–Figure 22g). Consequently the VSL SC did not operate the BKY breaker (grid side) and BKX breaker (wind farm side) as shown in Figure 22d and Figure 22h, respectively.

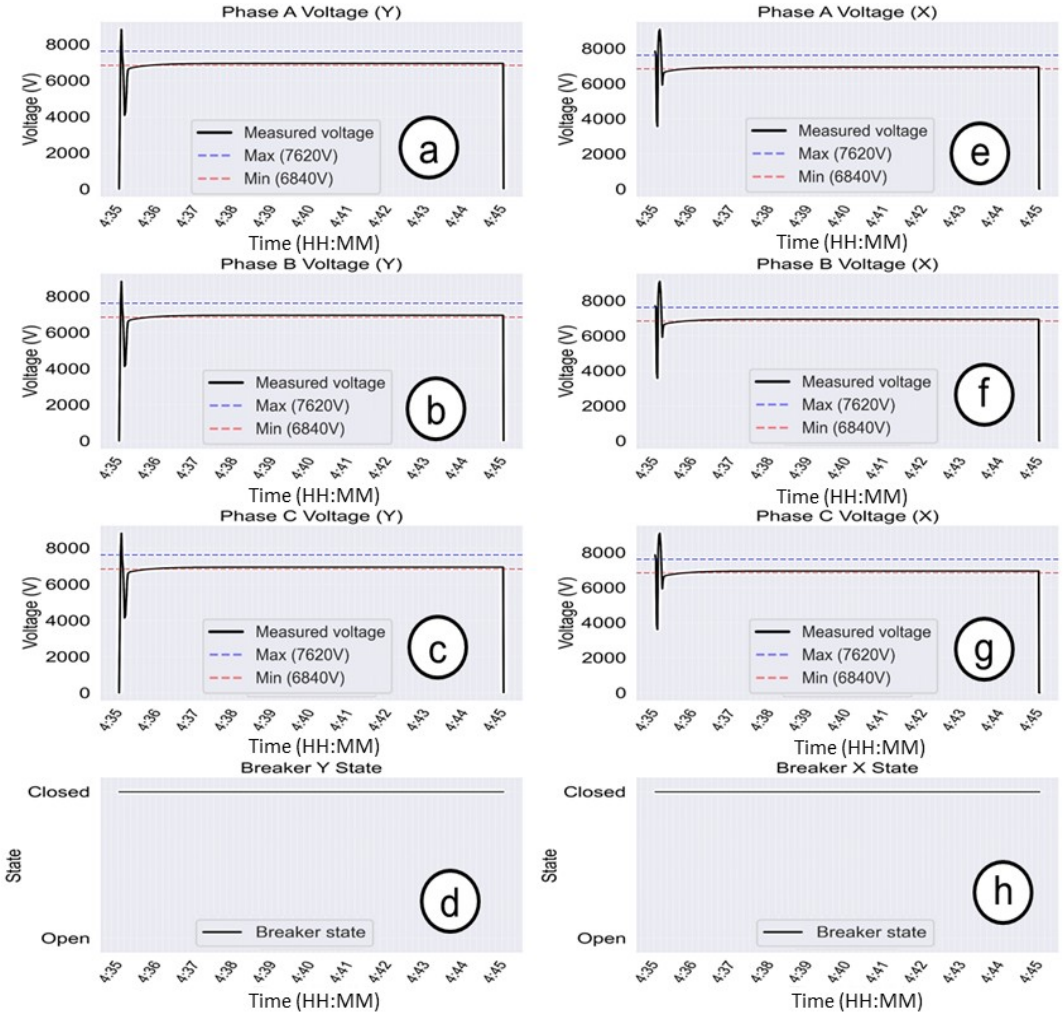


Figure 22. Normal situation and no breaker operation (Test 1) results from Cyber Grid Guard.

Test 2 for the VSL SC (Table 6) was performed for a normal situation. CGGS data are shown in Figure 23. The phase voltage magnitudes (Figure 23a–Figure 23c) and pole states (Figure 23d) of the BKY breaker for the grid side are presented on the left, and the phase voltage magnitudes (Figure 23e–Figure 23g) and pole states (Figure 23h) of the BKX breaker for the wind farm side are presented on the right. The main loads were fed with the BKY breaker (grid side) and BKY breaker (wind farm side), and an extra inductive load on the wind farm side was connected. In this case, both breakers measured phase voltage magnitudes below 6,840 V, then the VSL SC opened the BKX breaker (wind farm side). In the initial conditions of this test, the BKY breaker (grid side) and BKX breaker (wind farm side) were closed for the electrical grid (Figure 6). The wind farm side load had 2.06 MW, +1,000 VAR, and −1,000 VAR with an extra load of +1.00 MVAR. The main loads (2×[1.25 MW, +0.5 MVAR, −1,000 VAR]) were fed by the fossil fuel power plant (Utility B) through the substation (Utility A) and wind farm (Utility C). Then, the BKY breaker (grid side) and BKX breaker (wind farm side) presented phase voltage magnitudes below 6,840 V. The VSL SC opened the BKX breaker (wind farm side) after 400 s (Figure 23h), and the phase voltage magnitudes on the grid side were set between 6,840 and 7,620 V (Figure 23a–Figure 23c). Data collected from the SEL 700GT relay are shown in Figure 24. The instantaneous phase currents (Figure 24a and Figure 24c) and voltages (Figure 24b and Figure 24d) with the trip signals of the BKY and BKX breakers (Figure 24e) are shown for the SEL 700GT relay. The phase voltage magnitudes for the BKY breaker (Figure 24f) and BKX breaker (Figure 24g) are presented with the trip signals of the BKY and BKX breakers (Figure 24h) for the SEL 700GT relay; the manner in which the trip signal of the BKX breaker (wind farm side) is activated by the VSL SC is shown.

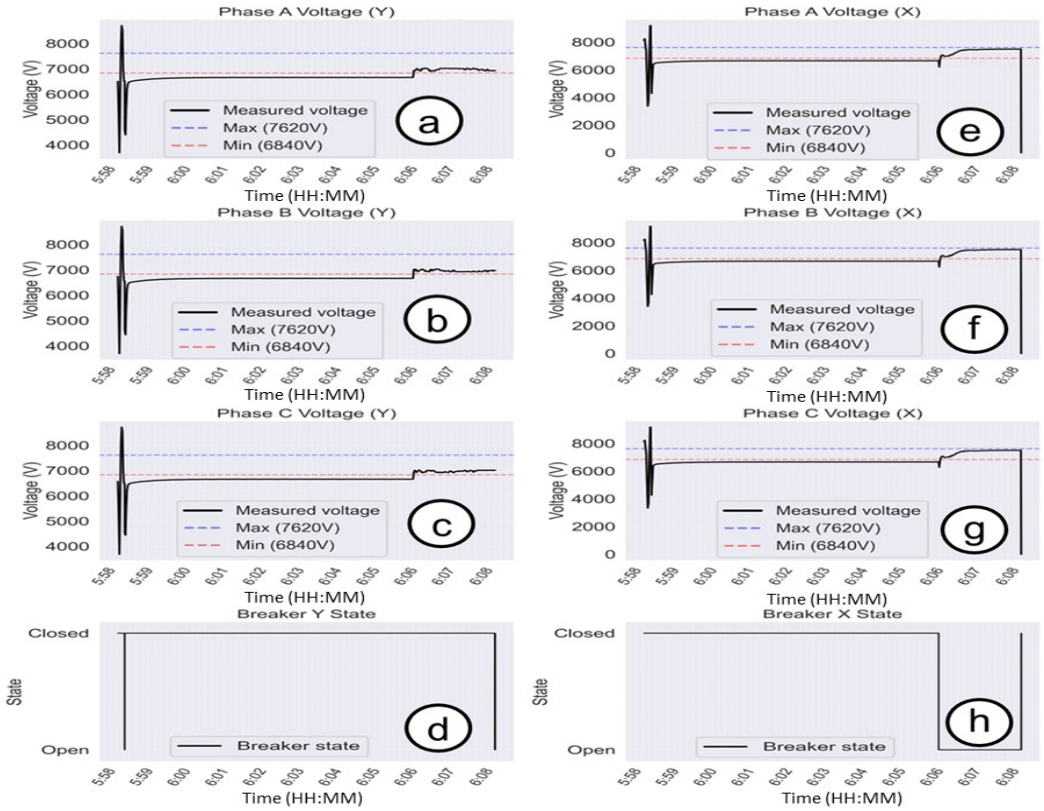


Figure 23. Normal situation with an extra inductive load on the wind farm side and breaker operation (Test 2) results from Cyber Grid Guard.

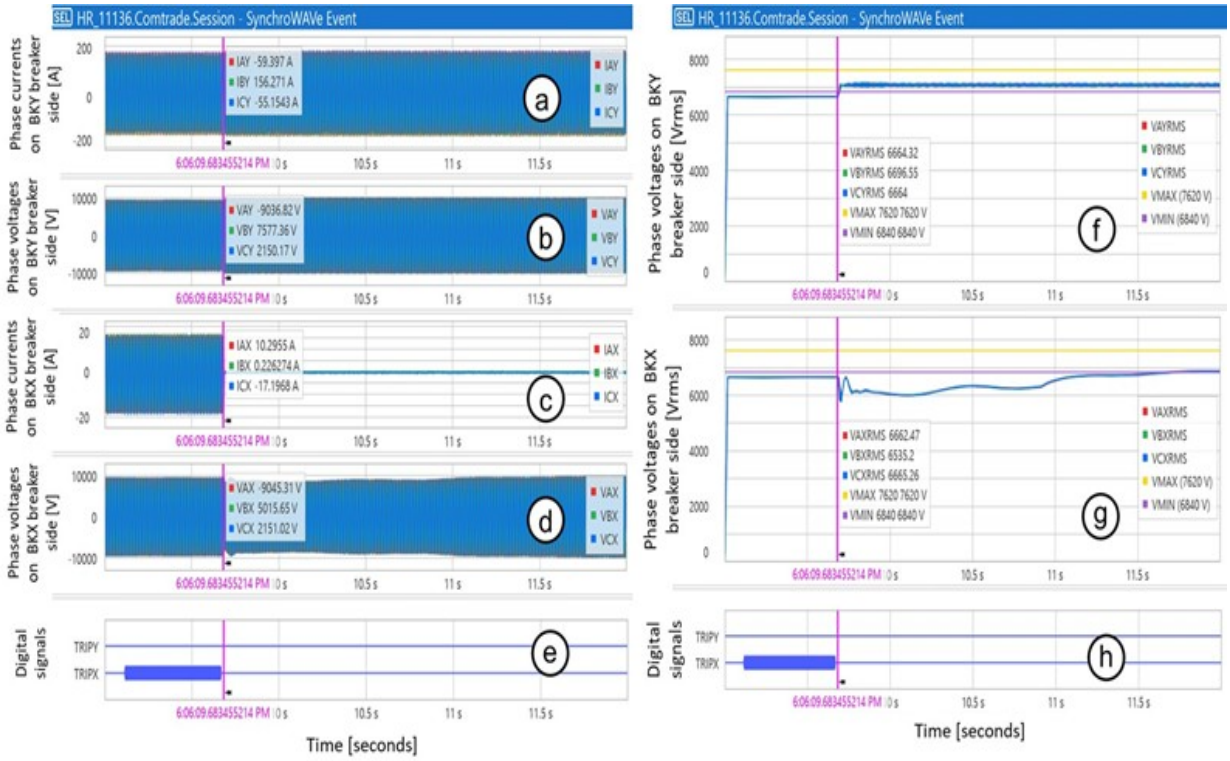


Figure 24. Normal situation with an extra inductive load on the wind farm side and breaker operation (Test 2) results from the SEL 700GT relay.

Test 3 for the VSL SC (Table 6) was performed for a normal situation. CGGS data are shown in Figure 25. The phase voltage magnitudes (Figure 25a–Figure 25c) and pole states (Figure 25d) of the BKY breaker for the grid side are presented on the left, and the phase voltage magnitudes (Figure 25e–Figure 25g) and pole states (Figure 25h) of the BKX breaker for the wind farm side are presented on the right. The main loads were fed with the BKY breaker (grid side) and BKY breaker (wind farm side), and an extra capacitive load on the wind farm side was connected. In this case, both breakers measured the phase voltage magnitudes above 7,620 V, then the VSL SC opened the BKX breaker (wind farm side). In the initial conditions of this test, the BKY breaker (grid side) and BKX breaker (wind farm side) were closed for the electrical grid in Figure 6. The wind farm side load had 2.06 MW, +1,000 VAR, and −1,000 VAR with an extra load of −3.00 MVAR. The main loads (2×[1.25 MW, +0.5 MVAR, −1,000 VAR]) were fed by the fossil fuel power plant (Utility B) through the substation (Utility A) and wind farm (Utility C). Then, the BKY breaker (grid side) and BKX breaker (wind farm side) presented phase voltage magnitudes above 7,620 V. The VSL SC opened the BKX breaker (wind farm side) after 400 s (Figure 25h), and the phase voltage magnitudes on the grid side were set between 6,840 and 7,620 V (Figure 25a–Figure 25c). The data collected from the SEL 700GT relay are shown in Figure 26. The instantaneous phase currents (Figure 26a and Figure 26c) and voltages (Figure 26b and Figure 26d) with the trip signals of the BKY and BKX breakers (Figure 26e) for the SEL 700GT relay are shown. The phase voltage magnitudes for the BKY breaker (Figure 26f) and BKX breaker (Figure 26f) are presented, with the trip signals of the BKY and BKX breakers (Figure 26h) for the SEL 700GT relay; the manner in which the trip signal of the BKX breaker (wind farm side) is activated by the VSL SC is shown.

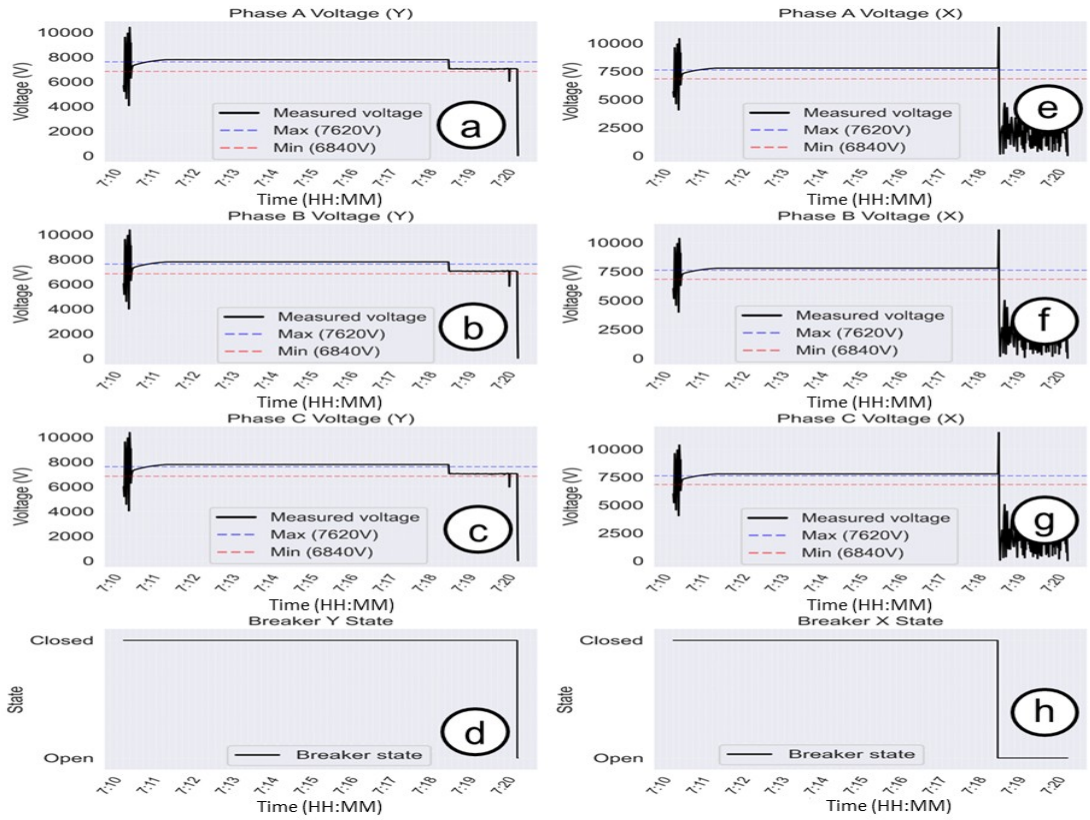


Figure 25. Normal situation with an extra capacitive load on the wind farm side and breaker operation (Test 3) results from Cyber Grid Guard.

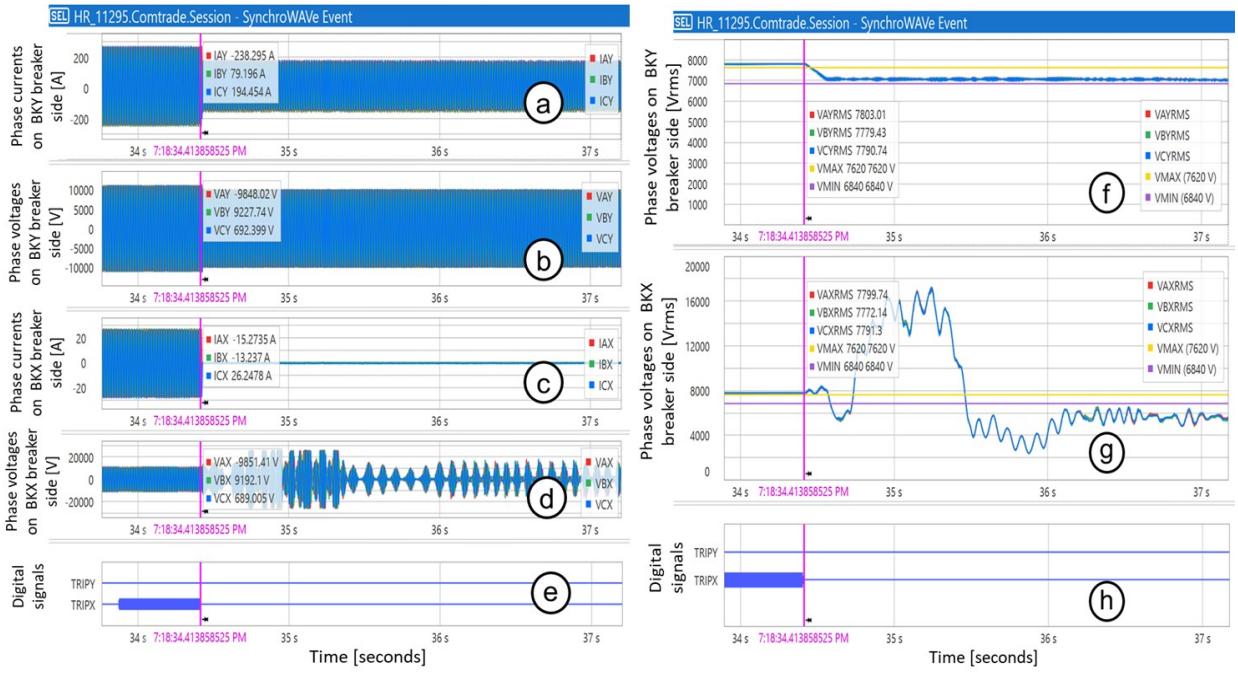


Figure 26. Normal situation with an extra capacitive load on the wind farm side and breaker operation (Test 3) results from the SEL 700GT relay.

7.2.2 Temporary Electrical Fault Tests

Test 4 for the VSL SC (Table 6) was performed for a line-to-line (LL) temporary electrical fault situation with no breaker operation. CGGS data are shown in Figure 27. The phase voltage magnitudes (Figure 27a–Figure 27c) and pole states (Figure 27d) of the BKY breaker for the grid side are presented on the left, and the phase voltage magnitudes (Figure 27e–Figure 27g) and pole states (Figure 27h) of the BKX breaker for the wind farm side are presented on the right. The main loads were fed with the BKY breaker (grid side), then an LL temporary electrical fault of 1 s (60 cycles) at the grid side power line was set, but the VSL SC did not operate any breaker because the electrical fault was cleared by itself. In the initial conditions of this test, the BKY breaker (grid side) and the BKX breaker (wind farm side) were closed for the electrical grid (Figure 6). The wind farm side load had 2.06 MW, +1,000 VAR, and −1,000 VAR. The main loads (2×[1.25 MW, +0.5 MVAR, −1,000 VAR]) were fed by the fossil fuel power plant (Utility B) through the substation (Utility A) and wind farm (Utility C). Then, the phase A to B temporary electrical fault of 1 s (60 cycles) at the end of the grid side power line was set at 100 s. The A and B phase voltage magnitudes for the BKY breaker (grid side) and BKX breaker (wind farm side) show an undervoltage situation during the fault state (Figure 27a, Figure 27b, Figure 27e, and Figure 27f), up to when the fault is cleared by itself. However, the VSL SC does not operate the BKY breaker (grid side) and BKX breaker (wind farm side) as shown in Figure 27d and Figure 27h, respectively, because it was a temporary anomaly event. The data collected from the SEL 700GT relay are shown in Figure 28. The instantaneous phase currents (Figure 28a and Figure 28c) and voltages (Figure 28b and Figure 28d) with the trip signals of the BKY and BKX breakers (Figure 28e) for the SEL 700GT relay are presented. The phase voltage magnitudes for the BKY breaker (Figure 28f) and BKX breaker (Figure 28f) are presented, with the trip signals of the BKY and BKX breakers (Figure 28h) for the SEL 700GT relay. It can be observed that the trip signal of the BKY breaker (grid side) and BKX breaker (wind farm side) are not activated by the VSL SC because the undervoltage of phases A and B were for a short duration generated by the LL temporary electrical fault.

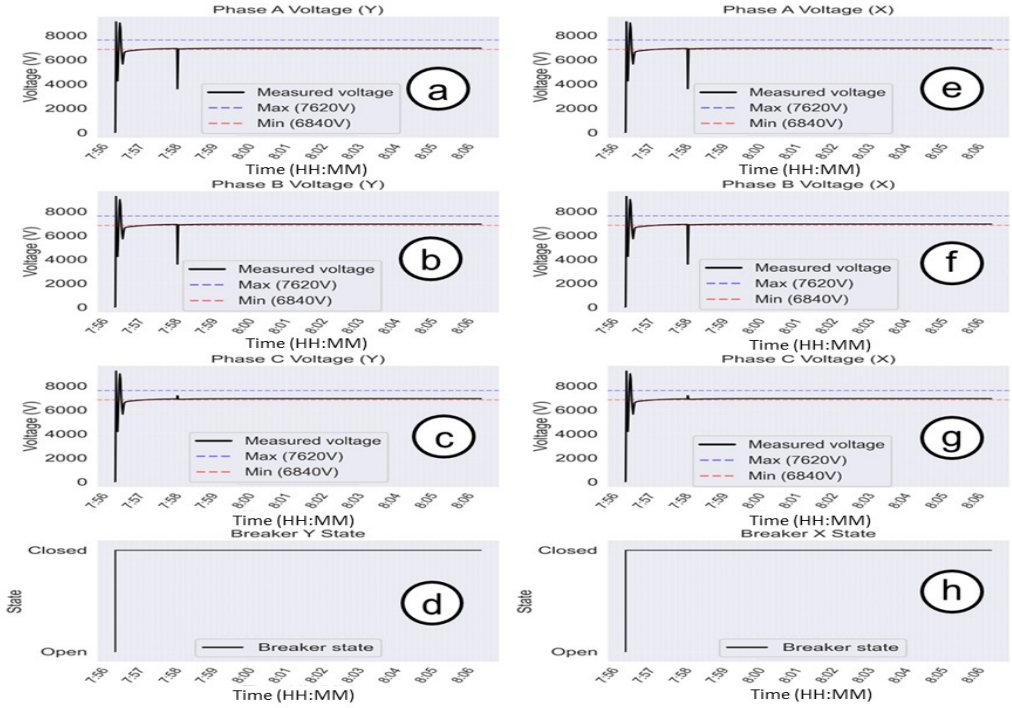


Figure 27. Temporary line-to-line electrical fault at the end of the power line and no breaker operation (Test 4) results from Cyber Grid Guard.

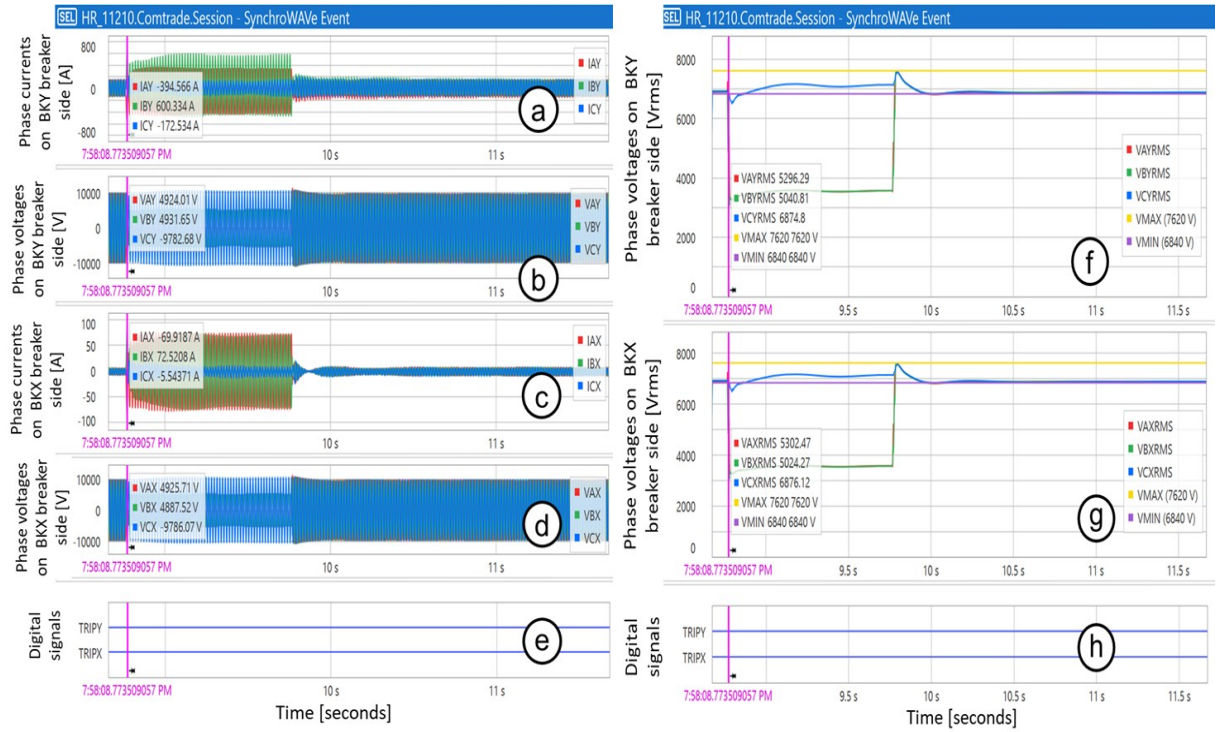


Figure 28. Temporary line-to-line electrical fault at the end of the power line and no breaker operation (Test 4) results from the SEL 700GT relay.

Test 5 for the VSL SC (Table 6) was performed for an SLG temporary electrical fault situation with no breaker operation. CGGS data are shown in Figure 29. The phase voltage magnitudes (Figure 29a–Figure 29c) and pole states (Figure 29d) of the BKY breaker for the grid side are presented on the left, and the phase voltage magnitudes (Figure 29e–Figure 29g) and pole states (Figure 29h) of the BKX breaker for the wind farm side are presented on the right. The main loads were fed with the BKY breaker (grid side), then an SLG temporary electrical fault of 1 s (60 cycles) at the grid-side power line was set, but the VSL SC did not operate any breaker because the electrical fault was cleared by itself. In the initial conditions, the BKY breaker (grid side) and the BKX breaker (wind farm side) were closed for the electrical grid (Figure 6). The wind farm side load had 2.06 MW, +1,000 VAR, and −1,000 VAR. The main loads (2×[1.25 MW, +0.5 MVAR, −1,000 VAR]) were fed by the fossil fuel power plant (Utility B) through the substation (Utility A) and wind farm (Utility C). Then, the phase A to ground temporary electrical fault of 1 s (60 cycles) at the end of the grid-side power line was set at 100 s. The phase A voltage magnitudes for the BKY breaker (grid side) and BKX breaker (wind farm side) show an undervoltage situation during the fault state (Figure 29a and Figure 29e), up to when the fault is cleared by itself. However, the VSL SC does not operate the BKY breaker (grid side) and BKX breaker (wind farm side) as shown in Figure 29d and Figure 29h, respectively, because it was a temporary anomaly event. The data collected from the SEL 700GT relay are shown in Figure 30. The instantaneous phase currents (Figure 30a and Figure 30c) and voltages (Figure 30b and Figure 30d) with the trip signals of the BKY and BKX breakers (Figure 30e) for the SEL 700GT relay are presented. The phase voltage magnitudes for the BKY breaker (Figure 30f) and BKX breaker (Figure 30f) are presented with the trip signals of the BKY and BKX breakers (Figure 30h) for the SEL 700GT relay. It can be observed that the trip signal of the BKY breaker (grid side) and BKX breaker (wind farm side) are not activated by the VSL SC because the undervoltage of phase A was for a short duration generated by the SLG temporary electrical fault.

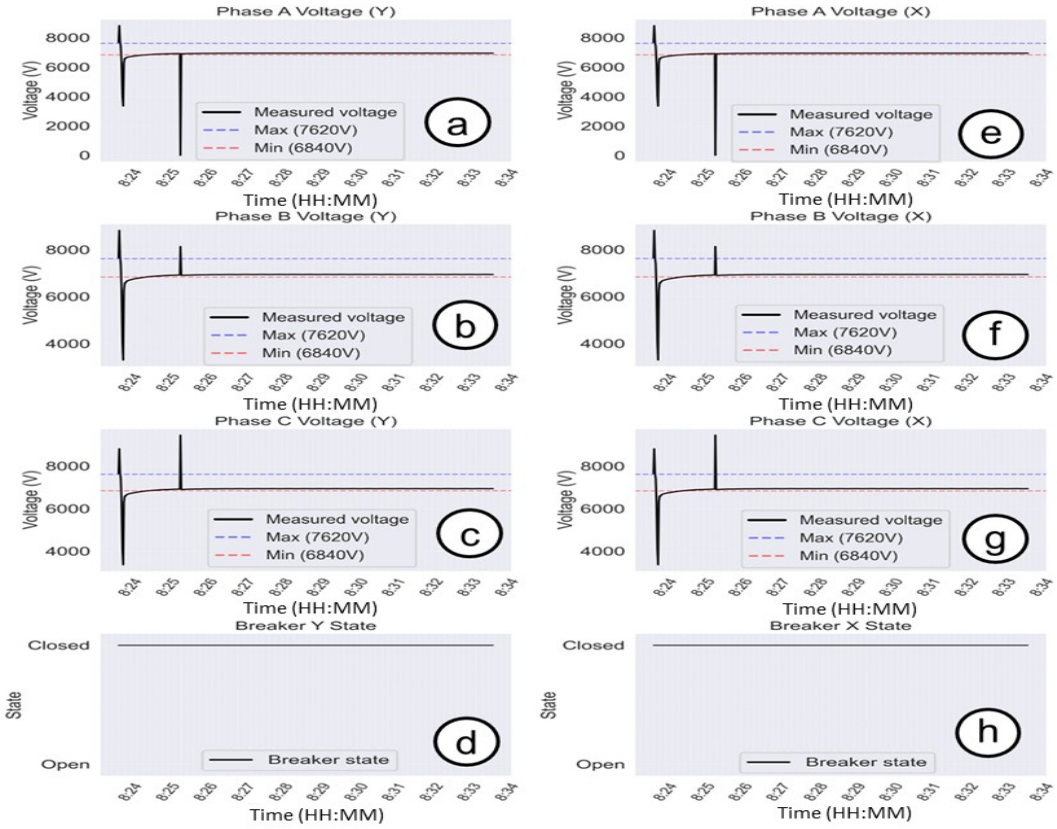


Figure 29. Temporary single-line-to-ground electrical fault at the end of the power line and no breaker operation (Test 5) results from Cyber Grid Guard.

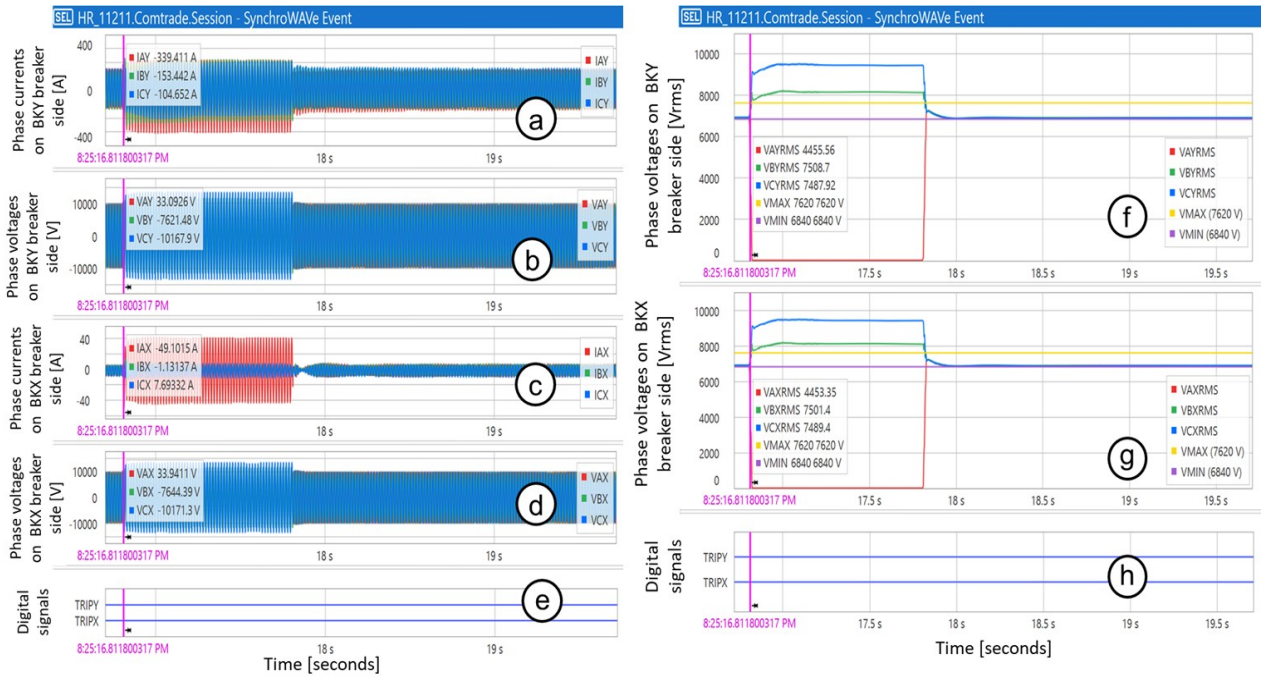


Figure 30. Temporary single-line-to-ground electrical fault at the end of the power line and no breaker operation (Test 5) results from the SEL 700GT relay.

8. DISCUSSIONS

8.1 DISCUSSIONS FOR TPF AND VSL SMART CONTRACTS

This study was performed by building experimental models for the TPF and VSL SCs, based on testing different use case scenarios like normal situation tests and temporary electrical fault tests. The presented results from the CGGS, real-time simulator, and SEL 700GT relay were collected for comparing the behavior of the measured TPF and VSL in the POI between the grid side and wind farm side. The most important objective was to evaluate the SCs for the TPF and VSL applications without performing a statistical analysis but rather comparing the measured phase voltage magnitudes, TPF values, and breaker trip signals; results from the TPF (Figure 18–Figure 21) and VSL (Figure 22–Figure 30) SCs were compared without quantifying the data of the results. The plots from the tests for the CGGS, real-time simulator, and SEL 700GT relay validated the TPF and VSL SC flowcharts (Figure 12 and Figure 14, respectively).

In the TPF from the real-time simulator (Figure 18b and Figure 19b) under normal situations (Tests 1 and 2), the TPF was observed to have swing behavior for the breaker at the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}). In Tests 1 and 2 (Table 5) for the TPF SC, all breakers initially were closed in the electrical grid (Figure 6), and a swing TPF ±1.00 was measured for a couple of seconds when the wind farm was connected. This situation resulted from a transient event generated by the connection of the wind farm (six 1.5 MW wind turbines) to feed the main loads on the grid side. The transient events of Tests 1 and 2 presented a ±1.00 variation of the TPF sign that is referred to a power swing phenomenon because of power system oscillations related to the reactive and active power flows along the power lines of the simulated electrical grid (Figure 6). A transient events remaining for a long period could have an effect on power grid reliability by operating the breakers from the protective relays for nondesired transient events, generating possible protective relay misoperations because of out-of-range time windows calibrated for protection functions like over/under voltage/frequency settings.

The TPF SC was evaluated by comparing test results for the normal situations (Figure 18 and Figure 19) and temporary electrical faults (Figure 20 and Figure 21). The normal situation (Test 1 of Table 5) offered a good TPF between +0.90 and +1.00 (grid side) for more than 400 s. In the POI between the grid side and wind farm side, the BKX (wind farm side) breaker was not operated by the TPF SC (Figure 18). The other normal situation (Test 2 of Table 5) had a poor TPF smaller than +0.90 (grid side) for more than 400 s. In the POI between the grid side and wind farm side, the BKX breaker (wind farm side) was opened by the TPF SC (Figure 19). In the temporary electrical fault tests (Tests 3 and 4 of Table 5), the test scenarios were performed at the 3LG (Figure 20) and SLG (Figure 21) faults located at the end of the power line on the grid side (Figure 6). The temporary electrical faults had a duration of 60 cycles (1 s) for Tests 3 and 4 (Table 5). The TPF on the grid side had shown a peak of a short duration for the 3LG (Figure 20a) and SLG (Figure 21a) temporary electrical fault tests, but the TPF variation occurred over a period of less than 400 s, and the TPF SC did not operate the BKX breaker (wind farm side) of the SEL 700GT relay in the POI.

In the TPF SC test scenarios for the temporary electrical fault test (Figure 20 and Figure 21), the measured TPF of the grid side (TPF_{BKY}) at fault states had shown a high and low swing value for the 3LG (Figure 20a) and SLG (Figure 21a) faults, respectively. These values were measured because the 3LG fault had a TPF that affected all phases, and the SLG fault had a TPF that affected one phase. In this analysis, the measured TPF of the grid side (TPF_{BKY}) and wind farm side (TPF_{BKX}) at fault states from the real-time simulator (Figure 20a and Figure 21a) have better resolution than the CGGS (Figure 20b and Figure 21b). The reason for this was that the real-time simulator used a time step of 50 µs to calculate the TPF values, and the CGGS collected the three-phase power factors (PF3X and PF3Y) from the SEL 700GT relay using IEC 61850 GOOSE messages. However, if a protective relay with time domain

protocol will be connected to the CGGS using DLT and SCs, the accuracy of measurements could be improved for the CGGS. The implementation of sample value protocols in measurements at the POI could enhance power system applications for the CGGS using DLT and SCs.

In the VSL SC test scenarios for the LL (Figure 27 and Figure 28) and SLG (Figure 29 and Figure 30) temporary electrical faults, the measured phase voltage magnitudes for the faulted phases dropped below 6,840 V (Figure 27a, Figure 27b, and Figure 29a) at the fault states. This happened during a period of approximately 1 s (60 cycles) that was the duration of the temporary electrical faults. The LL and SLG temporary electrical fault tests had shown how the VSL SC did not operate the breakers when the phase voltage magnitudes were not between 6,840 and 7,620 V for a short duration. This is similar to a temporary electrical fault that could generate an undervoltage situation on the grid side (BKY) and wind farm–side (BKX) breakers for only some cycles until the temporary electrical faults cleared themselves. The temporary electrical fault tests validated the VSL SC for the no breaker operation conditions during anomaly events for a short period. However, the tests for normal operation with extra inductive (Figure 23) and capacitive (Figure 25) loads show the validation of the VSL SC for the breaker operation conditions, observing how the breaker on the wind farm side (BKX) was tripped to keep the phase voltage magnitudes on the grid side between 6,840 and 7,620 V for the extra inductive (Figure 24) and capacitive (Figure 26) loads.

In the VSL SC, the recorded events from the SEL 700GT relay were collected for Tests 2–5 (Table 6). In Tests 2 and 3 (Table 6), the relay events were recorded when the BKX breaker (wind farm side) was tripped for the extra inductive (Figure 24) and extra capacitive (Figure 26) load tests. In Tests 4 and 5 (Table 6), the relay events were recorded for the LL (Figure 28) and SLG (Figure 30) temporary electrical fault tests. These events were recorded by using the trigger-events signal system (Figure 4) that enabled recording the events in the SEL 700GT relay at a specific time, in this case when the temporary electrical faults were simulated. The recorded events of the SEL 700GT relay for the LL and SLG temporary electrical fault tests are shown in Figure 28 and Figure 30, respectively. The normal situation tests for the extra inductive load (Figure 24) and extra capacitive load (Figure 26) show the "*hh:mm:ss*" stamped times of "6:06:09" and "7:18:34," respectively, when the BKX breaker (wind farm side) was tripped, matching with the "*hh:mm*" stamped times collected from the plots of the CGGS, "6:06" (Figure 23) and "7:18" (Figure 25),respectively. The anomaly situation tests for the LL (Figure 28) and SLG (Figure 30) temporary electrical fault tests show the "*hh:mm:ss*" stamped times of "7:58:08" and "8:25:16," respectively, when the temporary electrical faults happened, matching with the "*hh:mm*" stamped times collected from the plots of the CGGS, "7:58" (Figure 27) and "8:25" (Figure 29), respectively. The matched stamped times between the recorded events from the SEL 700GT relay and CGGS validated the VSL SCs for the use case scenarios.

The experimental models for the TPF and VSL SCs were run in the electrical substation grid testbed of the Advanced Protection Laboratory in the Grid Research Integration and Deployment Center at ORNL. The simulated three-phase power system circuit was based on an electrical grid substation with a customer-owned wind farm (Figure 6) that simulated real devices (e.g., power lines, loads, power transformers, wind farm). This testbed platform used a simulation time step of 50 µs for the real-time simulation tests, with real IEDs (relays and meters) in-the-loop. Therefore, the noise (or harmonics) of the phase current/voltage signals for the wind farm interconnection and electrical fault states were simulated with good accuracy, representing a sampling frequency of 20 kHz for measuring up to 333 samples per cycle (166th harmonics).

In this research, the TPF SC flowchart (Figure 12) was designed based on the theory of the power factor definitions [27–28, 32–33, 45]. But these terms and conditions were insufficient because the boundaries of the TPF SC also need to consider the behavior of the SEL 700GT relay. The TPF SC approach was based on measuring the unity TPF when the breakers were opened for the SEL 700GT relay and the IEEE

or/and IEC conventional power factor signs (IEEE was used for the SEL 700GT relay). Many of the studies for power grid applications with SCs were assessed with software simulations [50–53] that did not use real IEDs in-the-loop, and the software simulation approaches could result in issues for verifying applicability to realistic scenarios. In this study, the TPF SC was validated using a real-time simulator with an SEL 700GT relay-in-the-loop; this validation using real protective relays with substation protocols is critical to properly assess the proposed SCs with DLT for POI between the grid side and wind farm side.

The TPF and VSL SCs are performed as backup methods to reduce reactive losses and improve phase voltage magnitudes on the grid side, and they are performed before using load shedding, capacitor bank, and transformer/load tap changer techniques from the grid side. Therefore, the TPF and VSL SCs need a delay time window for being operated after using these traditional methods for improving the power quality. The time for operating the TPF and VSL SCs will depend on the needed time for operating the traditional power quality methods (load shedding, capacitor bank and transformer/load tap changers) on the grid side. Also, the time window will depend on the maximum time limit applied for the protection elements (under/over voltage and under/over frequency) of the protective relays, based on the complexity of the electrical grid. In this study, the TPF and VSL SCs used a time window of 450 s, but a different time window could be set, depending on the protection and control systems of the electrical grid.

In the TPF and VSL SCs, although DLT is capable of low latency of less than 1,000 ms for 10 transactions per second [54], the use of DLT may introduce additional computational overhead and network jitter that could affect real-time performance. This overhead could include the time taken for transaction preparation, consensus algorithms, and data replication across nodes [55], which could cause concerns about the scalability and real-time performance of large networks. However, by using the CGGS as a backup method, the performance and scalability issues for DLT could introduce delays in processing commands such as the trip of breakers at fault states. To mitigate these concerns, optimizations and enhancements can be applied to the DLT platform, such as sharding or partitioning the network to reduce transaction load on individual nodes [56] or employing lightweight consensus algorithms [57]. As an example, Hyperledger Fabric is a permissioned DLT that allows all participants to be authenticated and known. This situation eliminated the need for using mechanisms designed to prevent malicious behavior in open networks. Additionally, integrating DLT with existing communication protocols like GOOSE messages may require careful design to ensure timely delivery of time-critical commands because the transfer time for the trip command in GOOSE messages must be within 3 ms [58]. The CGGS with SCs is a useful tool for measuring the TPFs and phase voltage magnitudes in the POI between the grid side and customer-owned wind farm side, notably to operate breakers as a backup alternative through the protective relays using SCs with DLT.

Electrical grids could be vulnerable to severe weather events and cyberattacks, threating stability and operation that could result in severe costs to the grid operator. Cyberattacks can generate vulnerabilities in electrical grids with DERs, affecting integrity, confidentiality, availability, and accountability. Integrity attacks, such as false data injection [59], can lead to unauthorized modifications of field measurement data, potentially causing cascading failures in the electric grid [15]. However, severe weather events are the primary cause of large power outages, particularly in electrical distribution systems. These events, such as hurricanes and winter storms, have led to significant economic damages, with up to 90% of power failures attributed to disruptions in the distribution system [60]. The implementation of DLT with SCs for monitoring TPF and phase voltage magnitudes in the POI has an important economic benefit because of the possibility of enhancing cybersecurity and reliability. The immutability of blockchain ensures that all data transactions are secure and tamper-evident, which reduces the risk of fraudulent activities and operational errors [61]. This increased data security can lead to cost savings by minimizing the need for additional manual oversight and reducing potential downtime due to cyber threats [15]. DLT also offers a resilient platform that allows for continuous operation even in the event that individual components fail.

This translates to low operational costs because it reduces the frequency and impact of power outages. Although initial investments in setting up a DLT-based system can be substantial in both cost and complexity [19], the long-term benefits of reduced maintenance costs, enhanced security, and improved grid resilience make this approach economically viable for grid operators and utilities.

This report presents a novel method based on securing data between an utility grid and a customer-owned wind farm. Also, the TPF and VSL SCs implemented in this study, applied with the CGGS and SEL 700GT relay—for controlling breakers at the POI between the grid side and wind farm side—were based on SCs with multiple boundaries represented by the operation time, breaker states, and power quality conditions. They obtained TPF and phase voltage magnitude ranges of 0.90–1.00 and 6,840–7,620 V, respectively, at the grid side.

Future research will address the operation of SC, studying the behavior and assessment of the load-side power factor versus the effects of the grid-side power factor by using the SCs with DLT in the CGGS. The TPF and VSL SCs will be integrated into an adaptive protection communication scheme called Mirrored Bits for selecting different relays' setting groups. Consideration will be given to the TPF and VSL SCs' control of the breaker states of the point of common coupling device (SEL 700GT relay), defining different circuits paths (radial and nonradial power systems) and, consequently, different inverse time overcurrent and directional settings needed for all relays. Finally, the number of use case test scenarios with multiple DERs will be increased, not only focusing on power factor deviations for normal situations and temporary electrical faults but also covering extreme power grid conditions like cyberattacks and/or protective relay misoperations. This study will assess CGGS robustness, evaluating real-time performance and scalability of the SCs using DLT in large electrical grids.

9. CONCLUSIONS

This report presented a novel employment of a CGGS using DLT with TPF and VSL SCs. The method represents a modern electrical grid application using an SEL 700GT relay in the POI between the utility grid and a customer-owned wind farm to protect the integrity of shared data from the relay in the POI between the grid side and wind farm side. The TPF and VSL SCs were implemented using flowcharts and boundary conditions that were satisfactorily assessed with a CGGS using a real-time simulator and protective relays in-the-loop.

The TPF SC was assessed satisfactorily for different scenarios such as normal situation tests and temporary electrical fault tests. A poor TPF was defined as smaller than +0.90 (grid side) during more than 400 s, whereas a good TPF was between +0.90 and +1.00 on the grid side for more than 400 s. Operation of breakers in the POI was controlled by the CGGS and SEL 700GT relay. For the normal situation tests, the breakers in the POI were operated by the CGGS and SCs according to the boundaries and logic conditions of the TPF SC flowchart. In the anomaly tests, temporary electrical faults resulted in a swing TPF for a short duration on the grid side, but the TPF SC did not operate the breakers of the SEL 700GT relay because the temporary electrical fault states were set at 1 s (60 cycles).

The VSL SC was assessed satisfactorily for different test scenarios. In normal operation, a good voltage range was defined between 6,840 and 7,620 V without operating the breaker on the wind side. However, for an extra inductive and capacitive load on the wind farm side, the phase voltage magnitudes on the grid side were outside the range of 6,840–7,620 V, and the VSL SC tripped the wind farm side breaker after 400 s. Also, the VSL SC was assessed to temporary electrical faults, observing that phase voltage magnitudes outside the range of 6,840–7,620 V didn't operate the wind farm side breaker for short-duration anomaly events.

The TPF SC improved power quality and consequently reduced power line losses on the grid side. The TPF limit range was defined between +0.9 and +1.0 on the grid side. The VSL SC kept the phase voltage magnitudes on the grid side in the range of 6,840–7,620 V for 7.2 kV based on the ANSI C84.1 standard. The power quality was improved on the grid side by implementing the TPF and VSL SCs using a CGGS with DLT. The CGGS secured the data and breaker control commands between the utility grid and customer-owned wind farm. The operation of the breakers in the POI between the grid side and wind farm side was controlled by the TPF and VSL SCs. Data generated by the SEL 700GT relay and collected from the CGGS were used to perform the TPF and VSL SCs satisfactorily. Finally, the comparison of the events from the CGGS, real-time simulator, and SEL 700GT relay was used to assess the TPF and VSL SCs.

10. REFERENCES

[1] Piesciorovsky EC, Hahn G, Borges Hink R, Werth A. "Total Power Factor Smart Contract with Cyber Grid Guard Using Distributed Ledger Technology for Electrical Utility Grid with Customer-Owned Wind Farm." *Electronics*. 2024; 13(20): 4055. https://doi.org/10.3390/electronics13204055.

[2] Rizwan M, Hong L, Waseem M, Ahmad S, Sharaf M, Shafiq M. "A Robust Adaptive Overcurrent Relay Coordination Scheme for Wind-Farm-Integrated Power Systems Based on Forecasting the Wind Dynamics for Smart Energy Systems." *Applied Sciences*. 2020; 10(18): 6318. https://doi.org/10.3390/app10186318.

[3] Naheed KS, Louki S, Ghedira-Guegan C, Benkhelifa E, Bani-Hani A. "Blockchain smart contracts: Applications, challenges, and future trends." *Peer-to-Peer Networking and Applications* 2021; 14: 2901–2925.

[4] Foti M, Vavalis M. "What blockchain can do for power grids?" *Blockchain: Research and Applications* 2021; 2(1): 1–14.

[5] Bellaj B, Ouaddah A, Bertin E, Crespi N and Mezrioui A. "Drawing the Boundaries Between Blockchain and Blockchain-Like Systems: A Comprehensive Survey on Distributed Ledger Technologies." *Proceedings of the IEEE*. 2024; 112 (3): 247–299. https://doi.org/10.1109/JPROC.2024.3386257.

[6] AlSobeh AMR. "OSM: Leveraging model checking for observing dynamic behaviors in aspect-oriented applications." *Online Journal of Communication and Media Technologies*. 2023; 13(4) e202355: 1–18. https://doi.org/10.30935/ojcmt/13771.

[7] AlSobeh AMR and Magableh AA, "BlockASP: A Framework for AOP-Based Model Checking Blockchain System." *IEEE Access*, 2023; 11: 115062–115075. https://doi.org/10.1109/ACCESS.2023.3325060.

[8] Hong Z, Guo S, Zhou E, Chen W, Huang H, and Zomaya A. "GriDB: Scaling Blockchain Database via Sharding and Off-Chain Cross-Shard Mechanism." *Proceedings of the VLDB Endowment*. 2023; 16(7): 1685–1698. https://doi.org/10.14778/3587136.3587143.

[9] Li Y, Wang J, and Zhang H, "A survey of state-of-the-art sharding blockchains: Models, components, and attack surfaces." *Journal of Network and Computer Applications*. 2023; 217: 103686–103686. https://doi.org/10.1016/j.jnca.2023.103686.

[10] Gorzny J and Derka M. "A Rollup Comparison Framework." arXiv.org, 2024. https://doi.org/10.48550/arXiv.2404.16150 [accessed 1 April 2025].

[11] Thibault LT, Sarry T, and Hafid AS. “Blockchain Scaling using Rollups: A Comprehensive Survey.” *IEEE Access*. 2022; 20: 93039-93054. https://doi.org/10.1109/access.2022.3200051.

[12] Cai, T. et al. “On-Chain and Off-Chain Scalability Techniques” chapter. Editors: Chen, W., Zheng, Z., Huang, H. *Blockchain Scalability*. Springer, Singapore. https://doi.org/10.1007/978-981-99-1059-5_3.

[13] Liu Y, Lu Q, Zhu L, Paik HY, and Staples M. “A Systematic Literature Review on Blockchain Governance.” arXiv.org, 2021. https://doi.org/10.48550/arXiv.2105.05460 [accessed 1 April 2025].

[14] Barceló E, Dimić-Mišić K, Imani M, Brkić VS, Hummel M, and Gane P. “Regulatory Paradigm and Challenge for Blockchain Integration of Decentralized Systems: Example—Renewable Energy Grids.” *Sustainability*. 2023; 15(3), 2571: 1–27. https://doi.org/10.3390/su15032571.

[15] Zhuang P, Zamir T, and Liang H. “Blockchain for Cyber Security in Smart Grid: A Comprehensive Survey.” *IEEE Transactions on Industrial Informatics*. 2020; 17(1): 3–19. https://doi.org/10.1109/tii.2020.2998479.

[16] Chu H, Zhang P, Dong H, Xiao Y, Ji S, and Li W. “A survey on smart contract vulnerabilities: Data sources, detection and repair.” *Information and Software Technology*. 2023; 159(2023) 107221: 1–17. https://doi.org/10.1016/j.infsof.2023.107221.

[17] Kolb J, AbdelBaky M, Katz RH, and Culler DE. “Core Concepts, Challenges, and Future Directions in Blockchain.” ACM Computing Surveys (CSUR). 2020; 53(1): 1–39. https://doi.org/10.1145/3366370.

[18] Yap KY, Chin HH, and Klemeš JJ. “Blockchain technology for distributed generation: A review of current development, challenges and future prospect.” *Renewable and Sustainable Energy Reviews*. 2023; 175(2023),113170: 1–22. https://doi.org/10.1016/j.rser.2023.113170.

[19] Nour M, Chaves-Avila JP, and Sanchez-Miralles A. “Review of Blockchain Potential Applications in the Electricity Sector and Challenges for Large Scale Adoption.” *IEEE Access*. 2022; 10: 47384–47418. https://doi.org/10.1109/access.2022.3171227.

[20] Ji Q, Hu C, Duan Q, Huang C, and Zhao X. “Decentralized power grid fault traceability system based on internet of things and blockchain technology.” *Frontiers in Energy Research*. 2023; 10: 1–15. https://doi.org/10.3389/fenrg.2022.861321.

[21] Piesciorovsky EC, Hahn G, Borges Hink R, Werth A, and Lee A. “Electrical substation grid testbed for DLT applications of electrical fault detection, power quality monitoring, DERs use cases and cyber-events.” *Energy Reports*. 2023; 10: 1099–1115. https://doi.org/10.1016/j.egyr.2023.07.055.

[22] He H, et al. “Joint Operation Mechanism of Distributed Photovoltaic Power Generation Market and Carbon Market Based on Cross-Chain Trading Technology.” *IEEE Access*. 2020; 8: 66116–66130.

[23] Bokkisam HR, Singh S, Acharya RM, Selvan MP. “Blockchain-based peer-to-peer transactive energy system for community microgrid with demand response management.” *CSEE J. Power Energy Syst*. 2022; 8(1): 198–211.

[24] Liang G, Weller SR, Luo F, Zhao J, Dong ZY. “Distributed Blockchain-Based Data Protection Framework for Modern Power Systems Against Cyber Attacks.” I*EEE Trans. Smart Grid*. 2019; 10(3): 3162–3173.

[25] Mnatsakanyan A, Albeshr H, Al Marzooqi A, Bilbao E. “Blockchain-Integrated Virtual Power Plant Demonstration.” 2020 2nd International Conference on Smart Power and Internet Energy Systems. 2020; Bangkok, Thailand: 172–175.

[26] Cioara T, Antal M, Mihailescu VT, Antal CD, Anghel IM, Mitrea D. “Blockchain-Based Decentralized Virtual Power Plants of Small Prosumers.” *IEEE Access*. 2021; 9: 29490–29504.

[27] “SEL 700G Generator and Intertie Protection Relays Instruction Manual.” Schweitzer Engineering Laboratories Inc. https://selinc.com/products/700G/docs/ [accessed 1 April 2025].

[28] “PowerLogic™ PM5500/PM5600/PM5700 series User Manual.” Schneider Electric, document number HRB1684301, version 16, December 31, 2023: 1–238. https://www.se.com/us/en/download/document/HRB1684301/ [accessed 1 April 2025].

[29] Piesciorovsky EC, Borges Hink R, Werth A, Hahn G, Lee A, Richards J, Polsky Y. *Assessment of the Electrical Substation-Grid Test Bed with Inside/Outside Devices and Distributed Ledger.* ORNL/TM-2022/1840, Oak Ridge, Tennessee: Oak Ridge National Laboratory; 2022; 1–87.

[30] Piesciorovsky EC, Borges Hink R, Werth A, Hahn G, Lee A, and Polsky Y. “Assessment and Commissioning of Electrical Substation Grid Testbed with a Real-Time Simulator and Protective Relays/Power Meters in the Loop.” *Energies, Real-Time Simulation of Power Systems and Power Hardware-in-the-Loop*, 2023;16(11) 4407: 1–26. https://doi.org/10.3390/en16114407.

[31] Borges Hink R, Hahn G, Werth A, Piesciorovsky EC, Lee A, Monday W, Polsky Y. *Oak Ridge National Laboratory Pilot Demonstration of an Attestation and Anomaly Detection Framework Using Distributed Ledger Technology for Power Grid Infrastructure*. ORNL/TM-2022/2527, Oak Ridge, Tennessee: Oak Ridge National Laboratory, 2022; 1–56.

[32] IEEE Std 1459TM- 2010, Standard Definitions for the Measurement of Electric Power Quantities Under Sinusoidal, Nonsinusoidal, Balanced, or Unbalanced Conditions. *IEEE Power & Energy Society*, 2010; 1–40.

[33] “Introduction to Power Factor.” Wyandotte Municipal Services, Revision September 15, 2004; 1–2. https://cms2.revize.com/revize/wyandottems/document_center/Electric/PowerFactor.pdf [accessed 1 April 2025].

[34] “1C.2.1—Voltage Level and Range.” *Engineering Handbook*. Vol. 1-General; Part C-Power Quality, Rocky Mountain Power and Pacific Power, November 2024, pp. 1–6. https://www.pacificpower.net/content/dam/pcorp/documents/en/pp-rmp/power-quality-Std.s/1C_2_1_PF.pdf [accessed 1 April 2025].

[35] Edvard C. “Six common bus configurations in substations up to 354 kV.” *Electrical Engineering Portal*, March 18, 2019. https://electrical-engineering-portal.com/bus-configurations-substations-345-kv [accessed 1 April 2025].

[36] “SEL 735 Power Quality and Revenue Meter Instruction Manual.” Schweitzer Engineering Laboratories Inc. https://selinc.com/products/735/docs/ [accessed 1 April 2025].

[37] “SEL 421-4, -5, Protection, Automation, and Control System Instruction Manual.” Schweitzer Engineering Laboratories Inc. https://selinc.com/products/421/docs/ [accessed 1 April 2025].

[38] “SEL 451-5 Protection, Automation, and Bay Control System and SEL 400 Series Relays Instruction Manual.” Schweitzer Engineering Laboratories Inc. https://selinc.com/products/451/docs/ [accessed 1 April 2025].

[39] “SEL 351S Protection System Instruction Manual.” Schweitzer Engineering Laboratories Inc. https://selinc.com/products/351S/docs/ [accessed 1 April 2025].

[40] Du B, He Y, An B and Zhang C. “Remaining Useful Performance Estimation for Complex Analog Circuit Based on Maximal Information Coefficient and Bidirectional Gate Recurrent Unit.” *IEEE Access*. 2020; 8: 102449–102466.

[41] Piesciorovsky EC, Smith T, Mukherjee SK, Marshall MW. "A Generic Method for Interfacing IEDs using Low Voltage Interfaces to Real-time Simulators with Hardware in the Loop." *Electric Power Systems Research*. 2021; 199 (107431): 1–13.

[42] Piesciorovsky EC, Schulz NN. "Comparison of Programmable Logic and Setting Group Methods for Adaptive Overcurrent Protection in Microgrids." *Electric Power Systems Research*. 2017; 151: 273–282.

[43] Piesciorovsky EC, Schulz NN. "Fuse Relay Adaptive Overcurrent Protection Scheme for Microgrid with Distributed Generators." *The Institution of Engineering and Technology Journal, IET Generation, Transmission and Distribution*. 2016; 11 (2): 540–549.

[44] "Total Clearing Time—Current Characteristic Curves Positrol® Fuse Links–S&C "T" Speed (TCC 170-6-2)." S&C Electric Company. https://www.sandc.com/en/products--services/products/positrol-fuse-links/ [accessed 1 April 2025].

[45] "Electric Choice Understanding Power Factor." DTE Energy. https://www.dteenergy.com/content/dam/dteenergy/deg/website/business/service-request/electric/electric-choice/powerFactor.pdf [accessed 1 April 2025].

[46] Voloh I, Ernst T. "Review of Capacitor Bank Control Practices." GE Grid Solutions, 72nd Annual Conference for Protective Relay Engineers, Texas A&M University, College Station, TX, USA, March 25–28, 2019; 1–13.

[47] New WC. "Load Shedding, Load Restoration and Generator Protection Using Solid-state and Electromechanical Underfrequency Relays." GET-6449, GE Power Management, 215 Anderson Avenue, Markham, Ontario, Canada. https://www.gevernova.com/grid-solutions/products/applications/get6449.pdf [accessed 1 April 2025].

[48] ANSI/IEEE Standard C37.2, Standard for Electrical Power System Device Function Numbers, Acronyms, and Contact Designations. Substations Committee and the Power Systems Relaying Committee, 3 October 2008.

[49] Hosseinzadeh H. "Distribution System, Protection ES586B: Power System Protection." University of Western Ontario, May 5, 2008; 1–16. https://www.eng.uwo.ca/people/tsidhu/documents/es586b-hesam%20hosseinzadeh-250441131.pdf. [accessed 1 April 2025].

[50] Zhao C, Han D, Li C, Wang H. "A Blockchain Consensus Mechanism to Optimize Reputation-Based Distributed Energy Trading in Urban Energy System." *IEEE Access*. 2024; 12: 53698–53712. https://ieeexplore.ieee.org/document/10496579.

[51] Yao S, Tian X, Chen J, Xiong Y. "Privacy preserving distributed smart grid system based on Hyperledger Fabric and Wireguard." *International Journal of Network Management*. 2023; 33(3): 1–18. https://doi.org/10.1002/nem.2193.

[52] Zhao Z, Guo J, Luo X, Xue J, Sing Lai C, Xu Z, Lei Lai L. "Energy Transaction for Multi-Microgrids and Internal Microgrid Based on Blockchain." *IEEE Access*. 2020; 8: 144362–144372. https://ieeexplore.ieee.org/document/9159650.

[53] Sikeridis D, Bidram A, Devetsikiotis M, Reno MJ. "A Blockchain-Based Mechanism for Secure Data Exchange in Smart Grid Protection Systems." 2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA. 2020; 1–6. https://ieeexplore.ieee.org/document/9045368.

[54] Dreyer J, Fischer M, Tönjes R. "Performance Analysis of Hyperledger Fabric 2.0 Blockchain Platform." *Proceedings of the Workshop on Cloud Continuum Services for Smart IoT Systems* 2020; 32–38. https://doi.org/10.1145/3417310.3431398.

[55] Zhou Y, et al. “Application of Distributed Ledger Technology in Distribution Networks.” *Proceedings of the IEEE*. 2022; 110(12): 1963–1975. https://doi.org/10.1109/JPROC.2022.3181528.

[56] Liu Y, et al. “An overview of blockchain smart contract execution mechanism.” *Journal of Industrial Information Integration*. 2024; 41(2024) 100674: 1–29. https://doi.org/10.1016/j.jii.2024.100674.

[57] Stefanescu D, Montalvillo L, Galán-García P, Unzilla J, and Urbieta A. “A Systematic Literature Review of Lightweight Blockchain for IoT.” *IEEE Access*. 2022; 10: 123138–123159. https://doi.org/10.1109/ACCESS.2022.3224222.

[58] Van Rensburg M, Dolezilek D, Dearien J. “Case Study: Using IEC 61850 Network Engineering Guideline Test Procedures to Diagnose and Analyze Ethernet Network Installations.” PAC World Africa Conference, Johannesburg, South Africa, November 12–13; 2015; 1–10.

[59] Habib AA, Hasan MK, Alkhayyat A, Islam S, Sharma R, and Alkwai LM. “False data injection attack in smart grid cyber physical system: Issues, challenges, and future direction.” *Computers and Electrical Engineering*. 2023; 107(2023), 108638: 1–16. https://doi.org/10.1016/j.compeleceng.2023.108638.

[60] Daeli A and Mohagheghi S. “Power Grid Infrastructural Resilience Against Extreme Events.” *Energies*. 2023; 16(1)64: 1–17. https://doi.org/10.3390/en16010064.

[61] Immaniar D, Aryani AA, and Ula SZ. “Challenges Smart Grid in Blockchain Applications.” *Blockchain Frontier Technology*. 2023; 2(2): 1–10. https://doi.org/10.34306/bfront.v2i2.150.

APPENDIX A. CODES

Note: The codes in this appendix continue to being updated and improved in the Cyber Grid Guard system.

A-1. BREAKER CODE OF TOTAL POWER FACTOR SMART CONTRACT

a

```
#***********************************
# Authors: Aaron W. Werth, Emilio Piesciorovsky,
# and Gary Hahn
# Program: operate_x.py
# Description: This program will operate
# the breaker X of the relay based
# on the previous state (close or open)
# with the IP address 192.168.100.25.
#***********************************

import time
import sys
import telnetlib
HOST = "192.168.100.25"
PORT = "23"
with telnetlib.Telnet(HOST, PORT) as telnetObj:
    time.sleep(0.5)
    message = ("acc\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("OTTER\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("2ac\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAIL\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("OPEN X\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("Y\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)

print("Operating breaker X!")
```

b

```
#***********************************
# Authors: Aaron W. Werth, Emilio Piesciorovsky,
# and Gary Hahn
# Program: operate_y.py
# Description: This program will operate
# the breaker Y of the relay based
# on the previous state (close or open)
# with the IP address 192.168.100.25.
#***********************************

import time
import sys
import telnetlib
HOST = "192.168.100.25"
PORT = "23"
with telnetlib.Telnet(HOST, PORT) as telnetObj:
    time.sleep(0.5)
    message = ("acc\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("OTTER\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("2ac\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAIL\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("OPEN Y\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("Y\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)

print("Operating breaker Y!")
```

A-2. MAIN CODE OF TOTAL POWER FACTOR SMART CONTRACT

a

```
package main

import (
        "encoding/json"
        "fmt"
        "log"
        "strconv"
        "time"

        "github.com/hyperledger/fabric-contract-api-go/contractapi"
)

// DeviceRecord represents the structure for a device record
type DeviceRecord struct {
        DeviceID     string    `json:"deviceID"`
        VariableName string    `json:"variableName"`
        Value        float64   `json:"value"`
        Timestamp    time.Time `json:"timestamp"`
}

// ConditionRecord represents the structure for a condition trigger record
type ConditionRecord struct {
        DeviceID        string    `json:"deviceID"`
        ConditionNumber int       `json:"conditionNumber"`
        Timestamp       time.Time `json:"timestamp"`
}

// EventData represents the structure for an emitted event data record
type EventData struct {
        DeviceID        string    `json:"deviceID"`
        ConditionNumber  int       `json:"conditionNumber"`
        BreakerOperation string    `json:"breakerOperation"`
        StartTimestamp   time.Time `json:"startTimestamp"`
        EndTimestamp     time.Time `json:"endTimestamp"`
}
```

```
// SmartContract provides functions for managing device records
type SmartContract struct {
        contractapi.Contract
}
```

b

```
func queryDeviceHistoryByVariableAndTimeRange(ctx contractapi.TransactionContextInterface, deviceID string, variableName string, startTime string, endTime string) ([]*DeviceRecord, int,
       queryString := fmt.Sprintf(`{
       "selector": {
          "deviceID": "%s",
          "variableName": "%s",
          "timestamp": {
             "$gte": "%s",
             "$lte": "%s"
          }
       },
       "sort": [{"timestamp": "asc"}]
    }`, deviceID, variableName, startTime, endTime)

       log.Printf("Querying device '%s' history by variable '%s' for time range: %s - %s", deviceID, variableName, startTime, endTime)

       resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)
       if err != nil {
              return nil, 0, err
       }
       defer resultsIterator.Close()
```

c

```
              var records []*DeviceRecord
              var count int

              for resultsIterator.HasNext() {
                     queryResponse, err := resultsIterator.Next()
                     if err != nil {
                            return nil, 0, err
                     }

                     var deviceRecord DeviceRecord
                     err = json.Unmarshal(queryResponse.Value, &deviceRecord)
                     if err != nil {
                            return nil, 0, err
                     }

                     records = append(records, &deviceRecord)
                     count++
              }

              return records, count, nil
    }

     func executeQuery(ctx contractapi.TransactionContextInterface, queryString string) ([][]byte, error) {
             resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)
             if err != nil {
                    return nil, err
             }
             defer resultsIterator.Close()

             var results [][]byte
             for resultsIterator.HasNext() {
                    response, err := resultsIterator.Next()
                    if err != nil {
                           return nil, err
                    }
                    results = append(results, response.Value)
             }

             return results, nil
     }
```

```
func (s *SmartContract) queryDeviceHistoryByTimeRange(ctx contractapi.TransactionContextInterface, deviceID string, startTime string, endTime string) ([]*DeviceRecord, error) {
	queryString := fmt.Sprintf(`{
	"selector": {
	    "deviceID": "%s",
	    "timestamp": {
	        "$gte": "%s",
	        "$lte": "%s"
	    }
	},
	"sort": [{"timestamp": "asc"}]
   }`, deviceID, startTime, endTime)

	log.Printf("Querying device history by time range: %s - %s", startTime, endTime)

	results, err := executeQuery(ctx, queryString)
	if err != nil {
		return nil, err
	}

    var deviceRecords []*DeviceRecord
    for _, result := range results {
		var deviceRecord DeviceRecord
		if err := json.Unmarshal(result, &deviceRecord); err != nil {
			return nil, fmt.Errorf("error unmarshalling DeviceRecord: %v", err)
		}
		deviceRecords = append(deviceRecords, &deviceRecord)
    }

	return deviceRecords, nil
  }
```

d

```
  func (s *SmartContract) queryLatestBreakerState(ctx contractapi.TransactionContextInterface, deviceID string, variableName string, timestamp time.Time) (bool, error) {
	queryString := fmt.Sprintf(`{
	"selector": {
	    "deviceID": "%s",
	    "variableName": "%s",
	    "timestamp": {
	        "$lte": "%s"
	    }
	},
	"sort": [
	    {
	        "timestamp": "desc"
	    }
	],
	"limit": 1
    }`, deviceID, variableName, timestamp.Format(time.RFC3339))

	log.Printf("Querying latest breaker state '%s' as of %s", variableName, timestamp.Format(time.RFC3339))

	resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)
	if err != nil {
		return false, fmt.Errorf("failed to execute query: %v", err)
	}
	defer resultsIterator.Close()

	if !resultsIterator.HasNext() {
		return false, fmt.Errorf("no breaker state found for device %s and variable %s", deviceID, variableName)
	}

	queryResult, err := resultsIterator.Next()
	if err != nil {
		return false, fmt.Errorf("failed to get query result: %v", err)
	}

	var deviceRecord DeviceRecord
	err = json.Unmarshal(queryResult.Value, &deviceRecord)
	if err != nil {
		return false, fmt.Errorf("failed to unmarshal device record: %v", err)
	}

	return deviceRecord.Value == 1, nil
}
```

e

```
// corresponds to condition:
// 1. (!breakerYOpen AND !breakerXOpen) AND (0.9 <= magYTotPF <= 1.0 AND 0.9 <= magXTotPF <= 1.0)
// if true, then no breaker operation
func checkCondition1(records1 []*DeviceRecord, records2 []*DeviceRecord, breakerXOpen bool, breakerYOpen bool) (bool, error) {
        // if no records are found, then the condition is not met
        if len(records1) == 0 || len(records2) == 0 {
                log.Printf("checkCondition1: evaluated false - no records found")
                return false, nil
        }

        if breakerYOpen || breakerXOpen {
                log.Printf("checkCondition1: evaluated false - breakerYOpen = %t, breakerXOpen = %t", breakerYOpen, breakerXOpen)
                return false, nil
        }

        device1Met := true
        for _, record1 := range records1 {
                if record1.Value < 0.9 || record1.Value > 1.0 {
                        log.Printf("checkCondition1: device1Met evaluated false - magXTotPF = %f", record1.Value)
                        device1Met = false
                        break
                }
        }

        device2Met := true
        for _, record2 := range records2 {
                if record2.Value < 0.9 || record2.Value > 1.0 {
                        log.Printf("checkCondition1: device2Met evaluated false - magYTotPF = %f", record2.Value)
                        device2Met = false
                        break
                }
        }

        return device1Met && device2Met, nil
}

// corresponds to condition:
// 2. ((!breakerYOpen AND breakerXOpen) OR (breakerYOpen AND !breakerXOpen)) AND (0.9 <= magYTotPF <= 1.0 OR 0.9 <= magXTotPF <= 1.0)
// if true, then no breaker operation
func checkCondition2(records1 []*DeviceRecord, records2 []*DeviceRecord, breakerXOpen bool, breakerYOpen bool) (bool, error) {
        // if no records are found, then the condition is not met
        if len(records1) == 0 || len(records2) == 0 {
                log.Printf("checkCondition2: evaluated false - no records found")
                return false, nil
        }


            if (breakerYOpen && breakerXOpen) || (!breakerYOpen && !breakerXOpen) {
                    log.Printf("checkCondition2: evaluated false - breakerYOpen = %t, breakerXOpen = %t", breakerYOpen, breakerXOpen)
                    return false, nil
            }

            device1Met := true
            for _, record1 := range records1 {
                    if record1.Value < 0.9 || record1.Value > 1.0 {
                            log.Printf("checkCondition2: device1Met evaluated false - magXTotPF = %f", record1.Value)
                            device1Met = false
                            break
                    }
            }

            device2Met := true
            for _, record2 := range records2 {
                    if record2.Value < 0.9 || record2.Value > 1.0 {
                            log.Printf("checkCondition2: device2Met evaluated false - magYTotPF = %f", record2.Value)
                            device2Met = false
                            break
                    }
            }

            return device1Met || device2Met, nil
    }

    // corresponds to condition:
    // 3. (!breakerYOpen AND !breakerXOpen) AND (-1.0 <= magYTotPF < 0.9 AND -1.0 <= magXTotPF < 0.9)
    // if true, then operate breaker BKX
    func checkCondition3(records1 []*DeviceRecord, records2 []*DeviceRecord, breakerXOpen bool, breakerYOpen bool) (bool, error) {
            // if no records are found, then the condition is not met
            if len(records1) == 0 || len(records2) == 0 {
                    log.Printf("checkCondition3: evaluated false - no records found")
                    return false, nil
            }

            if breakerYOpen || breakerXOpen {
                    log.Printf("checkCondition3: evaluated false - breakerYOpen = %t, breakerXOpen = %t", breakerYOpen, breakerXOpen)
                    return false, nil
            }
```

f

g

```
        device1Met := true
        for _, record1 := range records1 {
                if record1.Value < -1.0 || record1.Value >= 0.9 {
                        log.Printf("checkCondition3: device1Met evaluated false - magXTotPF = %f", record1.Value)
                        device1Met = false
                        break
                }
        }

        device2Met := true
        for _, record2 := range records2 {
                if record2.Value < -1.0 || record2.Value >= 0.9 {
                        log.Printf("checkCondition3: device2Met evaluated false - magYTotPF = %f", record2.Value)
                        device2Met = false
                        break
                }
        }

        return device1Met && device2Met, nil
}

// corresponds to condition:
// 4. (((!breakerYOpen AND breakerXOpen) OR (breakerYOpen AND !breakerXOpen)) AND (-1.0 <= magYTotPF < 0.9 OR -1.0 <= magXTotPF < 0.9)
// if true, then operate breaker BKX
func checkCondition4(records1 []*DeviceRecord, records2 []*DeviceRecord, breakerXOpen bool, breakerYOpen bool) (bool, error) {
        // if no records are found, then the condition is not met
        if len(records1) == 0 || len(records2) == 0 {
                log.Printf("checkCondition4: evaluated false - no records found")
                return false, nil
        }

        if (breakerYOpen && breakerXOpen) || (!breakerYOpen && !breakerXOpen) {
                log.Printf("checkCondition4: evaluated false - breakerYOpen = %t, breakerXOpen = %t", breakerYOpen, breakerXOpen)
                return false, nil
        }

        device1Met := true
        for _, record1 := range records1 {
                if record1.Value < -1.0 || record1.Value >= 0.9 {
                        log.Printf("checkCondition4: device1Met evaluated false - magXTotPF = %f", record1.Value)
                        device1Met = false
                        break
                }
        }


        device2Met := true
        for _, record2 := range records2 {
                if record2.Value < -1.0 || record2.Value >= 0.9 {
                        log.Printf("checkCondition4: device2Met evaluated false - magYTotPF = %f", record2.Value)
                        device2Met = false
                        break
                }
        }

        return device1Met || device2Met, nil
}

func checkTimeStateConditions(timestamp time.Time) bool {
        for i := 0; i <= 450; i++ {
                priorTimestamp := timestamp.Add(time.Duration(-i) * time.Second)

                // check if the current timestamp falls within November (wind turbine maintenance) or January (ice accumulation on blades)
                if priorTimestamp.Month() == time.November || priorTimestamp.Month() == time.January {
                        log.Printf("checkTimeStateConditions: evaluated true - month = %s", priorTimestamp.Month().String())
                        return true
                }

                // check if the current timestamp falls within the time period 10PM to 7AM UTC (excess noise site regulations)
                hour := priorTimestamp.UTC().Hour()
                if hour >= 22 || hour < 7 {
                        log.Printf("checkTimeStateConditions: evaluated true - hour = %d", hour)
                        return true
                }
        }

        return false
}

// AddOrUpdateRecord adds a new device record or updates an existing one
func (s *SmartContract) AddOrUpdateRecord(ctx contractapi.TransactionContextInterface, deviceID string, variableName string, value float64, timestamp string) error {
        ts, err := time.Parse(time.RFC3339, timestamp)
        if err != nil {
                return fmt.Errorf("invalid timestamp: %s", err.Error())
        }

        log.Printf("* BEGIN AddOrUpdateRecord for deviceID %s, variableName %s, value %f at %s ...", deviceID, variableName, value, ts.Format(time.RFC3339))
```

```
        deviceRecord := DeviceRecord{
                DeviceID:     deviceID,
                VariableName: variableName,
                Value:        value,
                Timestamp:    ts,
        }

        key, err := ctx.GetStub().CreateCompositeKey("DeviceRecord", []string{deviceID, variableName, ts.Format(time.RFC3339)})
        if err != nil {
                log.Println("* END AddOrUpdateRecord.")
                return fmt.Errorf("failed to create composite key: %s", err.Error())
        }

        deviceRecordAsBytes, err := json.Marshal(deviceRecord)
        if err != nil {
                log.Println("* END AddOrUpdateRecord.")
                return err
        }

        err = ctx.GetStub().PutState(key, deviceRecordAsBytes)
        if err != nil {
                log.Println("* END AddOrUpdateRecord.")
                return err
        }

        // Perform condition checks
        err = s.CheckConditions(ctx, deviceID, timestamp)
        if err != nil {
                // log the condition check error and proceed with the transaction
                log.Printf("Automatic condition check failed for device %s, variable %s at timestamp %s: %s", deviceID, variableName, timestamp, err.Error())
        }

        log.Println("* END AddOrUpdateRecord.")
        return nil
}

func (s *SmartContract) AddOrUpdateRecords(ctx contractapi.TransactionContextInterface, recordsJSON string) error {
        log.Printf("* BEGIN AddOrUpdateRecords for records: %v ...", recordsJSON)

        var records []DeviceRecord
        err := json.Unmarshal([]byte(recordsJSON), &records)
        if err != nil {
                log.Println("* END AddOrUpdateRecords.")
                return fmt.Errorf("failed to unmarshal records JSON: %v", err)
        }

       for _, record := range records {
              key, err := ctx.GetStub().CreateCompositeKey("DeviceRecord", []string{record.DeviceID, record.VariableName, record.Timestamp.Format(time.RFC3339)})
              if err != nil {
                     log.Println("* END AddOrUpdateRecords.")
                     return fmt.Errorf("failed to create composite key: %s", err.Error())
              }

              recordAsBytes, err := json.Marshal(record)
              if err != nil {
                     log.Println("* END AddOrUpdateRecords.")
                     return err
              }

              err = ctx.GetStub().PutState(key, recordAsBytes)
              if err != nil {
                     log.Println("* END AddOrUpdateRecords.")
                     return err
              }
       }

       err = s.CheckConditions(ctx, records[0].DeviceID, records[0].Timestamp.Format(time.RFC3339))
       if err != nil {
              // log the condition check error and proceed with the transaction
              log.Printf("Automatic condition check failed for device %s at timestamp %s: %s", records[0].DeviceID, records[0].Timestamp.Format(time.RFC3339), err.Error())
       }

       log.Println("* END AddOrUpdateRecords.")
       return nil
}

// QueryDeviceHistory queries the history of power factor-timestamp tuples for a given device ID and time range
func (s *SmartContract) QueryDeviceHistory(ctx contractapi.TransactionContextInterface, deviceID string, startTime string, endTime string) ([]*DeviceRecord, error) {
       return s.queryDeviceHistoryByTimeRange(ctx, deviceID, startTime, endTime)
}

func (s *SmartContract) QueryConditions(ctx contractapi.TransactionContextInterface, deviceID string) ([]ConditionRecord, error) {
       queryString := fmt.Sprintf(`{
       "selector": {
           "deviceID": "%s"
       },
       "sort": [{"timestamp": "desc"}]
   }`, deviceID)

       results, err := executeQuery(ctx, queryString)
       if err != nil {
              return nil, err
       }
```

```
        var conditions []ConditionRecord
        for _, conditionBytes := range results {
                var condition ConditionRecord
                if err := json.Unmarshal(conditionBytes, &condition); err != nil {
                        return nil, err
                }
                conditions = append(conditions, condition)
        }

        return conditions, nil
}

func (s *SmartContract) LogCondition(ctx contractapi.TransactionContextInterface, deviceID string, conditionNumber int) error {
        // use transaction timestamp because it is the same across all endorsements for a given transaction
        timestamp, err := ctx.GetStub().GetTxTimestamp()
        if err != nil {
                return fmt.Errorf("failed to get transaction timestamp: %v", err)
        }

        timestampTime := time.Unix(timestamp.Seconds, int64(timestamp.Nanos))

        conditionRecord := ConditionRecord{
                DeviceID:        deviceID,
                ConditionNumber: conditionNumber,
                Timestamp:       timestampTime,
        }

        conditionRecordAsBytes, err := json.Marshal(conditionRecord)
        if err != nil {
                return err
        }

        // unique key for storing the condition record
        key, err := ctx.GetStub().CreateCompositeKey("ConditionRecord", []string{deviceID, strconv.Itoa(conditionNumber), timestampTime.Format(time.RFC3339)})
        if err != nil {
                return err
        }

        return ctx.GetStub().PutState(key, conditionRecordAsBytes)
}


func triggerConditionEvent(ctx contractapi.TransactionContextInterface, deviceID string, conditionNumber int, breakerOperation string, startTimestamp, endTimestamp time.Time) error {
        eventData := EventData{
                DeviceID:        deviceID,
                ConditionNumber:  conditionNumber,
                BreakerOperation: breakerOperation,
                StartTimestamp:   startTimestamp,
                EndTimestamp:     endTimestamp,
        }

        eventDataBytes, err := json.Marshal(eventData)
        if err != nil {
                return fmt.Errorf("failed to marshal event data: %v", err)
        }

        eventName := fmt.Sprintf("Condition%dEvent", conditionNumber)
        log.Printf("Triggering event '%s' with data: %s", eventName, eventDataBytes)
        err = ctx.GetStub().SetEvent(eventName, eventDataBytes)
        if err != nil {
                return fmt.Errorf("failed to set event: %v", err)
        }

        return nil
}

// CheckConditions checks the specified conditions for a given device
func (s *SmartContract) CheckConditions(ctx contractapi.TransactionContextInterface, deviceID string, currentTimestamp string) error {
        currentTs, err := time.Parse(time.RFC3339, currentTimestamp)
        if err != nil {
                return fmt.Errorf("invalid current timestamp: %s", err.Error())
        }
        log.Printf("* BEGIN checking conditions for device %s at %s ...", deviceID, currentTs.Format(time.RFC3339))

        if checkTimeStateConditions(currentTs) {
                log.Println("* Time state conditions met, skipping condition checks.")
                log.Println("* END checking conditions.")
                return nil // return early if time state conditions are met
        }

        // query the most recent values of breakerXOpen and breakerYOpen
        breakerXOpen, err := s.queryLatestBreakerState(ctx, deviceID, "breakerXOpen", currentTs)
        if err != nil {
                return err
        }
        breakerYOpen, err := s.queryLatestBreakerState(ctx, deviceID, "breakerYOpen", currentTs)
        if err != nil {
                return err
        }
```

```
// query the last 450 seconds of records starting from the current timestamp for checking conditions
startTimestamp := currentTs.Add(-450 * time.Second)
records1, count1, err := queryDeviceHistoryByVariableAndTimeRange(ctx, deviceID, "magXTotPF", startTimestamp.Format(time.RFC3339), currentTs.Format(time.RFC3339))
if err != nil {
	return err
}
log.Printf("Found %d records for device %s, variable magXTotPF within the last 450 seconds", count1, deviceID)

records2, count2, err := queryDeviceHistoryByVariableAndTimeRange(ctx, deviceID, "magYTotPF", startTimestamp.Format(time.RFC3339), currentTs.Format(time.RFC3339))
if err != nil {
	return err
}
log.Printf("Found %d records for device %s, variable magYTotPF within the last 450 seconds", count2, deviceID)

condition1, err := checkCondition1(records1, records2, breakerXOpen, breakerYOpen)
if err != nil {
	return err
}
if condition1 {
	err = triggerConditionEvent(ctx, deviceID, 1, "none", startTimestamp, currentTs)
	if err != nil {
		return err
	}
	if err := s.LogCondition(ctx, deviceID, 1); err != nil {
		return err
	}
	log.Println("* END checking conditions.")
	return nil
}

condition2, err := checkCondition2(records1, records2, breakerXOpen, breakerYOpen)
if err != nil {
	return err
}
if condition2 {
	err = triggerConditionEvent(ctx, deviceID, 2, "none", startTimestamp, currentTs)
	if err != nil {
		return err
	}
	if err := s.LogCondition(ctx, deviceID, 2); err != nil {
		return err
	}
	log.Println("* END checking conditions.")
	return nil
}

	condition3, err := checkCondition3(records1, records2, breakerXOpen, breakerYOpen)
	if err != nil {
		return err
	}
	if condition3 {
		breakerOperation := "open_x"
		err = triggerConditionEvent(ctx, deviceID, 3, breakerOperation, startTimestamp, currentTs)
		if err != nil {
			return err
		}
		if err := s.LogCondition(ctx, deviceID, 3); err != nil {
			return err
		}
		log.Println("* END checking conditions.")
		return nil
	}

	condition4, err := checkCondition4(records1, records2, breakerXOpen, breakerYOpen)
	if err != nil {
		return err
	}
	if condition4 {
		breakerOperation := "open_x"
		err = triggerConditionEvent(ctx, deviceID, 4, breakerOperation, startTimestamp, currentTs)
		if err != nil {
			return err
		}
		if err := s.LogCondition(ctx, deviceID, 4); err != nil {
			return err
		}
		log.Println("* END checking conditions.")
		return nil
	}

	log.Println("* END checking conditions.")
	return nil
}

func main() {
	chaincode, err := contractapi.NewChaincode(&SmartContract{})
	if err != nil {
		fmt.Printf("Error create chaincode: %s", err.Error())
		return
	}

	if err := chaincode.Start(); err != nil {
		fmt.Printf("Error starting chaincode: %s", err.Error())
	}
}
```

A-3. BREAKER CODE OF VOLTAGE SERVICE LIMITS SMART CONTRACT

```
#***********************************
# Authors: Aaron W. Werth, Emilio Piesciorovsky,
# and Gary Hahn
# Program: operate_x.py
# Description: This program will operate
# the breaker X of the relay based
# on the previous state (close or open)
# with the IP address 192.168.100.25.
#***********************************

import time
import sys
import telnetlib
HOST = "192.168.100.25"
PORT = "23"
with telnetlib.Telnet(HOST, PORT) as telnetObj:
    time.sleep(0.5)
    message = ("acc\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("OTTER\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("2ac\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAIL\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("OPEN X\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("Y\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)
    message = ("TAR R\r").encode('ascii')
    telnetObj.write(message)
    time.sleep(0.5)

print("Operating breaker X!")
```

A-4. MAIN CODE OF VOLTAGE SERVICE LIMITS SMART CONTRACT

```
package main

import (
        "encoding/json"
        "fmt"
        "log"
        "time"

        "github.com/hyperledger/fabric-contract-api-go/contractapi"
)

// DeviceRecord represents the structure for a device record
type DeviceRecord struct {
        DeviceID            string     `json:"deviceID"`
        MagXVoltagePhaseA   float64    `json:"magXVoltagePhaseA"`
        MagYVoltagePhaseA   float64    `json:"magYVoltagePhaseA"`
        MagXVoltagePhaseB   float64    `json:"magXVoltagePhaseB"`
        MagYVoltagePhaseB   float64    `json:"magYVoltagePhaseB"`
        MagXVoltagePhaseC   float64    `json:"magXVoltagePhaseC"`
        MagYVoltagePhaseC   float64    `json:"magYVoltagePhaseC"`
        MagXFreqHz          float64    `json:"magXFreqHz"`
        MagYFreqHz          float64    `json:"magYFreqHz"`
        BreakerXOpen        bool       `json:"breakerXOpen"`
        BreakerYOpen        bool       `json:"breakerYOpen"`
        Timestamp           time.Time  `json:"timestamp"`
}

// ConditionRecord represents the structure for a condition trigger record
type ConditionRecord struct {
        DeviceID         string     `json:"deviceID"`
        ConditionStatus  string     `json:"conditionStatus"`
        Timestamp        time.Time  `json:"timestamp"`
}

// EventData represents the structure for an emitted event data record
type EventData struct {
        DeviceID         string     `json:"deviceID"`
        ConditionStatus  string     `json:"conditionStatus"`
        BreakerOperation string     `json:"breakerOperation"`
        StartTimestamp   time.Time  `json:"startTimestamp"`
        EndTimestamp     time.Time  `json:"endTimestamp"`
}

// SmartContract provides functions for managing device records
type SmartContract struct {
        contractapi.Contract
}

// ConditionState tracks which voltage conditions are still valid
type ConditionState struct {
        NormalVoltage     bool
        YOverXWithin      bool
        YUnderXWithin     bool
        BothUnderVoltage  bool
        BothOverVoltage   bool
        XOverYWithin      bool
        XUnderYWithin     bool
        MaintenanceMode   bool
}
```

```
func queryDeviceHistoryByTimeRange(ctx contractapi.TransactionContextInterface, deviceID string, startTime string, endTime string) ([]*DeviceRecord, int, error) {
        queryString := fmt.Sprintf(`{
        "selector": {
            "deviceID": "%s",
            "timestamp": {
                "$gte": "%s",
                "$lte": "%s"
            }
        },
        "sort": [{"timestamp": "asc"}]
    }`, deviceID, startTime, endTime)

        log.Printf("Querying device '%s' history for time range: %s - %s", deviceID, startTime, endTime)

        resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)
        if err != nil {
                return nil, 0, err
        }
        defer resultsIterator.Close()

        var records []*DeviceRecord
        var count int

        for resultsIterator.HasNext() {
                queryResponse, err := resultsIterator.Next()
                if err != nil {
                        return nil, 0, err
                }

                var deviceRecord DeviceRecord
                err = json.Unmarshal(queryResponse.Value, &deviceRecord)
                if err != nil {
                        return nil, 0, err
                }

                records = append(records, &deviceRecord)
                count++
        }

        return records, count, nil
}
```

b

```
    func (s *SmartContract) AddOrUpdateRecord(ctx contractapi.TransactionContextInterface, deviceID string,
            magXFreqHz float64, magYFreqHz float64,
            magXVoltagePhaseA float64, magXVoltagePhaseB float64,
            magXVoltagePhaseC float64, magYVoltagePhaseA float64,
            magYVoltagePhaseB float64, magYVoltagePhaseC float64,
            breakerXOpen bool, breakerYOpen bool,
            timestamp string) error {

            ts, err := time.Parse(time.RFC3339, timestamp)
            if err != nil {
                    return fmt.Errorf("invalid timestamp: %s", err.Error())
            }

            log.Printf("* BEGIN AddOrUpdateRecord for deviceID %s at %s ...", deviceID, ts.Format(time.RFC3339))

            deviceRecord := DeviceRecord{
                    DeviceID:          deviceID,
                    MagXVoltagePhaseA: magXVoltagePhaseA,
                    MagYVoltagePhaseA: magYVoltagePhaseA,
                    MagXVoltagePhaseB: magXVoltagePhaseB,
                    MagYVoltagePhaseB: magYVoltagePhaseB,
                    MagXVoltagePhaseC: magXVoltagePhaseC,
                    MagYVoltagePhaseC: magYVoltagePhaseC,
                    MagXFreqHz:        magXFreqHz,
                    MagYFreqHz:        magYFreqHz,
                    BreakerXOpen:      breakerXOpen,
                    BreakerYOpen:      breakerYOpen,
                    Timestamp:         ts,
            }

            key, err := ctx.GetStub().CreateCompositeKey("DeviceRecord", []string{deviceID, ts.Format(time.RFC3339)})
            if err != nil {
                    log.Println("* END AddOrUpdateRecord.")
                    return fmt.Errorf("failed to create composite key: %s", err.Error())
            }

            deviceRecordAsBytes, err := json.Marshal(deviceRecord)
            if err != nil {
                    log.Println("* END AddOrUpdateRecord.")
                    return err
            }
```

c

```
        err = ctx.GetStub().PutState(key, deviceRecordAsBytes)
        if err != nil {
                log.Println("* END AddOrUpdateRecord.")
                return err
        }

        log.Printf("Added record: %+v", deviceRecord)

        err = s.CheckConditions(ctx, deviceID, timestamp)
        if err != nil {
                log.Printf("Automatic condition check failed for device %s at timestamp %s: %s", deviceID, timestamp, err.Error())
        }

        log.Println("* END AddOrUpdateRecord.")
        return nil
}

func (s *SmartContract) LogCondition(ctx contractapi.TransactionContextInterface, deviceID string, conditionStatus string) error {
        timestamp, err := ctx.GetStub().GetTxTimestamp()
        if err != nil {
                return fmt.Errorf("failed to get transaction timestamp: %v", err)
        }

        timestampTime := time.Unix(timestamp.Seconds, int64(timestamp.Nanos))

        conditionRecord := ConditionRecord{
                DeviceID:        deviceID,
                ConditionStatus: conditionStatus,
                Timestamp:       timestampTime,
        }

        conditionRecordAsBytes, err := json.Marshal(conditionRecord)
        if err != nil {
                return err
        }

        key, err := ctx.GetStub().CreateCompositeKey("ConditionRecord", []string{deviceID, conditionStatus, timestampTime.Format(time.RFC3339)})
        if err != nil {
                return err
        }

        return ctx.GetStub().PutState(key, conditionRecordAsBytes)
}

func initializeValidConditions(currentRecord *DeviceRecord) ConditionState {
        state := ConditionState{}

                if !currentRecord.BreakerYOpen && !currentRecord.BreakerXOpen {
                        // Case 1: Y closed, X closed
                        state.NormalVoltage = true
                        state.YOverXWithin = true
                        state.YUnderXWithin = true
                        state.BothUnderVoltage = true
                        state.BothOverVoltage = true
                } else if currentRecord.BreakerYOpen && !currentRecord.BreakerXOpen {
                        // Case 2: Y open, X closed
                        state.NormalVoltage = true
                        state.XOverYWithin = true
                        state.XUnderYWithin = true
                } else if !currentRecord.BreakerYOpen && currentRecord.BreakerXOpen {
                        // Case 3: Y closed, X open
                        state.NormalVoltage = true
                        state.YOverXWithin = true
                        state.YUnderXWithin = true
                } else {
                        // Case 4: Y open, X open
                        state.MaintenanceMode = true
                }

                return state
        }

        func checkVoltage(records []*DeviceRecord) (string, string, error) {
                const (
                        upperVoltageLimit = 7620.0 // v (7.62 kv)
                        lowerVoltageLimit = 6840.0 // v (6.84 kv)
                )

                if len(records) == 0 {
                        return "", "", fmt.Errorf("no records provided to checkVoltage function")
                }

                // get breaker state from most recent record
                currentRecord := records[len(records)-1]
                log.Printf("Using breaker states from most recent record @ %s: X=%v, Y=%v",
                        currentRecord.Timestamp.Format(time.RFC3339),
                        currentRecord.BreakerXOpen,
                        currentRecord.BreakerYOpen)

                // initialize valid conditions based on breaker state
                validConditions := initializeValidConditions(currentRecord)
```

```
// helper functions for voltage checks
isWithinLimits := func(magA, magB, magC float64) bool {
        return magA <= upperVoltageLimit && magA >= lowerVoltageLimit &&
                magB <= upperVoltageLimit && magB >= lowerVoltageLimit &&
                magC <= upperVoltageLimit && magC >= lowerVoltageLimit
}

isAboveMax := func(magA, magB, magC float64) bool {
        return magA > upperVoltageLimit || magB > upperVoltageLimit || magC > upperVoltageLimit
}

isBelowMin := func(magA, magB, magC float64) bool {
        return magA < lowerVoltageLimit || magB < lowerVoltageLimit || magC < lowerVoltageLimit
}

isNonNegative := func(magA, magB, magC float64) bool {
        return magA >= 0 && magB >= 0 && magC >= 0
}

for _, record := range records {
        // track which conditions this record satisfies
        recordConditions := ConditionState{}

        if !currentRecord.BreakerYOpen && !currentRecord.BreakerXOpen {
                // Case 1: Y closed, X closed
                if isWithinLimits(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isNonNegative(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.NormalVoltage = true
                }
                if isAboveMax(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isWithinLimits(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.YOverXWithin = true
                }
                if isBelowMin(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isWithinLimits(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.YUnderXWithin = true
                }
                if isBelowMin(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isBelowMin(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.BothUnderVoltage = true
                }
                if isAboveMax(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isAboveMax(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.BothOverVoltage = true
                }
        } else if currentRecord.BreakerYOpen && !currentRecord.BreakerXOpen {
                // Case 2: Y open, X closed
                if isWithinLimits(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) &&
                        isNonNegative(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) {
                        recordConditions.NormalVoltage = true
                }
                if isAboveMax(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) &&
                        isWithinLimits(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) {
                        recordConditions.XOverYWithin = true
                }
                if isBelowMin(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) &&
                        isWithinLimits(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) {
                        recordConditions.XUnderYWithin = true
                }
        } else if !currentRecord.BreakerYOpen && currentRecord.BreakerXOpen {
                // Case 3: Y closed, X open
                if isWithinLimits(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isNonNegative(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.NormalVoltage = true
                }
                if isAboveMax(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isWithinLimits(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.YOverXWithin = true
                }
                if isBelowMin(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isWithinLimits(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.YUnderXWithin = true
                }
        } else {
                // Case 4: Y open, X open
                if isNonNegative(record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC) &&
                        isNonNegative(record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC) {
                        recordConditions.MaintenanceMode = true
                }
        }
```

```
            // update valid conditions, only keep conditions that remain true
            validConditions.NormalVoltage = validConditions.NormalVoltage && recordConditions.NormalVoltage
            validConditions.YOverXWithin = validConditions.YOverXWithin && recordConditions.YOverXWithin
            validConditions.YUnderXWithin = validConditions.YUnderXWithin && recordConditions.YUnderXWithin
            validConditions.BothUnderVoltage = validConditions.BothUnderVoltage && recordConditions.BothUnderVoltage
            validConditions.BothOverVoltage = validConditions.BothOverVoltage && recordConditions.BothOverVoltage
            validConditions.XOverYWithin = validConditions.XOverYWithin && recordConditions.XOverYWithin
            validConditions.XUnderYWithin = validConditions.XUnderYWithin && recordConditions.XUnderYWithin
            validConditions.MaintenanceMode = validConditions.MaintenanceMode && recordConditions.MaintenanceMode

            // log.Printf("record @ %s - Y(A=%.1f V, B=%.1f V, C=%.1f V), X(A=%.1f V, B=%.1f V, C=%.1f V)",
            //      record.Timestamp.Format(time.RFC3339),
            //      record.MagYVoltagePhaseA, record.MagYVoltagePhaseB, record.MagYVoltagePhaseC,
            //      record.MagXVoltagePhaseA, record.MagXVoltagePhaseB, record.MagXVoltagePhaseC)
    }

    // check the final valid conditions to get result
    if !currentRecord.BreakerYOpen && !currentRecord.BreakerXOpen {
            // Case 1: Y closed, X closed
            if validConditions.NormalVoltage {
                    return "normal voltage", "none", nil
            }
            if validConditions.YOverXWithin || validConditions.BothOverVoltage {
                    return "over-voltage", "open_x", nil
            }
            if validConditions.YUnderXWithin || validConditions.BothUnderVoltage {
                    return "under-voltage", "open_x", nil
            }
            return "", "", fmt.Errorf("unexpected result in checkVoltage function: failed check for 'breaker Y is closed and breaker X is closed' situation")
    } else if currentRecord.BreakerYOpen && !currentRecord.BreakerXOpen {
            // Case 2: Y open, X closed
            if validConditions.NormalVoltage {
                    return "normal voltage", "none", nil
            }
            if validConditions.XOverYWithin {
                    return "over-voltage", "open_y", nil
            }
            if validConditions.XUnderYWithin {
                    return "under-voltage", "open_y", nil
            }
            return "", "", fmt.Errorf("unexpected result in checkVoltage function: failed check for 'breaker Y is open and breaker X is closed' situation")
    } else if !currentRecord.BreakerYOpen && currentRecord.BreakerXOpen {
```

h

```
            // Case 3: Y closed, X open
            if validConditions.NormalVoltage {
                    return "normal voltage", "none", nil
            }
            if validConditions.YOverXWithin {
                    return "over-voltage", "open_x", nil
            }
            if validConditions.YUnderXWithin {
                    return "under-voltage", "open_x", nil
            }
            return "", "", fmt.Errorf("unexpected result in checkVoltage function: failed check for 'breaker Y is closed and breaker X is open' situation")
    } else {
            // Case 4: Y open, X open
            if validConditions.MaintenanceMode {
                    return "maintenance operation or electrical fault", "none", nil
            }
            return "", "", fmt.Errorf("unexpected result in checkVoltage function: failed check for 'breaker Y is open and breaker X is open' situation")
    }
}

func checkTimeStateConditions(timestamp time.Time) bool {
    for i := 0; i <= 450; i++ {
            priorTimestamp := timestamp.Add(time.Duration(-i) * time.Second)

            // check if the current timestamp falls within the time period 10PM to 7AM UTC (excess noise site regulations)
            hour := priorTimestamp.UTC().Hour()
            if hour >= 22 || hour < 7 {
                    log.Printf("checkTimeStateConditions: evaluated true - hour = %d", hour)
                    return true
            }
    }

    return false
}

func triggerConditionEvent(ctx contractapi.TransactionContextInterface, deviceID string, conditionStatus string, breakerOperation string, startTimestamp, endTimestamp time.Time) error
    eventData := EventData{
            DeviceID:         deviceID,
            ConditionStatus:  conditionStatus,
            BreakerOperation: breakerOperation,
            StartTimestamp:   startTimestamp,
            EndTimestamp:     endTimestamp,
    }

    eventDataBytes, err := json.Marshal(eventData)
    if err != nil {
            return fmt.Errorf("failed to marshal event data: %v", err)
    }
```

i

```
        eventName := fmt.Sprintf("VoltageCondition_%s", conditionStatus)
        log.Printf("Triggering event '%s' with data: %s", eventName, eventDataBytes)
        err = ctx.GetStub().SetEvent(eventName, eventDataBytes)
        if err != nil {
                return fmt.Errorf("failed to set event: %v", err)
        }

        return nil
}

// CheckConditions checks the specified conditions for a given device
func (s *SmartContract) CheckConditions(ctx contractapi.TransactionContextInterface, deviceID string, timestamp string) error {
        currentTs, err := time.Parse(time.RFC3339, timestamp)
        if err != nil {
                return fmt.Errorf("invalid current timestamp: %s", err.Error())
        }
        log.Printf("* BEGIN checking conditions for device %s at %s ...", deviceID, currentTs.Format(time.RFC3339))

        if checkTimeStateConditions(currentTs) {
                log.Println("* Time state conditions met, skipping condition checks.")
                log.Println("* END checking conditions.")
                return nil
        }

        // query device records within the time window
        startTimestamp := currentTs.Add(-450 * time.Second)
        records, count, err := queryDeviceHistoryByTimeRange(ctx, deviceID,
                startTimestamp.Format(time.RFC3339), currentTs.Format(time.RFC3339))
        if err != nil {
                return fmt.Errorf("failed to query device history: %v", err)
        }
        log.Printf("Found %d records for device %s within the last 450 seconds", count, deviceID)

        // check if we have enough historical data to perform condition checks
        if count > 0 {
                oldestRecord := records[0] // records are sorted by timestamp ascending
                recordAge := currentTs.Sub(oldestRecord.Timestamp).Seconds()
                if recordAge < 400 {
                        log.Printf("Oldest record is only %.1f seconds old, need at least 400 seconds of history", recordAge)
                        log.Println("* END checking conditions.")
                        return nil
                }
        }


        conditionStatus, breakerOperation, err := checkVoltage(records)
        if err != nil {
                return fmt.Errorf("failed to check voltage: %v", err)
        }

        if conditionStatus != "normal voltage" {
                err = triggerConditionEvent(ctx, deviceID, conditionStatus, breakerOperation, startTimestamp, currentTs)
                if err != nil {
                        return err
                }
                if err := s.LogCondition(ctx, deviceID, conditionStatus); err != nil {
                        return err
                }
                log.Printf("* Condition check result: %s, breaker operation: %s", conditionStatus, breakerOperation)
                log.Println("* END checking conditions.")
                return nil
        }

        log.Println("* END checking conditions.")
        return nil
}

// QueryDeviceHistory queries the history of device records for a given time range
func (s *SmartContract) QueryDeviceHistory(ctx contractapi.TransactionContextInterface, deviceID string, startTime string, endTime string) ([]*DeviceRecord, error) {
        records, _, err := queryDeviceHistoryByTimeRange(ctx, deviceID, startTime, endTime)
        if err != nil {
                return nil, fmt.Errorf("failed to query device history: %v", err)
        }
        return records, nil
}

func main() {
        chaincode, err := contractapi.NewChaincode(&SmartContract{})
        if err != nil {
                fmt.Printf("Error create chaincode: %s", err.Error())
                return
        }

        if err := chaincode.Start(); err != nil {
                fmt.Printf("Error starting chaincode: %s", err.Error())
        }
}
```

A-5. EVENT LISTENER CODE OF VOLTAGE SERVICE LIMITS SMART CONTRACT

```
package main

import (
        "bytes"
        "context"
        "database/sql"
        "encoding/json"
        "flag"
        "fmt"
        "os/exec"
        "sync"
        "time"

        "github.com/hyperledger/fabric-gateway/pkg/client"
        _ "github.com/mattn/go-sqlite3"

        "app/pkg/connect"
)

const (
        channelName   = "mychannel"
        dbPath        = "./events.db"
        skipDuration = 450 * time.Second
)

type EventData struct {
        DeviceID          string     `json:"deviceID"`
        ConditionNumber   int        `json:"conditionNumber"`
        BreakerOperation  string     `json:"breakerOperation"`
        StartTimestamp    time.Time  `json:"startTimestamp"`
        EndTimestamp      time.Time  `json:"endTimestamp"`
        ActionTimestamp   time.Time
        EventTime         string
        EventName         string
        FormattedJSON     string
}

type EventHandler struct {
        mu sync.Mutex
        db *sql.DB
}

func (h *EventHandler) createEventsTable() error {
        h.mu.Lock()
        defer h.mu.Unlock()

    _, err := h.db.Exec(`
        CREATE TABLE IF NOT EXISTS events (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                device_id TEXT,
                condition_number INTEGER,
                breaker_operation TEXT,
                start_timestamp DATETIME,
                end_timestamp DATETIME,
                action_timestamp DATETIME
        )
    `)
    return err
}

func (h *EventHandler) saveEvent(eventData EventData) error {
    _, err := h.db.Exec(`
        INSERT INTO events (
            device_id, condition_number, breaker_operation,
            start_timestamp, end_timestamp, action_timestamp
        )
        VALUES (?, ?, ?, ?, ?, ?)
    `, eventData.DeviceID, eventData.ConditionNumber, eventData.BreakerOperation,
        eventData.StartTimestamp, eventData.EndTimestamp, eventData.ActionTimestamp)

    return err
}

func (h *EventHandler) getLastEvent(deviceID string, conditionNumber int) (*EventData, error) {
    var lastEvent EventData
    err := h.db.QueryRow(`
        SELECT device_id, condition_number, breaker_operation, start_timestamp, end_timestamp, action_timestamp
        FROM events
        WHERE device_id = ? AND condition_number = ?
        ORDER BY action_timestamp DESC
        LIMIT 1
    `, deviceID, conditionNumber).Scan(
        &lastEvent.DeviceID,
        &lastEvent.ConditionNumber,
        &lastEvent.BreakerOperation,
        &lastEvent.StartTimestamp,
        &lastEvent.EndTimestamp,
        &lastEvent.ActionTimestamp,
    )
```

```
        if err == sql.ErrNoRows {
                return nil, nil
        }
        if err != nil {
                return nil, err
        }
        return &lastEvent, nil
}

func (h *EventHandler) handleEvent(eventData EventData) {
        h.mu.Lock()
        defer h.mu.Unlock()

        lastEvent, err := h.getLastEvent(eventData.DeviceID, eventData.ConditionNumber)
        if err != nil {
                fmt.Printf("Error getting last event: %v\n", err)
                return
        }

        eventData.ActionTimestamp = time.Now()

        if lastEvent != nil {
                timeSinceLastEvent := eventData.ActionTimestamp.Sub(lastEvent.ActionTimestamp)
                if timeSinceLastEvent <= skipDuration {
                        fmt.Printf("Skipping event, less than %d seconds since last event for device %s (last event: %v, current event: %v)\n", int(skipDuration.Seconds()), eventData.
                        return
                }
        }

        fmt.Printf("<-- [%s] Chaincode event received: %s - %s\n", eventData.EventTime, eventData.EventName, eventData.FormattedJSON)

        if err := h.saveEvent(eventData); err != nil {
                fmt.Printf("Error saving event: %v\n", err)
                return
        }

        if eventData.BreakerOperation == "none" {
                fmt.Println("No action required for breaker_operation 'none'")
                return
        }

            scriptName := ""
            switch eventData.BreakerOperation {
            case "open_x", "close_x":
                    scriptName = "operate_x.py"
            case "open_y", "close_y":
                    scriptName = "operate_y.py"
            default:
                    fmt.Println("Invalid breaker_operation")
                    return
            }

            cmd := exec.Command("python3", scriptName)
            output, err := cmd.CombinedOutput()
            if err != nil {
                    fmt.Printf("Failed to execute script %s: %v, output: %s\n", scriptName, err, string(output))
                    return
            }

            currentTime := time.Now().Format(time.RFC3339)
            fmt.Printf("[%s] Output from script %s: %s\n", currentTime, scriptName, string(output))
}

func main() {
            var err error
            db, err := sql.Open("sqlite3", dbPath)
            if err != nil {
                    panic(fmt.Errorf("failed to open database: %w", err))
            }
            defer db.Close()

            eventHandler := &EventHandler{
                    mu: sync.Mutex{},
                    db: db,
            }

            err = eventHandler.createEventsTable()
            if err != nil {
                    panic(fmt.Errorf("failed to create events table: %w", err))
            }

            clientConnection := connect.NewGrpcConnection()
            defer clientConnection.Close()

            id := connect.NewIdentity()
            sign := connect.NewSign()
```

e

```
        var chaincodeName string
        flag.StringVar(&chaincodeName, "chaincodeName", "", "The name of the chaincode to monitor")
        flag.Parse()

        if chaincodeName == "" {
                fmt.Println("Chaincode name is required.")
                flag.PrintDefaults()
                return
        }

        fmt.Printf("Starting event listener for chaincode: %s\n", chaincodeName)

        gateway, err := client.Connect(
                id,
                client.WithSign(sign),
                client.WithClientConnection(clientConnection),
                client.WithEvaluateTimeout(5*time.Second),
                client.WithEndorseTimeout(15*time.Second),
                client.WithSubmitTimeout(5*time.Second),
                client.WithCommitStatusTimeout(1*time.Minute),
        )
        if err != nil {
                panic(err)
        }
        defer gateway.Close()

        network := gateway.GetNetwork(channelName)

        ctx, cancel := context.WithCancel(context.Background())
        defer cancel()

        startChaincodeEventListening(ctx, network, chaincodeName, eventHandler)

        select {}
}

func startChaincodeEventListening(ctx context.Context, network *client.Network, chaincodeName string, eventHandler *EventHandler) {
        fmt.Println("\n*** Start chaincode event listening")
```

f

```
        events, err := network.ChaincodeEvents(ctx, chaincodeName)
        if err != nil {
                panic(fmt.Errorf("failed to start chaincode event listening: %w", err))
        }

        go func() {
                for event := range events {
                        currentTime := time.Now().Format(time.RFC3339)
                        formattedJSON := formatJSON(event.Payload)

                        var eventData EventData
                        if err := json.Unmarshal(event.Payload, &eventData); err != nil {
                                fmt.Printf("Error parsing JSON payload: %v\n", err)
                                continue
                        }

                        eventData.EventTime = currentTime
                        eventData.EventName = event.EventName
                        eventData.FormattedJSON = formattedJSON

                        go eventHandler.handleEvent(eventData)
                }
        }()
}

func formatJSON(data []byte) string {
        var result bytes.Buffer
        if err := json.Indent(&result, data, "", "  "); err != nil {
                panic(fmt.Errorf("failed to parse JSON: %w", err))
        }
        return result.String()
}
```