Real-Time Byzantine Resilience for Power Grid Substations

Sahiti Bommareddy
Department of Computer Science
Johns Hopkins University
sahiti@cs.jhu.edu

Daniel Qian
Department of Computer Science
Johns Hopkins University
dqian3@jhu.edu

Christopher Bonebrake
Pacific Northwest National Labs
christopher.bonebrake@pnnl.gov

Paul Skare
Pacific Northwest National Labs
paul.skare@pnnl.gov

Yair Amir
Department of Computer Science
Johns Hopkins University
yairamir@cs.jhu.edu

***Abstract*—In the world of increasing cyber threats, a compromised protective relay can put power grid resilience at risk by irreparably damaging costly power assets or by causing significant disruptions. We present the first architecture and protocols for the substation that ensure correct protective relay operation in the face of successful relay intrusions and network attacks while meeting the required latency constraint of a quarter power cycle (4.167ms).**

Our architecture supports other rigid requirements, including continuous availability over a long system lifetime and seamless substation integration. We evaluate our implementation in a range of fault-free and faulty operation conditions, and provide deployment tradeoffs.

I. Introduction

The power grid is a complex critical infrastructure that generally connects thousands of power plants to millions of electricity customers. Electricity is generated at power generation plants and flows in the grid through several substations (at different voltage levels) before reaching the consumers. Some of the most critical protection functions are carried out by protective relays in the substations. For example, they protect the grid in case of incidents like short-circuits or overload currents. With the blurring lines between the grid Operational Technology (OT) and Information Technology (IT) networks, protective relays are attractive targets for malicious actors due to their critical role in grid resilience [1]. With hundreds to thousands of substations across a country to support the transmission and distribution of power, the attack surface is vast [2]. Hence, the growing landscape of cyber threats to protective relays in substations is a severe threat to overall grid resilience. This paper aims to strengthen grid resilience by addressing sophisticated cyber threats to protective relays and protection schemes in substations. Specifically, we develop Byzantine resilience for the substation that ensures correct protective relay operation even in the face of successful intrusions that compromise protective relays.

The role of the protective relay is to monitor its part of the grid continuously, and upon detecting that the substation state is not as desired (for example, due to a risky event), trip a breaker to disconnect the power circuit in order to protect power assets. A high voltage (345kV and up) transformer is an example of an asset protected by protective relays in transmission substations. Since such a transformer serves vast spans of the grid, costs millions of dollars, and takes over a year to procure, damaging it threatens grid stability. All protection schemes are time-critical, i.e., the reaction time needed is very exact to effectively protect the asset [3]. Due to the critical role of the protective relays and the very tight reaction time requirements, operators may deploy multiple relays for the same protection function, each with unilateral power to trip a breaker.

However, existing state-of-the-art protective relay technology is susceptible to intrusions. A protective relay with unilateral power under the control of an attacker presents two problems: First, a relay that does not trip when it should, can cause irreparable damage to the grid and its connected customers. Second, a relay that does unnecessarily trip causes a significant disruption to many customers. Either case is costly and highly undesirable.

Constructing a Byzantine resilient protective relay for the substation is challenging due to several rigid factors:

- The standard [3] requires relay reaction time to be within a quarter of a power cycle. Specifically, in a 60Hz system (e.g., in North America), a power cycle amounts to 16.67 milliseconds (ms), and a quarter cycle amounts to 4.167ms. This exact real-time constraint on the responsiveness of the relay is extremely demanding in the face of a successful attack.
- A Byzantine resilient protective relay function has to be continuously available in the field over a long lifetime (years). Therefore, proactive recovery is necessary to periodically, automatically and seamlessly regain control of compromised relays. This prevents the attacker from gaining long term access to relays and limits the attacker's ability to compromise a critical number of relays [4], [5].
- The protective relay is a relatively expensive device, and there are many of them across grid substations. Hence, to reduce cost there is a high incentive to limit the additional relays needed in a Byzantine resilient scheme.

- Any practical Byzantine resilient scheme has to seamlessly integrate into existing substations without significant changes to other substation components or overall substation architecture, in order to facilitate actual deployment.

The first intuition to address Byzantine faults at the substation level would be to tune and adapt classic state machine replication-based protocols (BFT SMR). However, in our view, it is incredibly challenging to meet all the substation requirements with state machine replication-based protocols as the environment is fundamentally different and operates at much finer time scales. For example, achieving the quarter-power cycle requirement in all operating conditions, including during successful intrusions and simultaneous network attacks, is highly challenging.

Our key insight is that the circuit breaker state in a relay-based protection function only depends on the most recent relay action (trip or close). This means that the total ordering provided by BFT SMR protocols is not strictly necessary to construct a Byzantine resilient system. Moreover, with BFT SMR, the minimum number of relays needed to tolerate f Byzantine relays with k relays undergoing proactive recovery simultaneously would be $3f + 2k + 1$ [4]. This means a BFT SMR-based solution needs at least six relays to practically tolerate a single intrusion. Since a protective relay is a fairly expensive device (tens of thousands of dollars), BFT SMR-based schemes will be costly in practice, reducing the feasibility of broad adoption.

We look at the environment in the grid substation and the specific need of a Byzantine resilient protective relay. We design an architecture and protocols to ensure protection functions work correctly even in the presence of successful intrusions and network attacks.

The primary contributions of this paper are:

- We propose the first real-time Byzantine resilient architecture and protocols for the substation that simultaneously address system compromises and network attacks while meeting the strict reaction time requirement.
- Our architecture can tolerate f Byzantine relays and k relays undergoing proactive recovery simultaneously with just $2f + k + 1$ total relays. This is in contrast to a traditional BFT SMR-based solution that would need $3f + 2k + 1$ total relays. This reduced cost can directly translate into deployment feasibility.
- Our architecture seamlessly integrates into existing substations without the need to modify existing components.
- We implement our architecture and two real-time protocols in Spire for the Substation, a system that extends the open-source Spire, the network-attack resilient intrusion-tolerant SCADA for the power grid [6].
- We deploy and evaluate our architecture and protocols in a substation testbed and show that the system meets the stringent quarter of a power cycle (4.167ms) latency requirement.

The rest of the paper is organized as follows: Section II presents our threat model and assumptions. Section III presents the Byzantine Resilient architecture for the substation. Section IV presents the Arbiter Protocol, a Byzantine protocol that operates within the architecture optimizing latency. Section V presents an alternative protocol, the Peer Protocol that reduces the attack surface and facilitates ease of integration while paying some latency costs. Section VI presents performance evaluation of both the Arbiter Protocol and the Peer Protocol in different operating conditions. Section VII describes related work and Section VIII concludes the paper.

II. Threat Model and Goals of a Solution

Our threat model is broad, requiring weak assumptions and covering both system-level compromises and network-level attacks on the substation network.

At the system level, we consider compromised (Byzantine) relays entirely under the attacker's control and that may exhibit arbitrary behavior. Byzantine relays can coordinate but are computationally bounded, and as such, cannot subvert the cryptographic mechanisms detailed in sections IV and V.

At the network level, we consider malicious network attacks, including low-level attacks such as arp-spoofing and other man-in-the-middle attacks, black hole attacks, and denial of service attacks on the substation local area network. The intrusion-tolerant networking foundation in Spire [6] addresses this broad network threat model. In conjunction with its network intrusion detection subsystem [7], it precludes long-running denial of service resource consumption attacks.

The requirement for real-time coordinated action within one-quarter of a power cycle necessitates some assumptions on clock synchronization between the substation protocol participants, which goes beyond the traditional asynchronous settings with eventual progress during stable times. Ideally, we would like to make the weakest assumption about clock synchronization that will still allow us to meet the required demanding timeliness. Since the requirement calls for the whole coordination process end-to-end, from sampled values to trip at the breaker, to take no more than 4.167ms, our protocols assume that the substation protocol participants will be synchronized to within one millisecond of each other.

In fact, substation standards demand precise time synchronization between all substation components. In practice, this is achieved by using the IEEE 1588 Precision Time Protocol (PTP) or IRIG-B, which provides accuracy in the sub-microsecond range [8]. Therefore, assuming a one-millisecond synchronization accuracy is at least three orders of magnitude coarser than necessary for substation functions to work, and hence is a very weak assumption.

We assume that all correct relays will issue a trip if the grid state requires a relay to trip. Of course, there can be still some differences between correct relays (e.g., from different manufacturers, and even different versions of the product from the same manufacturer) regarding how close to the boundary of necessity they trigger a trip but by that boundary (i.e., when it is necessary to protect grid assets) all correct relays will issue a trip.

We assume that at any time, at most f relays are compromised and k relays undergo proactive recovery simultaneously. As we support proactive recovery, to compromise the system, an attacker needs to compromise more than f relays in a single rejuvenation interval (i.e., time it takes until a relay node undergoes proactive recovery and completes the recovery process) [4]. Our protocols make the following guarantees:

1) A trip (or close) command is issued to the breaker only if there is at least one correct relay that decides to trip (or close) the breaker at that time.
2) If the grid needs to trip to protect power assets, a trip will be issued at that time.

Finally, we assume that the attacker has no physical access to system components and cannot manipulate Sampled Values (IEC61850-9-2, measurements of the power grid that act as inputs to the relay, [9]). Our threat model also does not cover a faulty circuit breaker. However, our architecture protects all system endpoints, including circuit breakers, making them considerably more difficult to compromise.

III. Byzantine Resilient Architecture

The protective relays continuously monitor the grid through electric current and voltage measurements. The measurements are multicasted by Merging Units over the process bus in Sampled Value messages. Protective relays process the measurements to detect situations that put the grid and its customers at risk (e.g., deviation in expected frequency). Relays, upon detecting such a situation, issue GOOSE messages (Generic Object Oriented Substation Event) to trip the circuit breaker, cut the power flow, and protect the grid assets and connected devices. Once the issue that caused the trip is resolved, an operator usually instructs the relay to issue a GOOSE message to close the circuit breaker.

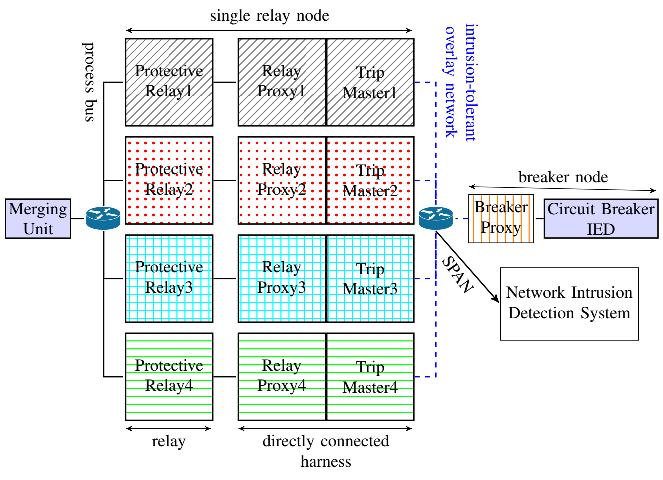

Fig. 1. An architecture that tolerates one Byzantine relay node ($f = 1$) and one relay node undergoing proactive recovery ($k = 1$) simultaneously with a total of four relay nodes in an IEC61850 Substation

A practical solution must be transparent to both the protective relays and the circuit breakers. Furthermore, such a solution must ensure that all other existing substation components are entirely ignorant of the new Byzantine resilience capabilities and continue to function unchanged. Therefore, we design our solution such that each protective relay works as if they were directly connected to the breaker and using the same (unprotected) protocols to issue their trip and close action commands. Similarly, the circuit breaker operates unchanged. To facilitate this mode of operation, our harness running Byzantine resilient protocols is directly attached to the protective relays on one side and the circuit breaker on the other side, as shown in Fig. 1. Our architecture in Fig. 1 shows that each relay and its directly connected harness (Relay Proxy and Trip Master) form a single logical entity called a relay node. Similarly, the circuit breaker and its directly attached harness form a single logical entity called a breaker node. The proxies limits the use of insecure communication protocols, namely the GOOSE messages in IEC61850, so that these messages are only used on a direct wire between the protective relay and its proxy and the circuit breaker and its proxy (i.e, they never flow on any of the networks).

All network communication in the system, between relay nodes and other relay nodes or the breaker node, is conducted over an intrusion-tolerant overlay network, implemented using the same networking component used in Spire [6]. This intrusion-tolerant network design addresses network-level threats in the substation's Local Area Network. This design is complemented by a machine-learning-based Network Intrusion Detection System to detect suspicious and anomalous traffic, providing situational awareness [7].

IV. Arbiter Protocol

To prevent f Byzantine relays from controlling the circuit breaker, we need the circuit breaker to act only if $f + 1$ relays issue the same action at about the same time. Thus, the total number of relays required to tolerate f Byzantine relays will minimally be $2f+1$. However, we need to support the diversity of attack surface and proactive recovery [4] . Therefore, to support k simultaneous relays going through proactive recovery, a total of $2f + k + 1$ relays are needed [10].

A simplistic solution could be to just have each relay node sign an action (i.e., trip or close) and send the action to the breaker node. Suppose the breaker node receives $f+1$ distinct verified relay messages with the same action and acts on it. In that case, an adversary that recorded these messages could replay them to change the breaker state in the future. Hence, we need to verify the freshness of relay messages. One way to ensure the freshness of the messages is to have the breaker node issue a nonce that is valid for a short time period. The relay nodes would then include this nonce along with the action in their signed message, and the breaker node would verify the freshness of the nonce when counting the $f + 1$ messages. In such a solution, the breaker node could issue nonces periodically or on-demand by relay nodes. Both of these approaches incur additional costs compared to simply generating a signed message with no nonce. While the former needs the breaker node to generate more messages periodically, the latter would add latency, requiring an additional round before generating a signed message. Either case would render the breaker node logic more complex. Furthermore, scaling would pose a challenge since these relay nodes can operate more than one circuit breaker. It would result in either additional messages or added latency, each of which is undesirable.

Fortunately, we can eliminate the need for an explicit nonce generation. In our Arbiter Protocol, we use the current time as part of the signed message sent by the relay node. The breaker node will verify freshness by ensuring the message is within a short period from its current clock time. The substation standards require time synchronization between substation components to be in the order of sub-microsecond. While we do not need such fine-grain synchronization, we still need to ensure the responsiveness to within the quarter cycle (4.167ms). Therefore, we chose our freshness period to be one millisecond, which is three orders of magnitude coarser than the substation requirements (hence adds a very weak assumption) but still allows our solution to meet the quarter cycle guarantee without the overhead associated with the explicit nonce techniques.

A. *Protocol Description*

Let the breaker be closed. Upon detection of a risky situation, a correct relay issues a trip message. The connected Trip Master (in the relay node) receives this trip action, generates a signed trip message with its local time, and sends it to the breaker node. The breaker node verifies signature and freshness (i.e., the time in the message is within one millisecond of its current time). Upon receipt of $f+1$ distinct fresh relay node trip messages, the breaker node trips the circuit breaker. After the circuit breaker trips, the breaker node multicasts the trip acknowledgment to the relay nodes.

Relay nodes continue to generate fresh signed trip messages (with updated time) every one millisecond as long as their relay remains in a trip state and they have not received the trip acknowledgment message.

When the risky situation is resolved, a correct relay issues a close message. The corresponding relay node generates a signed close message with its local time and sends it to the breaker node. Similar to the tripping logic above, upon receiving $f+1$ distinct fresh relay node close messages, the breaker node closes the circuit breaker. After the circuit breaker closes, the breaker node multicasts the close acknowledgment to the relay nodes.

Relay nodes continue to generate fresh signed close messages (with updated time) every one millisecond as long as their relay remains in a close state and they have not received the close acknowledgment message.

As described above, the breaker node in the Arbiter Protocol is complex with sophisticated logic. This increases attack surface and presents integration challenges as the system scales. In Section V we present the Peer Protocol addressing these important concerns.

V. Peer Protocol

In order to facilitate seamless integration while reducing the attack surface of the breaker node (which is a single point of failure), the breaker node should not have a role beyond authenticating the action message and executing the action. This can be achieved by moving the logic of the breaker node in the Arbiter Protocol into a distributed Byzantine resilient protocol in the Trip Master. This protocol coordinates the relay nodes and uses threshold signatures to ensure the support of $f+1$ relay nodes. With this support, relay nodes can generate a single threshold-signed action message to the breaker node. We present the details of this Peer Protocol below.

A. *Architecture Abstraction through Threshold Cryptography and Time Discretization*

The relay nodes make use of a threshold signature scheme. At a high level, an $(f+1, n)$ threshold signature scheme creates a public key and a signing key, just like a typical signature scheme. However, n parties split the signing key, and $f+1$ of these parties need to collaborate to create a signature. Most threshold signature schemes (including the one used in this protocol) use a two-step process to create these signatures. First, each party can create partial signatures on the same message (generating shares). Second, if any distinct set of at least $f+1$ of these valid shares exists on the same message, we can form the threshold signature by combining them. The main challenge with this process is that shares should be generated on the exact same message. One way to do this would be to have relay nodes agree on a message. However, this additional coordination would require at least an extra round of communication, which is prohibitively expensive given the stringent latency requirement.

To reduce latency, our solution is to have the relays independently generate shares on the same message without coordination. This means that only one round of communication is needed to collect the shares and generate a threshold-signed message for the breaker node. However, this message still needs to carry a nonce (like time in the Arbiter Protocol above) that can prove freshness. The raw local time can not be used as a nonce to create the same message on different relay nodes as the local times of the relay nodes would vary and messages would not be identical. Instead, we can discretize time to closest millisecond and use the discretized time as the nonce, henceforth called the Discretized Time Stamp (DTS).

Given our assumption that the nodes are synchronized to within 1ms of each other, two messages created at the same time on different nodes will always have the same or consecutive DTS. To see why these can be consecutive, we define a *DTS interval* as the set of times that discretize to the same DTS. The difference in time synchronization can cause one node's local time to be in one DTS interval, and the other node's local time to be in the next DTS interval. In fact, any arbitrarily small synchronization difference can cause this, if the local times are close enough to the endpoints of the intervals.

Now, let us describe how the DTSs are used in the protocol. If a relay node decides on an action, it will generate a share with the action and the current DTS. Relay nodes exchange these shares, and if there are $f + 1$ distinct valid shares on the same message (i.e., the same action and same DTS), relay nodes will generate a threshold signature message and send it to the circuit breaker node. Upon receiving the threshold signature message, the circuit breaker then verifies the signature and checks freshness by comparing the DTS on the message to its local DTS.

However, when the DTS of two different relay nodes are different, the nodes would not be able to combine their shares into a threshold signature. To address this, the relay node generates two shares: one for the current DTS and one for the next DTS. The relay node then sends both of these shares in the same message. Therefore, if two relay nodes simultaneously decide on the same action, there will be at least one share with the same DTS. Therefore, only one round of communication between the relay nodes is needed to threshold sign the action.

B. State Machine

The above description of the use of Threshold Cryptography and Discretized Time Stamps covers how relay nodes act when they need to generate a threshold-signed message (i.e., a trip or a close action). To completely specify the behavior of the Peer Protocol, we construct a state machine that handles the messages from the local relay, the other relay nodes, and the breaker node. This forms a single logical real-time Byzantine-resilient relay node out of multiple intrusion-prone real-time relay nodes.

For the ease of understanding, we describe the state machine in two steps: a partial state machine in Fig. 2 and a complete state machine in Fig. 3. The partial state machine presents the coordination algorithm during fault-free operation while assuming all correct relays issue an action at the same time. The complete state machine adds support for non-identical relays, benign and Byzantine faults, and proactive recovery.

The system state for each relay node is defined based on the relay state (denoted by r) and the breaker state (denoted by b). When the breaker node starts, it acquires the circuit breaker state and notifies the relay nodes of that state. The relay node updates its local relay state when the local relay (the one directly connected to that node) issues an action. In every relay node, the relay state and the breaker state each carry two essential pieces of information: the state (trip or close) and the DTS of the event occurrence (denoted by $r.t$ and $b.t$). For example, in Fig. 2, the *Closed* state in the top left corner has a relay state close (i.e., the local relay issued a close action), and the breaker state is close. Another example in Fig. 2, in *Attempt Trip* state, the local relay issued a trip and the breaker is closed. Note, when a relay node starts or recovers after proactive recovery, it acquires b and r, then uses their state and DTS data to transition into appropriate state in the state machine and resumes operation. However, to keep the state machine visualization simple, in Fig. 2 and Fig. 3 we do not show recovery and transitions to appropriate state.

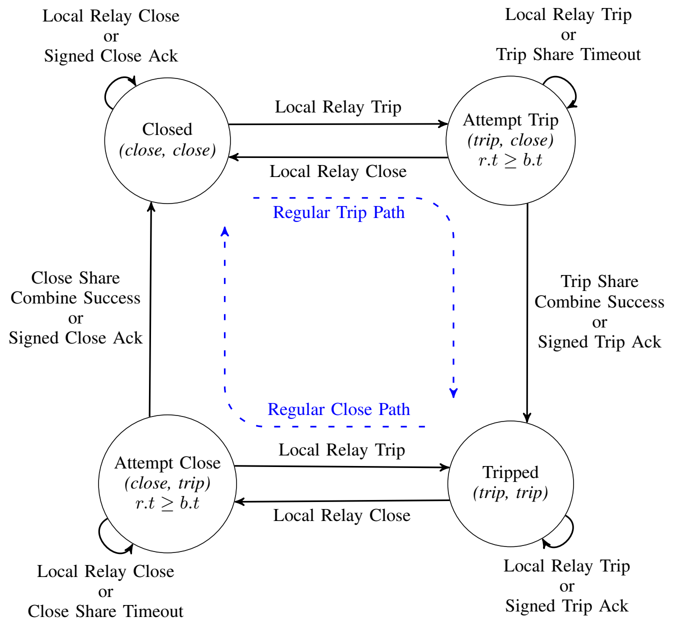

Fig. 2. A Partial State Machine

Let us assume that the relay node is in the *Closed* state (top left corner in Fig. 2). If a correct local relay detects a risky event, it will issue a local relay trip message ($r.t \geq b.t$). The relay node attempts to trip the breaker by generating and sending a trip share message to the other relay nodes with current and next DTS shares. It then transitions to the *Attempt Trip* state. While in that state, it receives shares from other relay nodes and keeps generating trip shares every DTS interval. Upon successfully combining shares and generating a valid threshold-signed trip message, the relay node sends that message to the breaker node and transitions to the *Tripped* state.

In the *Tripped* state, the relay node periodically sends the threshold-signed trip message to the breaker node until it receives a valid trip acknowledgment signed by the breaker node. A relay node will remain in the *Attempt Trip* state until either it can generate a valid threshold-signed trip message (by combining other relays shares) or until it receives a valid trip acknowledgment message from the circuit breaker node. The Regular Trip Path in Fig. 2 marks this path where relay nodes coordinate to trip a closed breaker.

A similar flow occurs when closing the breaker after the risky situation is resolved. A relay node in the *Tripped* state receives a close message ($r.t \geq b.t$) from its local relay. The relay node then attempts to close the breaker by generating and sending a close share message to the other relay nodes, with current and next DTS shares. It then transitions to *Attempt Close* state. While in that state, it receives shares from other relay nodes and keeps generating close shares every DTS interval. Upon successfully combining shares, generating a valid threshold-signed close message, the relay node sends

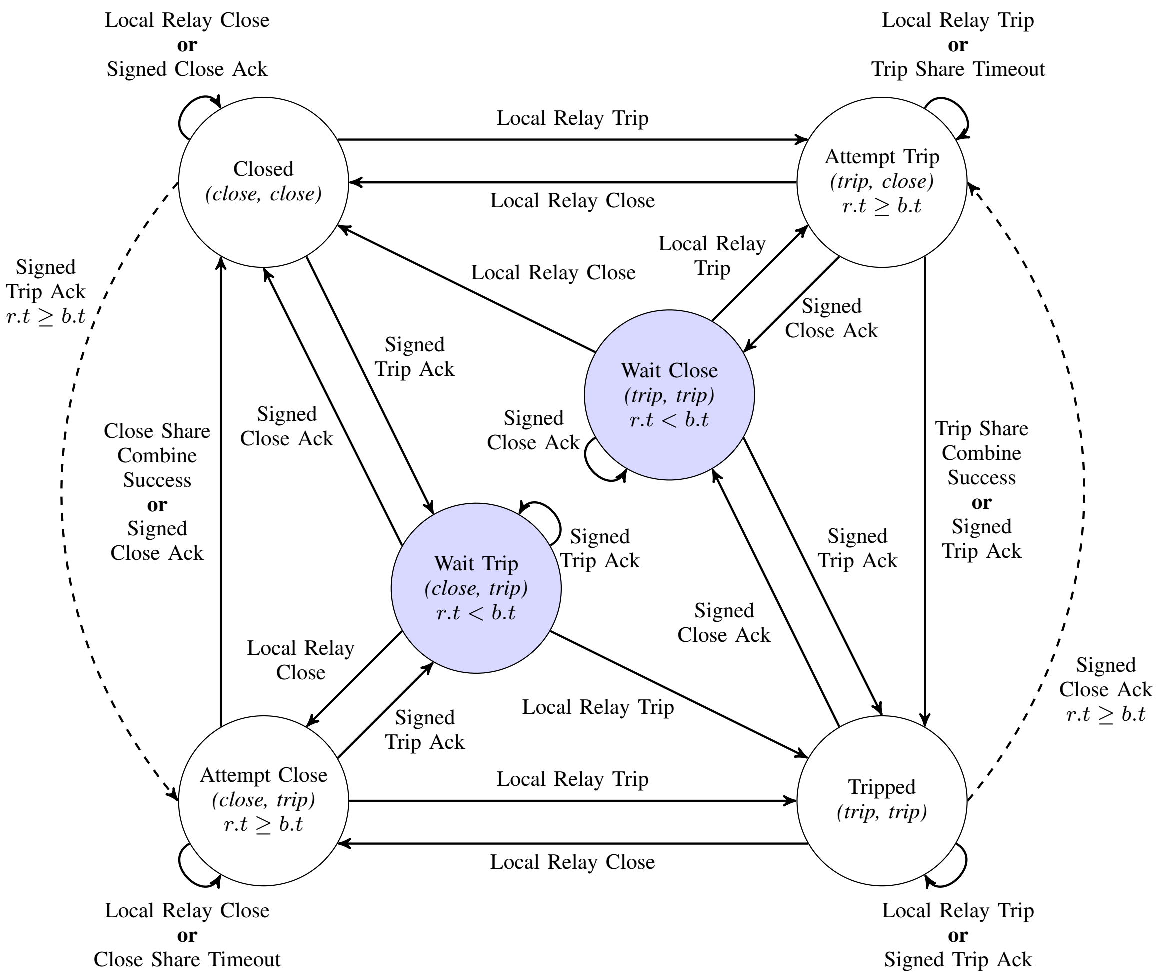

Fig. 3. A Full State Machine

that message to the breaker node and transitions to the *Closed* state.

In the *Closed* state, the relay node periodically sends the threshold-signed close message to the breaker node until it receives a valid close acknowledgment signed by the breaker node. A relay node will remain in the *Attempt Close* state until either it can generate a valid threshold-signed close message (by combining other relays shares) or until it receives a valid close acknowledgment message from the breaker node. The Regular Close Path in Fig. 2 marks this path where relay nodes coordinate to close a tripped breaker.

Unfortunately, relays may not be identical, i.e., correct relays may not trip or close simultaneously (at the exact same time) due to relay diversity (different manufacturers or versions). The partial state machine (Fig. 2) does not support a correct relay node that might trip or close relatively slower than other correct relay nodes. For example, let us assume all correct relay nodes are in *Closed* State and two relay nodes are fast and use the Regular Trip Path to transition from the *Closed* to the *Tripped* state. Another correct but slightly slower relay node remains in the *Closed* state (has not yet received a trip message from its local relay) but then receives a valid signed trip acknowledgement. It cannot follow the Regular Trip Path. Hence, to support non-identical relays, we add two wait states (*Wait Trip* and *Wait Close*), the shaded states in the complete state machine (Fig. 3). The relay nodes wait in these states for their local relay to catch up to their faster peers. In our example, the relay node can transition from the *Closed* state to the *Wait Trip* state. If the local relay is correct, it will eventually issue a local relay trip message and the relay node will transition from the *Wait Trip* state to the *Tripped* state. Similarly, a correct but slower relay node in the *Tripped* state can move to the *Wait Close* state on receiving a valid close acknowledgment before receiving its local relay close message. When its local relay issues a close message, the relay node transitions from the *Wait Close* state to the *Closed* state.

Furthermore, as discussed in the threat model above, in addition to a relay being delayed, it may fail and be unresponsive. For example, consider the scenario where Relay Node 1 is in the *Closed* state, and its local relay stops issuing any messages. There is a risky situation in which two other relay nodes (say Relay Node 2 and Relay Node 3) decide to trip and transition along the Regular Trip Path. Relay Node 1 will receive a valid trip acknowledgment message from the breaker node and will transition to the *Wait Trip* state. When the risky event is resolved, the other tripped relay nodes close the breaker, transitioning along the Regular Close Path. Upon receiving a valid close acknowledgment message, Relay Node 1 transitions from the *Wait Trip* to the *Closed* state. Until the local relay node resumes operation, the relay node toggles between *Closed* and *Wait Trip* states or *Tripped* and *Wait Close* states. Thus, the state machine addresses benign local relay faults.

Finally, a relay node under the control of an intruder can behave arbitrarily, i.e., exhibit Byzantine faults. For example, it can trip unnecessarily or not trip when needed. The Byzantine relay node can generate its own shares but cannot generate the threshold-signed trip message by itself ($f + 1$ valid shares are needed to generate the threshold-signed message). Hence, it cannot unilaterally trip the closed breaker or close the tripped breaker. The formal guarantees are elaborated in the proof sketch subsection.

C. *Diversity and Proactive Recovery*

In addition to the exact real-time latency requirement, a solution has to support a long system lifetime with continuous availability. The Byzantine resilient protocol can guarantee correctness as long as the compromised relay nodes do not exceed the design threshold of f. To enable the system to continue to work correctly as long as no more than a certain fraction of the relay nodes are compromised, we adapt from existing works the techniques of diversity and proactive recovery [5], [11]. Proactive Recovery entails forcefully restarting the relay node from a clean state, diversifying its attack surface, and refreshing cryptographic keys, before resuming operation.

A diverse attack surface in our setup requires diverse relay nodes, meaning both the protective relays and the harness need to be diverse. To accomplish that, we diversify the harness using standard approaches such as automatic software diversity at either compilation or run time [12]–[14]. Other useful techniques in that space include operating system diversity [15] and (the more expensive) N-version programming [16], [17]. A good approach to achieve attack surface diversity for the protective relays may be to procure them from different manufacturers (e.g. GE, Siemens, Hitachi Energy). Alternatively, the aforementioned diversity techniques could be employed for protective relays from the same manufacturer.

To maintain availability in the presence of both f intrusions and k relay nodes undergoing proactive recovery simultaneously, a BFT SMR-based solution requires $3f + 2k + 1$ total nodes, out of which $2f + k + 1$ nodes are needed to make progress. In contrast, both Peer Protocol and our Arbiter Protocol support proactive recovery with only $2f + k + 1$ total nodes, out of which only $f + 1$ nodes are needed to make progress [4], [10].

D. *Proof Sketch*

Peer Protocol gives the following guarantees:

Guarantee 1: Assuming no more than f relay nodes are compromised simultaneously, when a trip is issued to the circuit breaker, at least one correct protective relay issued a trip at that time.

Proof: The trip is issued to the circuit breaker through a threshold-signed message. In our scheme, the threshold is $f + 1$, i.e., $f + 1$ valid relay node shares are needed to combine and generate the threshold-signed message. Since we assume at most f simultaneous Byzantine relay nodes, there is valid share from a correct relay node in the $f + 1$ shares that were combined to generate the threshold-signed trip message.

Guarantee 2: Assuming no more than f relay nodes are compromised and no more than k relay nodes are undergoing proactive recovery simultaneously, if the grid needs to trip to protect power assets, a trip will be issued at that time.

Proof: We assume that any correct relay node will trip when the grid requires tripping. In our scheme, the threshold is $f+1$, i.e., $f + 1$ valid relay node shares are needed to generate the threshold-signed trip message. The total number of relay nodes in the system is $2f + k + 1$. Hence, even with simultaneous f compromises and k nodes going through proactive recovery, there are always at least $f + 1$ (i.e. $(2f + k + 1) - (f + k)$) correct relay nodes that will issue trip shares, enabling the generation of a valid threshold-signed trip message. According to the protocol, a correct relay node sends the threshold-signed trip message to the breaker node upon its generation.

Similar guarantees apply to the close action as well.

VI. Evaluation

In this section, we present detailed performance benchmarks for system with the Arbiter Protocol (Section IV) and Peer Protocol (Section V). These benchmarks are conducted in a range of fault-free and faulty operating conditions and demonstrate the advantages and the tradeoffs of the protocols.

A. *Performance Benchmarks Setup*

We deploy and benchmark the Arbiter Protocol and Peer Protocol in a testbed with real substation settings, allowing us to evaluate each protocol's ability to support the quarter-power cycle (4.167ms) latency requirement. The testbed machines are Intel NUCs running CentOS 8, with 1Gbps network connections and PTP-based clock synchronization. We emulate relays with a process that sends GOOSE (action) messages according to the IEC61850 standard. This relay emulator runs on each relay node, alongside the Relay Proxy and Trip Master processes, and publishes GOOSE messages based on Sampled Values. To emulate these Sampled Values, another machine sends a standard IP multicast messages to all the relay nodes, which then triggers a trip or close GOOSE action on the respective relay emulators.

The system consists of four relay nodes and therefore can simultaneously tolerate up to one faulty relay node (fail-stop or Byzantine) and up to one additional relay node undergoing proactive recovery. The end-to-end latency is measured,

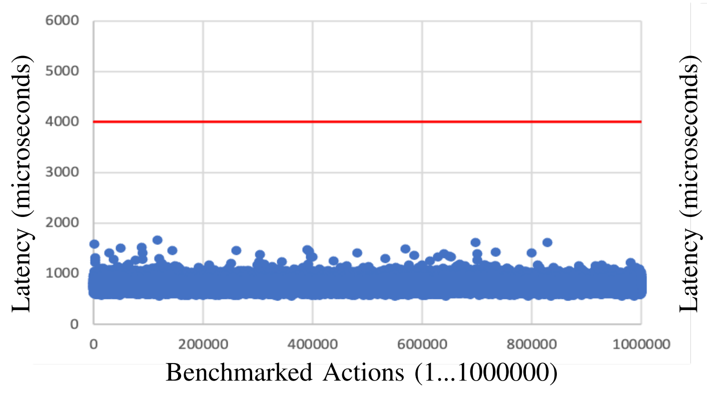

Fig. 4. Arbiter Protocol - Fault-Free

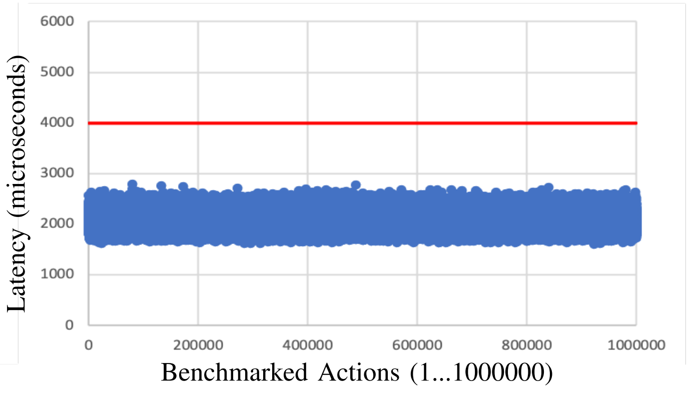

Fig. 5. Peer Protocol - Fault-Free

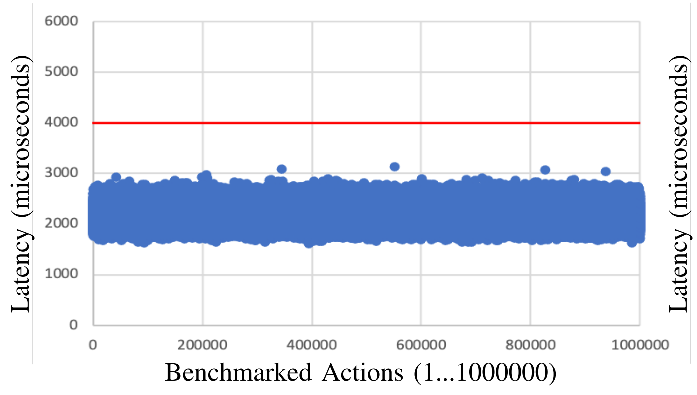

Fig. 6. Peer Protocol - Fail-Stop Fault

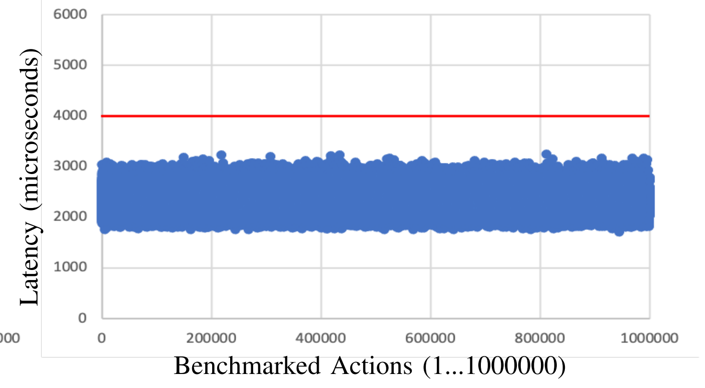

Fig. 7. Peer Protocol - Byzantine Fault

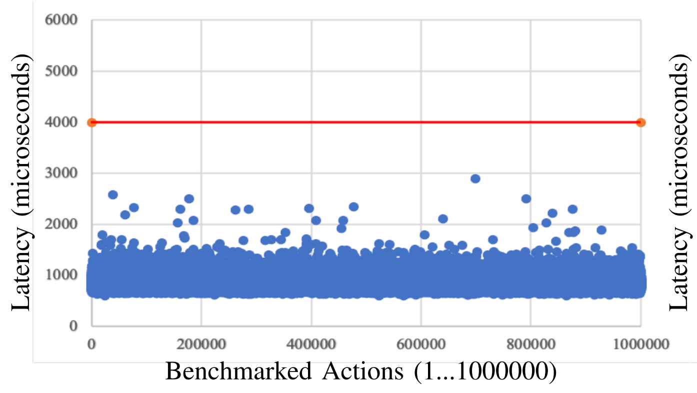

Fig. 8. Arbiter Protocol - Fail-Stop Fault and Proactive Recovery

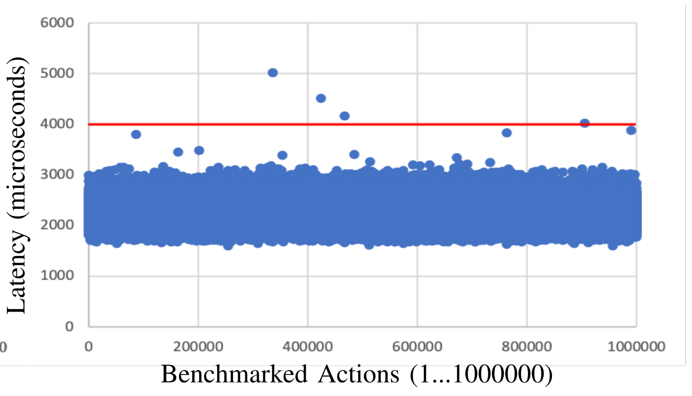

Fig. 9. Peer Protocol - Fail-Stop Fault and Proactive Recovery

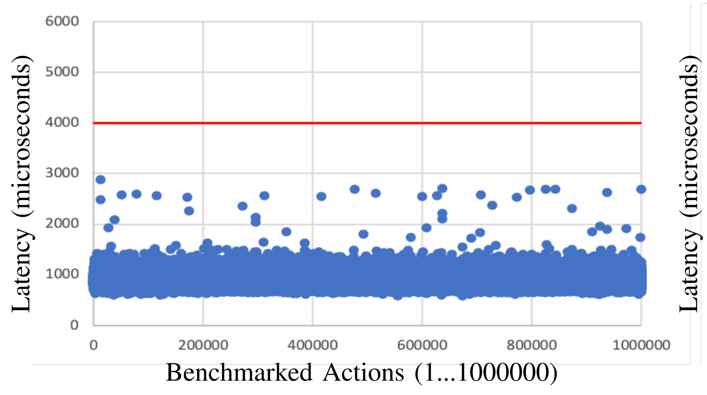

Fig. 10. Arbiter Protocol - Byzantine Fault and Proactive Recovery

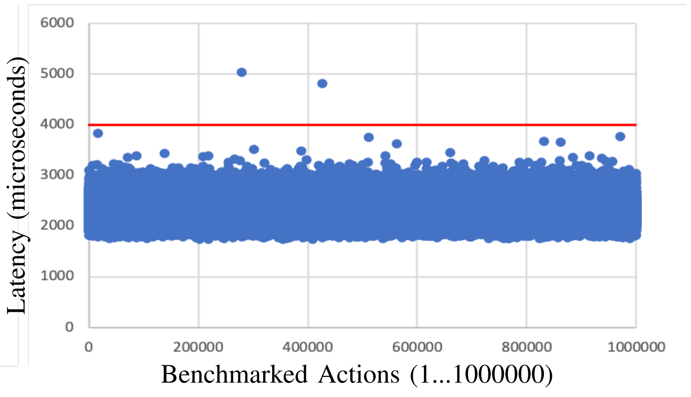

Fig. 11. Peer Protocol - Byzantine Fault and Proactive Recovery

TABLE I
PERFORMANCE OF THE ARBITER PROTOCOL AND PEER PROTOCOL UNDER DIFFERENT OPERATING CONDITIONS WITH FOUR RELAY NODES ($f = 1, k = 1$)

	Arbiter Protocol (microseconds)			**Peer Protocol (microseconds)**		
Operating Condition	**Minimum**	**Average**	**Maximum**	**Minimum**	**Average**	**Maximum**
Fault-Free (Normal)	**556**	**794**	**1665**	**1604**	**2048**	**2789**
Fail-Stop Fault or Proactive Recovery	**584**	**844**	**2104**	**1604**	**2132**	**3135**
Fail-Stop Fault and Proactive Recovery	**593**	**858**	**2887**	**1595**	**2167**	**5025** (4*)
Byzantine Fault	**564**	**887**	**2691**	**1716**	**2268**	**3253**
Byzantine Fault and Proactive Recovery	**577**	**904**	**2879**	**1733**	**2261**	**5028** (2*)

* The count of actions that crossed 4.167 milliseconds (out of 1 million total actions)

starting when a Sampled Value is multicast and ending when the circuit breaker changes its state. The Sampled Values are multicast from the same machine running the circuit breaker node so that the clock readings are done using the same local clock. Each benchmark includes one million end-to-end measurements of the trip and the close actions, conducted over a period of about 24 hours.

We evaluate the performance of the system in all five possible operational conditions, presented below:

Normal or Fault-free: all four relay nodes are working correctly.

Fail-Stop Fault: one of the relay nodes is unavailable due to a fail-stop fault. Note that this situation is identical to having one of the relay nodes undergoing proactive recovery.

Fail-Stop Fault with Proactive Recovery: one of the relay nodes is unavailable due to a fail-stop fault while simultaneously, an additional relay node is undergoing proactive recovery. Effectively, only two correct relay nodes are available.

Byzantine Fault: one of the relay nodes is under the control of a sophisticated attacker. We programmed such a node to perform two simultaneous attacks for each action. First, the Byzantine relay node sends a corrupt share that will result in an unsuccessful combination, costing the system precious time and computational resources of the relay nodes. Second, the Byzantine relay node performs a short intermittent denial of service attack on the other relay nodes to consume their network and computational resources further. Note that the Section III discusses why sustained resource consumption attacks are excluded.

Byzantine Fault with Proactive Recovery: one of the relay nodes exhibits the Byzantine Fault condition described above, and simultaneously, an additional relay node is undergoing proactive recovery. Effectively, only two correct relay nodes are available, while another compromised node attacks the system.

Note that in reality, our relay nodes complete proactive recovery and resume correct operation in a short period of time. However, we conduct the benchmarks involving proactive recovery without rejuvenating the relay node. In essence, these benchmarks are in the worst case scenario, where only two correct relay nodes are available, over all one million actions throughout the 24 hours. We refer to this situation as non-optionality.

Under non-optionality, meeting the latency requirement is particularly tough. With only two nodes, a random delay on either (e.g., from network delays, kernel scheduling, or even effects of a Byzantine node's actions) would be reflected in the end-to-end latency. Compare this situation to one in which three nodes are available. In such a case, delays would have to occur on two of the three nodes independently and at the same time in order to be reflected in the final latency. If only one node is delayed, the other two nodes would still be able to complete the action in a timely manner.

B. Performance Benchmarks

Table I reports the minimum, maximum and average of the latency distribution under each possible operating condition for both the protocols. For several of the notable benchmarks, we also present detailed scatter plots for the one million actions (Figs. 4 - 11). By comparing these plots, we can visualize the differences between the protocols and the effects of Byzantine faults and proactive recovery.

An immediate observation, as marked in the table, is both the Arbiter Protocol and the Peer Protocol meet the real time latency requirement during the Fault-Free operating condition (Fig. 4 and Fig. 5 respectively). When comparing the average latencies for the two protocols in the table, the Peer Protocol takes on average about 1300 microseconds longer. This is the price we pay for its advantages: the reduced attack surface for the circuit breaker and seamless integration with the substation.

In Fail-Stop Fault or Proactive Recovery operation conditions, the table shows that losing a single correct relay node shifts the distribution of latencies higher, with an average increase of about 50 microseconds for Arbiter Protocol and about 80 microseconds for Peer Protocol. However, both the protocols meet the real time latency requirement.

Another observation comes from comparing the Fail-Stop condition to the Byzantine Fault condition, as it quantifies the effect of an active intruder: the average latency increases by 50 microseconds for the Arbiter Protocol and 140 microseconds in case of the Peer Protocol, in addition to slight increase in variance. The impact of two factors—the cost of losing a single correct relay node and the effects of a Byzantine node—can be viewed as components that sum to the total difference in latency between the normal case and Byzantine fault (about 100 microsecond for the Arbiter Protocol and 220 microseconds for the Peer Protocol).

An interesting observation is the effect of non-optionality on both protocols. This effect is observed by comparing the Fail-Stop Fault condition to the Fail-Stop Fault with Proactive Recovery condition, and similarly, the Byzantine Fault condition to the Byzantine Fault with Proactive Recovery condition. While non-optionality does not have a significant impact on the average case as seen in Table I, it has a significant impact on the worst cases, as seen by the outliers (including those that do not cross the latency requirement) (Figs. 8, 9, 10, 11).

In the non-optionality conditions, the Arbiter Protocol meets the real time latency requirements in all cases (Figs. 8, 10), while the Peer Protocol meets it with a few exceptions: 4 actions out of one million in the Fail-Stop Fault with Proactive Recovery condition (Fig. 9) and 2 actions out of one million in the Byzantine Fault with Proactive Recovery condition (Fig. 11) violate the requirement. That the Fail-Stop Fault with Proactive Recovery condition has more exceptions is simply due to random chance, as the non-optionality is the critical factor for the worst case latencies, while the Byzantine behavior mainly impacts the average latency.

We emphasize that non-optionality only occurs for a short period of time in practice, as the node undergoing proactive recovery rejoins the protocol within a few seconds. The Peer Protocol, while not perfect, still provides better than five nines dependability (99.999%) even in these worst case conditions.

C. Protocol Analysis and Deployment Tradeoff

We consider the four design challenges: providing real-time operation, managing the economic cost, supporting continuous availability over a long system lifetime, and allowing seamless substation integration.

Real-time: The Arbiter Protocol requires a single one-way communication from relay nodes to the breaker node to achieve a coordinated action, providing Byzantine resilience with the least latency. The Peer Protocol adds additional latency due to exchanging messages for threshold cryptography. Therefore, the Arbiter Protocol meets strict latency requirements in all operating conditions while the Peer Protocol has few exceptions only in non-optionality condition.

Any deployable BFT SMR protocol would need at least one additional one-way exchange of messages during Fault-Free operation for ordering (if not a full round), compared with the Peer Protocol. We can estimate that this additional one-way exchange would make the average latency over 3ms, considering the one-way exchange in the Arbiter Protocol takes on average about 1ms. Furthermore, for any leader-based protocol, a malicious leader could trigger a leader election at a critical time. The system would then need to detect this failure and perform the leader change, further increasing the latency. Therefore, we can conclude that a BFT SMR-based protocol is unlikely to provide the latency requirement in the face of Byzantine faults, while using similar hardware, operating system, and network.

Economic Cost: Both the Arbiter Protocol and Peer Protocol need $2f + k + 1$ relay nodes to tolerate f Byzantine faults and k relays undergoing proactive recovery simultaneously. Compared to the $3f + 2k + 1$ relay nodes needed by any BFT SMR protocol for the same guarantee, this is a significant reduction that translates directly into cost savings. Given that relays are relatively expensive, this also makes our architecture much more viable for deployment.

Long System Lifetime: All approaches support diversity and proactive recovery. Proactive recovery ensures that we can reclaim relay nodes from an intruder after a successful intrusion.

Seamless Substation Integration: The Arbiter Protocol, while optimal in terms of latency, has a complex breaker node: any involved circuit breaker in the substation needs to know the architecture of the relay nodes, their identities, and threshold for action ($f + 1$ out of $2f + k + 1$). Additionally, this sophisticated logic in breaker nodes increases the attack surface across the substation, making such nodes more susceptible to intrusions and harder to harden. Any configuration changes in the Byzantine resilient solution, such as the level of resilience associated with f and k, or even a replacement of a relay node associated with a new id and keys, would affect the configuration of any involved breaker node. This makes a seamless deployment and scaling of the Arbiter Protocol much less feasible when compared to the Peer Protocol.

Deployment Tradeoff: The Arbiter and Peer Protocols provide a deployment tradeoff: While the Arbiter Protocol always meets the real-time requirement in all situations, it presents a larger attack surface as well as added integration complexity. The Peer Protocol significantly reduces the attack surface while providing seamless integration into the substation, for the cost of a slight risk of not meeting the latency requirement in the non-optionality edge cases. As both protocols are implemented within the intrusion-tolerant architecture, the choice of which protocol to use can be made at deployment.

VII. Related Work

Most of the literature related to intrusion-tolerant power grid infrastructure focuses on the control center SCADA (Supervisory Control and Data Acquisition) system and its connection to field devices (Remote Terminal Units). Kirsch et al. [18] made the first attempt to add intrusion-tolerance to SCADA. Eclipse SCADA [19] is an open-source SCADA that achieves intrusion tolerance using BFT-SMART [20] to replicate the SCADA Master. Another open-source SCADA system is Spire [6], which uses Prime [21] to replicate the SCADA Master. Spire also provides intrusion-tolerance at the network level by using an intrusion-tolerant overlay network [22]. The latency requirement at the control center level is on the order of 100-200 milliseconds. These works are complementary to ours as they protect the higher levels of the grid.

There is a wide range of BFT SMR protocols built to support different requirements. For example, protocols like

MinBFT [23] and Cheap BFT [24] reduce the number of replicas needed by making stronger assumptions. Protocols like RAM [25], EBAWA [26], Zyzzyva [27], and Aliph [28] focus on improving performance in the fault-free case. Protocols like Prime [21], Aardvark [29], and RBFT [30] are robust, i.e., achieve steady performance even under attacks. Protocols like HoneyBadger [31], SBFT [32], and Hotstuff [33] are built to scale. Nevertheless, they all report latency numbers that are at least between one and two orders of magnitude higher than our quarter of a cycle requirement in the worst case (including leader election).

More recent fault-tolerant SMR protocols such as HovercRaft [34], DARE [35], and Hermes [36] can achieve very low latencies (in tens of microseconds) in the fault-free case but can incur latencies in the tens of milliseconds during leader election. In addition, to achieve the excellent latency in the fault-free case, they require a special NIC (Network Interface Card), a lossless network, and Remote Direct Memory Access (RDMA) support [37]. These protocols are designed for applications deployed in Datacenter or HPC (High-Performance Computing Clusters), relying on high-speed networks (e.g., 100Gig/sec), making them unsuitable for the more conservative substation environment with its need for rugged hardware and traditional networks (e.g., 1 Gig/sec).

The works [4], [10] introduced the concept of tolerating f intrusions using $2f + k + 1$ replicas with proactive recovery, which we use in our work. However, there are fundamental differences between these works and ours: First, the Arbiter and Peer protocols are very different from the protocol in [4] and [10]. Specifically, [4] and [10] use symmetric cryptography with a trusted component in each replica, and use a leader. In contrast, our leaderless protocols use public key cryptography (the Arbiter Protocol) and threshold cryptography (the Peer Protocol) without a trusted component in each of the replicas. Second, the use of public key cryptography and threshold cryptography in our protocols, while advantageous from security perspective, presents a challenge to meet the real-time latency requirement that was never addressed in either [4] or [10]: they focus on average latency and throughput while our work focuses on worst-case latency, which, as shown in Table I, is very different.

Many cyber-attack incidents and red team experiments [7] have shown that approaches such as protecting the system with domain-specific firewalls [10], [38], Intrusion Detection Systems (IDS) [39], [40], and other industry best practices [41] are not sufficient. Therefore, we consider such techniques supplementary to intrusion-tolerant solutions, including our system.

VIII. Conclusion

We presented the first real-time Byzantine resilient architecture and protocols for the substation that simultaneously address system compromises and network attacks while meeting the strict timeliness requirement (4.167ms). Our architecture can tolerate f Byzantine relays and k relays undergoing proactive recovery simultaneously with just $2f + k + 1$ total relays. This is in contrast to a traditional BFT SMR-based solution that would need $3f + 2k + 1$ total relays. Our architecture seamlessly integrates into existing substations without the need to modify existing components.

Spire for the Substation is part of the open-source Spire, the network-attack resilient intrusion-tolerant SCADA for the power grid [6]. The system underwent a red team experiment that included protective relays from Siemens, GE, and Hitachi Energy in 2022.

Acknowledgement

This work was supported in part by the Department of Energy (DOE) Offices of Cybersecurity, Energy Security, and Emergency Response (CESER); Electricity (OE); and Nuclear Energy (NE) under the Grid Modernization Laboratory Consortium (GMLC) Topic 5.1.4 — Cyber-Physical Security. Its contents are solely the responsibility of the authors and do not represent the official view of DOE.

Yair Amir is a co-founder and member of and holds equity in Spread Concepts LLC. The results discussed in this paper could affect the value of Spread Concepts LLC. This arrangement has been reviewed and approved by the Johns Hopkins University in accordance with its conflict of interest policies.

References

[1] D. Weinberg, “The u.s. relies on transformers – and that’s a little scary,” Mar 2014. [Online]. Available: https://www.marketplace.org/2014/03/13/us-relies-transformers-and-thats-little-scary/

[2] “Electricity.” [Online]. Available: https://atlas.eia.gov/app/895faaf79d744f2ab3b72f8bd5778e68

[3] IEEE, “Ieee standard communication delivery time performance requirements for electric power substation automation,” *IEEE Std 1646-2004*, pp. 1–24, 2005.

[4] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo, “Highly available intrusion-tolerant services with proactive-reactive recovery,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 4, pp. 452–465, 2009.

[5] M. Castro and B. Liskov, “Practical byzantine fault tolerance and proactive recovery,” *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[6] A. Babay, T. Tantillo, T. Aron, M. Platania, and Y. Amir, “Network-attack-resilient intrusion-tolerant scada for the power grid,” in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 255–266.

[7] A. Babay, J. Schultz, T. Tantillo, S. Beckley, E. Jordan, K. Ruddell, K. Jordan, and Y. Amir, “Deploying intrusion-tolerant scada for the power grid,” in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 328–335.

[8] “Ieee 1588-2019 - ieee standard for a precision clock synchronization protocol for networked measurement and control systems.” [Online]. Available: https://standards.ieee.org/content/ieee-standards/en/standard/1588-2019.html

[9] “Iec.” [Online]. Available: https://webstore.iec.ch/publication/6007

[10] A. N. Bessani, P. Sousa, M. Correia, N. F. Neves, and P. Verissimo, “The crutial way of critical infrastructure protection,” *IEEE Security & Privacy*, vol. 6, no. 6, pp. 44–51, 2008.

[11] T. Roeder and F. B. Schneider, “Proactive obfuscation,” *ACM Transactions on Computer Systems (TOCS)*, vol. 28, no. 2, pp. 1–54, 2010.

[12] F. B. Cohen, “Operating system protection through program evolution.” *Comput. Secur.*, vol. 12, no. 6, pp. 565–584, 1993.

[13] S. Forrest, A. Somayaji, and D. H. Ackley, “Building diverse computer systems,” in *Proceedings. The Sixth Workshop on Hot Topics in Operating Systems (Cat. No. 97TB100133)*. IEEE, 1997, pp. 67–72.

[14] V. Pappas, M. Polychronakis, and A. D. Keromytis, "Smashing the gadgets: Hindering return-oriented programming using in-place code randomization," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 601–615.

[15] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Os diversity for intrusion tolerance: Myth or reality?" in *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2011, pp. 383–394.

[16] A. Avizienis, "The n-version approach to fault-tolerant software," *IEEE Transactions on software engineering*, no. 12, pp. 1491–1501, 1985.

[17] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Transactions on software engineering*, no. 1, pp. 96–109, 1986.

[18] J. Kirsch, S. Goose, Y. Amir, D. Wei, and P. Skare, "Survivable scada via intrusion-tolerant replication," *IEEE Transactions on Smart Grid*, vol. 1, no. 5, pp. 60–70, 2014.

[19] A. Nogueira, A. Bessani, and N. Neves, "Intrusion-tolerant eclipse scada," in *Symposium on Innovative Smart Grid Cybersecurity Solutions, Vienna, Austria*, 2017.

[20] A. Bessani, J. Sousa, and E. E. Alchieri, "State machine replication for the masses with bft-smart," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 355–362.

[21] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Prime: Byzantine replication under attack," *IEEE transactions on dependable and secure computing*, vol. 8, no. 4, pp. 564–577, 2010.

[22] D. Obenshain, T. Tantillo, A. Babay, J. Schultz, A. Newell, M. E. Hoque, Y. Amir, and C. Nita-Rotaru, "Practical intrusion-tolerant networks," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 45–56.

[23] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient byzantine fault-tolerance," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 16–30, 2011.

[24] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "Cheapbft: Resource-efficient byzantine fault tolerance," in *Proceedings of the 7th ACM european conference on Computer Systems*, 2012, pp. 295–308.

[25] Y. Mao, F. P. Junqueira, and K. Marzullo, "Towards low latency state machine replication for uncivil wide-area networks," in *Workshop on Hot Topics in System Dependability*. Citeseer, 2009.

[26] G. S. Veronese, M. Correia, A. N. Bessani, and L. C. Lung, "Ebawa: Efficient byzantine agreement for wide-area networks," in *2010 IEEE 12th International Symposium on High Assurance Systems Engineering*. IEEE, 2010, pp. 10–19.

[27] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, 2007, pp. 45–58.

[28] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 363–376.

[29] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making byzantine fault tolerant systems tolerate byzantine faults." in *NSDI*, vol. 9, 2009, pp. 153–168.

[30] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "Rbft: Redundant byzantine fault tolerance," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 297–306.

[31] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 31–42.

[32] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: a scalable and decentralized trust infrastructure," in *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2019, pp. 568–580.

[33] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[34] M. Kogias and E. Bugnion, "Hovercraft: achieving scalability and fault-tolerance for microsecond-scale datacenter services," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–17.

[35] M. Poke and T. Hoefler, "Dare: High-performance state machine replication on rdma networks," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, 2015, pp. 107–118.

[36] A. Katsarakis, V. Gavrielatos, M. S. Katebzadeh, A. Joshi, A. Dragojevic, B. Grot, and V. Nagarajan, "Hermes: A fast, fault-tolerant and linearizable replication protocol," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 201–217.

[37] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 202–215.

[38] M. Garcia, N. Neves, and A. Bessani, "Sieveq: A layered bft protection system for critical services," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 3, pp. 511–525, 2016.

[39] A. Bohara, U. Thakore, and W. H. Sanders, "Intrusion detection in enterprise systems by combining and clustering diverse monitor data," in *Proceedings of the Symposium and Bootcamp on the Science of Security*, 2016, pp. 7–16.

[40] S. Zonouz, J. Rrushi, and S. McLaughlin, "Detecting industrial control malware using automated plc code analytics," *IEEE Security & Privacy*, vol. 12, no. 6, pp. 40–47, 2014.

[41] V. M. Igure, S. A. Laughter, and R. D. Williams, "Security issues in scada networks," *computers & security*, vol. 25, no. 7, pp. 498–506, 2006.