On the Challenges of Building a BFT SCADA

André Nogueira, Miguel Garcia, Alysson Bessani, and Nuno Neves
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

Abstract—**In the last decade, Industrial Control Systems have been a frequent target of cyber attacks. As the current defenses sometimes fail to prevent more sophisticated threats, it is necessary to add advanced protection mechanisms to guarantee that correct operation is (always) maintained. In this work, we describe a Supervisory Control and Data Acquisition (SCADA) system enhanced with Byzantine fault-tolerant (BFT) techniques. We document the challenges of building such system from a "traditional" non-BFT solution. This effort resulted in a prototype that integrates the Eclipse NeoSCADA and the BFT-SMaRt open-source projects. We also present an evaluation comparing Eclipse NeoSCADA with our BFT solution. Although the results show a decrease in performance, our solution is still more than enough to accommodate realistic workloads.**

Keywords—**Byzantine fault tolerance, SCADA systems, State Machine Replication, Eclipse NeoSCADA.**

I. Introduction

Industrial Control Systems (ICS) security has relied on the past years on firewalls, Intrusion Detection Systems (IDS), and air-gapped architectures. Unfortunately, these mechanisms are not enough: Firewalls are prone to attacks [1] and they are typically a single-point-of-failure; IDS may fail to discover unknown attacks as their detection is based on attack signatures or anomaly detection [2]; In 2010, the Stuxnet attack [3] was designed to overpass the air-gapped defenses using an infected USB pen. In addition, the widespread integration of field and corporate networks make Supervisory Control and Data Acquisition (SCADA) systems more exposed to the plethora of attacks plaguing internet-based systems [4], which may lead to operational failures.

SCADA systems are the backbone of ICSs. For example, they monitor and manage the power grid and water supply systems machinery. Figure 1 shows a generic SCADA architecture with its main components: the Human-Machine Interface (HMI) is a computer that allows an operator to view the state of the infrastructure and react to events by issuing commands; the SCADA Master is a server that monitors and sends commands to Remote Terminal Units (RTU); the Frontends work as protocol translators between the RTUs and the SCADA Master; and the RTUs aggregate data from sensors located in the field, and execute commands in the actuators based on the SCADA Master instructions.

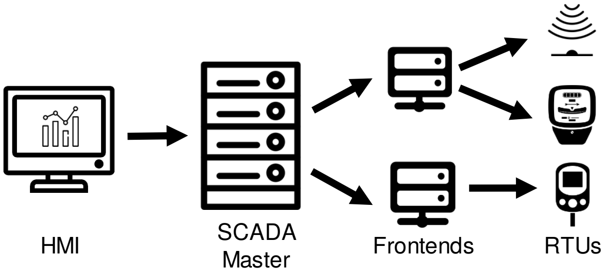


Fig. 1. SCADA generic architecture.

Modern SCADA systems normally employ fault tolerance techniques in the SCADA Master to ensure reliability, as it is the most critical component of the system. The SCADA Master is often deployed in a hot-standby configuration, where a primary server processes all collected data, mirroring its state changes to the backup server. In this type of configuration (i.e., passive replication) recovering from failures can be delayed while the backup server takes place as the primary. Moreover, this type of solution fails to protect the SCADA Master from malicious faults.

State Machine Replication (SMR) is an active replication approach that has been employed to ensure fault tolerance of fundamental services in modern internet-scale infrastructures (e.g., [5], [6], [7]). Given the critical role of the SCADA Master in the system, its design must consider correctness and availability. In particular, to build systems capable of operating correctly even in the presence of intrusions [8], it is necessary to employ Byzantine fault-tolerant (BFT) SMR, a particular case of SMR.

In this work, we describe the endeavor of making an intrusion-tolerant SCADA Master using BFT SMR. In this process, we have found some generic challenges that are relevant to other SCADA systems (e.g., [9], [10]). These challenges result from the need to guarantee the SMR properties, i.e., determinism and coordination, in a system that was built to be non-deterministic. One of the key decisions of our solution was to make it as modular as possible to avoid code modifications both on the SCADA and BFT library. In particular, our solution results from the integration of Eclipse NeoSCADA [11] with BFT-SMaRt [12], which are both stable open-source projects with several years of development. Although the idea of using BFT replication in a SCADA system was initially proposed in [9], no description was made about the challenges we are addressing in this work.

Our contributions can be summarized as follows:

- The identification of the main challenges to make intrusion tolerant a "traditional" SCADA Master;
- The design and implementation of a prototype, named SMaRt-SCADA, addressing these challenges;
- A preliminary performance evaluation to assess the overhead introduced by our solution.

The remainder of the paper is organized as follows: In §II,

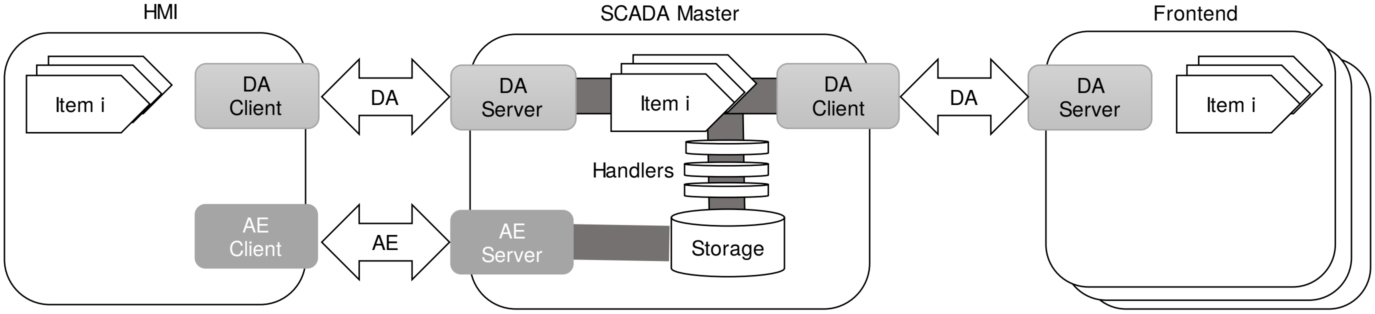


Fig. 2. Internal details of the main components of Eclipse NeoSCADA.

we introduce the non-replicated SCADA solution; In §III, we identify the challenges of making a BFT SCADA Master; In §IV, we present our prototype solution and how it addresses the identified challenges; In §V, we present the evaluation of the prototype; We present the related works in §VI; Finally, in §VII we present the conclusion and the lessons learned.

II. Eclipse NeoSCADA

Most of the existing SCADA solutions are commercial, and therefore, their code is not available for inspection and modification. Among the few open-source SCADA available, we decided to use the Eclipse NeoSCADA [11] in this work. It is a multi-platform "construction kit" for SCADA systems allowing system customizations. NeoSCADA is a project created in 2013, and its current version supports natively the following protocols (others can be added) Modbus TCP and RTU, Simatic S7 PLC, and JDBC. This system has been used in production to manage some industrial facilities [13].

A. Architecture

As can be observed in Figure 2, NeoSCADA has all the components of a typical SCADA system. NeoSCADA supports multiple Frontends, which are all connected to the SCADA Master. Although RTU devices are not part of NeoSCADA (and thus not shown in the figure), they are connected to the Frontends.

NeoSCADA employs two communication interfaces that specify the operations that can be performed. The *Data Access* (DA) interface defines a set of operations that permit to read or update values. The *Alarms and Events* (AE) interface allows the subscription of events that are generated when a controlled item's value reaches a certain threshold.

These interfaces allow the communication of the SCADA components: The Frontend contains items that represent the devices (e.g., *Item i* in Figure 2), such as sensors and actuators in the field, which are connected to the RTUs. Each item is composed of a name and a value. The SCADA Master also contains items, but these items are representations of the items in the Frontends. To receive and send data from/to the Frontends, the SCADA Master subscribes to the items in the Frontends using the DA communication interface. The HMI also contains items that are mapped inside the SCADA Master. Similar to the SCADA Master, the HMI also uses the DA communication interface to subscribe to the items.

In the SCADA Master, handlers can be added to the items to obtain enhanced functionalities. Handlers are associated with items to process their data values. NeoSCADA contains a set of default handlers: *Scale*, scales the value of an item; *Override*, overrides the current value of an item with a predefined value; *Monitor*, checks whether a value passes a certain threshold; and *Block*, blocks an operation while it waits for some condition to be verified. Every time a handler processes data, an event may be created and saved in the storage component. An event is created when the value is modified or when it reaches a predefined condition. To receive events, the HMI must subscribe to the SCADA Master's items of interest using the AE communication interface.

B. Operational use cases

Two main use cases define how the DA and AE interfaces are used, and the messages exchanged in the SCADA. Understanding these use cases is important as they will be modified in the BFT version:

a) Item update: This case encompasses the scenario where a Frontend is notified by a RTU about an item's value update, e.g., some RTU detected temperature changes. We consider that all the items available in the Frontend were previously subscribed by the HMI and SCADA Master. Upon the item's update, the Frontend informs the SCADA Master of the update, which after processing the information tells the HMI about the change. If the updated value raises an alarm in the SCADA Master, the HMI is also notified of that alarm.

Figure 3 shows the messages exchanged in the described scenario. Upon the item's update value, the Frontend creates an `ItemUpdate(ID,val)` message and sends it to the SCADA Master's DA client (DAC) via DA server (DAS) (1). In the SCADA Master, the update message goes through two different subsystems (2). One is the DA server, where the message is forwarded to all subscribers of that item. All DA clients that subscribed that item will receive the update (3). The other path is through the AE server (AES), where the handlers will process the item update. Each handler associated with that item will process the message and, if the values are modified or reached some condition, an event is created and

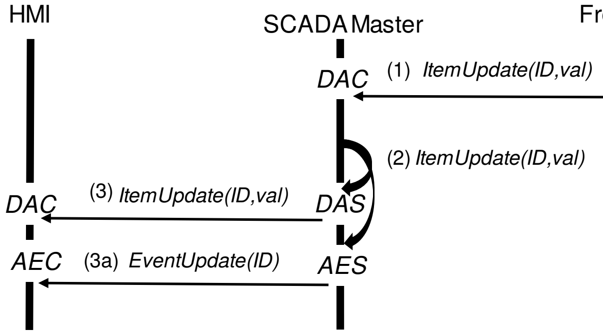


Fig. 3. Messages exchange in the Item update use case.

saved in the storage. All AE clients (AEC) that subscribed to that item will receive event notifications (3a).

b) Write value: This use case describes a scenario in which an operator, via HMI, requests a change for an item's value in a RTU. The request stays blocked in the SCADA Master waiting for a confirmation stating that the item could be modified. In the end, the HMI informs the operator if the item's value was successfully changed or not.

Figure 4 depicts the messages exchanged in this scenario. When the operator performs a write, the HMI sends a `WriteValue(ID, val)` from the DA client to the DA server, which places the write message in the DA subsystem (1) of the SCADA Master. The message is processed by its handlers (2), and then the SCADA Master sends the `WriteValue` message to the Frontend (3). After the corresponding RTU replies with a message, the Frontend returns a `WriteResult` message to the SCADA Master, indicating if the operation was successfully completed or not (4). In the SCADA Master, the message follows two different paths, similarly to what happens in the previous use case (5). In the DA subsystem, the `WriteResult` message is forwarded to DA client that performed the write operation (in this case, to the HMI) (6). In the AE subsystem, the handlers will process that write result. If some condition is activated, an event is created and saved in the internal storage and an `EventUpdate` message is sent to the HMI (6a).

In the case where there is an issue with a handler (2), a `WriteValue` message may not be sent to the Frontend (3). One of the default handlers of the SCADA Master is the Block handler. This handler blocks the message while it checks whether the operator can perform the write operation. If the operation is denied, the SCADA Master replies with two messages to the operator: The DA sends a `WriteResult` informing the operator that the requested operation was not performed, and an `EventUpdate` message is sent via AE. This message contains the reason why the SCADA Master was not able to execute the operation. The latter message is transmitted to the operator because the Block handler creates an event which contains recorded logging information and saves it in internal storage.

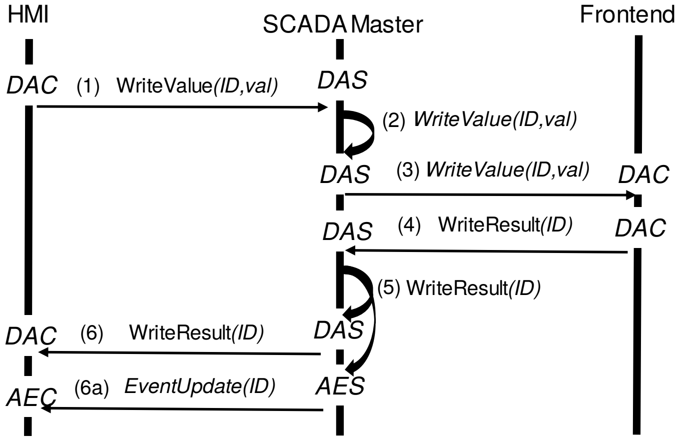


Fig. 4. Messages exchange in the Write value use case.

III. Building a BFT SCADA

In this section, we identify the main issues associated with the integration of a BFT library with the SCADA Master. Before we do that, we explain what is BFT replication in a nutshell.

A. BFT replication

BFT SMR is a well-known approach to replicate a service for Byzantine fault tolerance [14]. Typically, it is implemented as a request-reply protocol between clients and replicated servers. Each client issues requests to the servers, which process the same requests in the same order. Then, the servers reply to the client, which waits for a sufficient amount of matching replies. The key idea is to make replicas deterministically execute the same sequence of requests in such a way that, despite the failure of a fraction of the replicas, there is a quorum of correct nodes that have the same state and ensure the validity of the offered services.

B. Identified challenges

We have identified some generic challenges in building a BFT SCADA. They refer to the process of making a non-deterministic SCADA Master works as a state machine [15]. SMR requires that all replicas assure the following properties: *(1)* All SCADA Masters start from the same state; *(2)* All SCADA Masters execute the same sequence of messages; *(3)* All SCADA Masters execute the same state transitions. The following challenges identify the problems that we have found that violate these properties.

a) Multiple entry points: The SCADA Master contains multiple communication entry points. To communicate with the Frontends, the SCADA Master uses DA clients, while to communicate with the HMI, it uses a DA server and an AE server. Therefore, the SCADA Master can receive requests and replies from all these modules simultaneously. For example, the SCADA Master can receive at the same instant an `ItemUpdate` message from the Frontend and a `WriteValue` message from the HMI. In a single SCADA

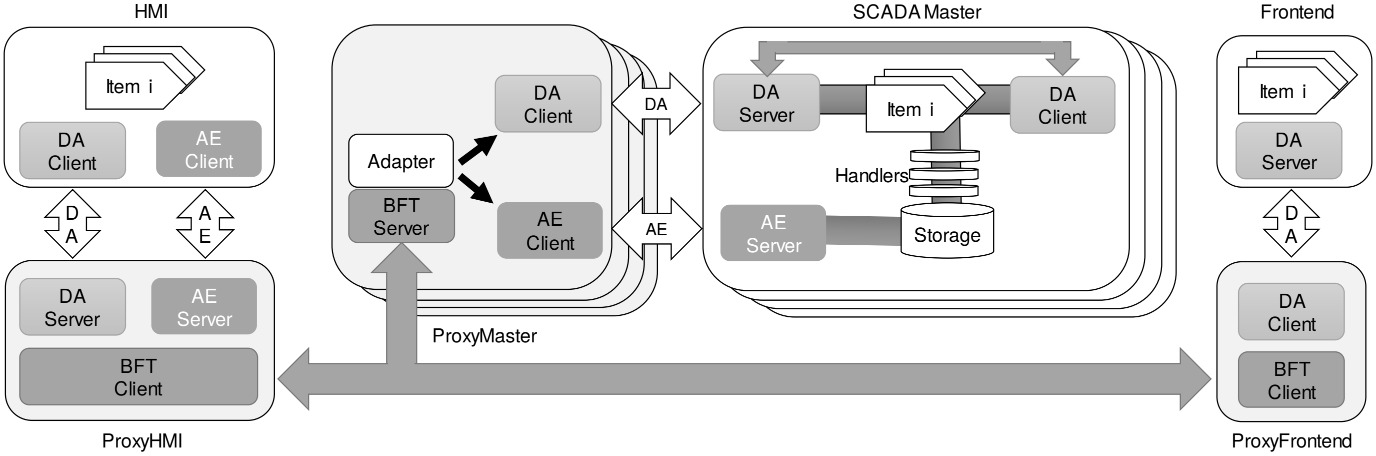


Fig. 5. The main components of SMaRt-SCADA.

Master deployment, these multiple entry points may enhance the server's performance, as there is no single bottleneck component to process all requests. However, in a replicated setting, the SCADA Master replicas need to process all these messages in the same order.

b) Multi-threading: Internally, the SCADA Master's DA and AE subsystems have several modules that execute concurrently. This enables the SCADA Master to receive and process multiple requests in parallel. In a context of SMR, multi-threading is a major difficulty because the execution of the requests is not deterministic. Even if all SCADA Master replicas receive the same requests, each replica could process them differently due to distinct scheduling decisions. This could cause the replicas to evolve to diverse internal states.

c) Non-deterministic timestamps: In the AE subsystem, some modules retrieve information from the operating system during their execution. For instance, when an event is created, a timestamp is retrieved from the operating system and assigned to the event. In a replicated solution, it is necessary to ensure that all replicas generate the same timestamp for the same event. Otherwise, replicas produce distinct events, compromising the required determinism.

d) Asynchronous messages: NeoSCADA was designed following a publish/subscribe architecture. The HMI must subscribe to an item in the SCADA Master to receive data or events associated with it. After that, the HMI starts receiving messages asynchronously. The SCADA Master can send multiple messages to the HMI in response to a processed message. For instance, after receiving an `ItemUpdate` message from the Frontend, the HMI can receive `ItemUpdate` and `EventUpdate` messages from the SCADA Master. In the replicated configuration, the HMI receives messages asynchronously from a set of replicas. Without additional information included in the messages, the HMI will not be able to know in which context these messages were produced.

IV. SMaRt-SCADA

In this section, we present SMaRt-SCADA, a BFT SCADA solution that addresses the identified challenges.

A. Architecture overview

Figure 5 depicts the SMaRt-SCADA architecture. The figure shows the main modifications that we made in NeoSCADA. The integration of NeoSCADA with BFT-SMaRt was simplified by using proxies that allowed us to minimize code modifications in the original system. Each original component has its own proxy to accommodate the BFT-SMaRt code: *1)* The ProxyMaster is responsible for forwarding all NeoSCADA messages that come from the Frontend and the HMI to the SCADA Master. Each ProxyMaster contains a BFT server, which is the server-side of the library where a BFT replication protocol runs. The BFT server communicates with the Adapter that is responsible for adding information to each incoming message and to decide to which client the message should be forwarded, DA or AE. *2)* The ProxyHMI receives the HMI messages and sends them via its BFT client, to the ProxyMaster. The BFT client is the client-side of the BFT library. In this proxy, we have a DA server and an AE server which simulate the servers available in the SCADA Master. *3)* The ProxyFrontend, which guarantees the communication between the Frontend and the SCADA Master. This proxy employs the BFT client of the library to transmit all messages that come from the Frontend to the SCADA Master. When the SCADA Master needs to communicate with the Frontend, the ProxyFrontend receives messages from the client-side of the library and forwards them using the DA client.

B. System model

SMaRt-SCADA assumes the system model used in [12], i.e., $n \geq 3f+1$, where n is the total number of replicas (each one a SCADA Master with the associated proxy), and f is the number of tolerated Byzantine replicas. As in other works, we assume that replicas fail independently due to some diversity mechanism [16]. We assume that both HMI and Frontends are correct as they need to cope with the protocol established to issue requests to the replicas. Moreover, the communication of HMI, SCADA Master, and Frontend with their proxies is protected by separate secure connections (currently TLS channels). Although this work is focused on the replication of the SCADA Master, we consider that the traditional defense mechanisms are present in the infrastructure, such as IDS and firewalls. SMaRt-SCADA complements their protection without interfering with them. Thus, it increases the effort that an adversary needs to spend to break the overall system.

C. Addressing the challenges

In the following, we show how SMaRt-SCADA addresses the challenges identified in § III-B. SMaRt-SCADA was developed by integrating the BFT-SMaRt state machine replication library with NeoSCADA. BFT-SMaRt guarantees that all SCADA Master nodes execute the same sequence of operations. It implements an agreement protocol that runs among the replicas. However, BFT-SMaRt assumes that all the replicas start from the same state and apply deterministic operations on the messages throughout the execution. Therefore, we needed to modify the NeoSCADA to meet these assumptions.

a) Multiple entry points: We introduced the proxy components and made a small number of internal modifications in the SCADA Master to convert the multiple entry points into a single one. This way, the SCADA Master does not receive messages simultaneously, and all messages that arrive are processed, one by one, following the order defined by the BFT library. The HMI and Frontends use the DA and AE channels as in the original version. Therefore they are not aware of the replication library in between. The SCADA Master is replicated in n instances and each one receives the requests from a ServerProxy instead of the HMI and Frontends. Additionally, we modified the SCADA Master DA Server and DA Client to guarantee that the messages coming from the Frontend DA server are placed correctly in the SCADA Master DA client (see Figure 5).

b) Multi-threading: The original SCADA Master processes messages in parallel. Without changes, it would compromise the determinism property as all SCADA Master replicas must apply the same modifications to their state. The best solution would be to use a replication library that supports multi-threaded applications (e.g., [17], [18], [19]). By resorting to these libraries, the necessary modifications in the source code of NeoSCADA would be minimal. Unfortunately, none of these implementations is available as open-source. Therefore, we had to refactor the SCADA Master to remove multi-threading to meet the BFT-SMaRt requirements. Then, as the execution becomes predictable, we can guarantee that all replicas execute deterministically.

c) Non-deterministic timestamps & Asynchronous messages: In the original version, the HMI and Frontend use the DA and AE client channels to communicate with the AE and DA servers in the SCADA Master. Since we have eliminated the multiple entry points in SMaRt-SCADA, we need to take the messages from the single entry point and deliver them to the correct channel server. We developed the Adapter to add a timestamp and ordering information to each incoming message, and then forward the messages to each SCADA Master subsystem and vice-versa. After receiving a message, the DA server in the SCADA Master passes the timestamp and the ordering data to the ContextInfo module. Modifying the DA and AE subsystems to retrieve this information from ContextInfo guarantees that all generated messages and events for a certain operation would have the same timestamp. In addition, the HMI can identify asynchronous messages as they contain information related to the ordering process.

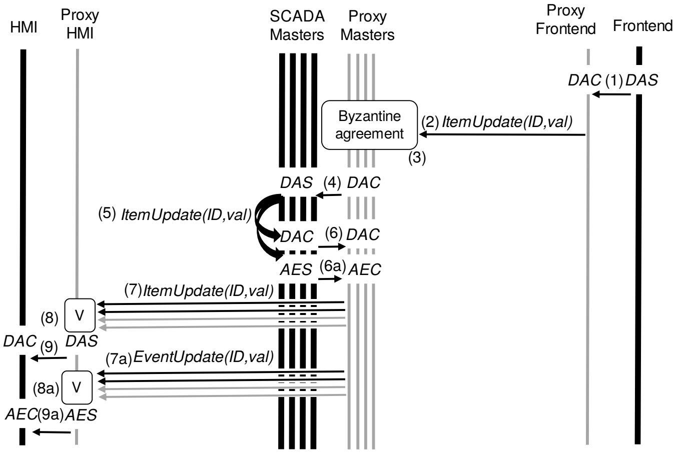


Fig. 6. The messages flow when an item update occurs in SMaRt-SCADA.

D. Operational use cases

We revisit the previous use cases, i.e., Item update and Write value, to show the communication changes in the replicated version of the SCADA Master.

a) Item update: Similar to the non-replicated Item update use case, we assume that all the items available in the Frontend were previously subscribed by the HMI and SCADA Master. In the following, we describe the scenario where a Frontend is notified by a RTU about an item value update. The Frontend uses its proxy to transmit the message to the ProxyMasters, which run a Byzantine agreement before delivering the message update to the SCADA Masters. Before the update reaches the HMI, the ProxyHMI waits for $f + 1$ matching messages from the replicas. As before, if the updated value raises an alarm in the SCADA Masters, the HMI is also notified of that alarm, in this case, it also waits for $f + 1$ matching messages.

Figure 6 shows the messages flow when an item's update occurs in the SMaRt-SCADA. When there is an item's update, the Frontend sends an `ItemUpdate` message to the ProxyFrontend (1). The ProxyFrontend sends the message to the ProxyMasters using the client-side of the BFT library (2). The ProxyMaster replicas run a Byzantine agreement to decide the message order (3) and deliver the message in the same order to the Adapters. Then, each replica Adapter places the message in the correct subsystem. In this case, such component is the DA client, which then sends the message to the DA server (4).

There, the DA server puts the message in the channel mapped to that item (5). The DA client connected to that channel receives the `ItemUpdate` and passes it to the DA and AE subsystems. Then, an `ItemUpdate` and an `EventUpdate` message are sent from the DA Client and AE Server, respectively, to the ProxyMaster (6 and 6a). The `ItemUpdate` and `EventUpdate` messages are transmitted to the ProxyHMI (7 and 7a), which also demultiplexes the messages and places them in the right communications interfaces. Then, the ProxyHMI waits for $f + 1$ equal messages from the ProxyMaster for both `ItemUpdate` and `EventUpdate` messages (8 and 8a). Next, the `ItemUpdate` and `EventUpdate` go to the DA server and to AE Server (9 and 9a), respectively. Finally, the HMI receives these messages.

b) Write value: Similar to the non-replicated write value use case, we describe the scenario where an operator, via HMI, requests a change to an item value in a RTU. In this scenario, the ProxyHMI and ProxyFrontend mediate the protocol by invoking the Byzantine agreement in the ProxyMasters. Then, they wait for $f+1$ matching messages from the ProxyMasters. After that, they can deliver the messages to both HMI and Frontend, respectively.

Figure 7 presents the message flow of sending a `WriteValue` from the HMI to the Frontend. The HMI uses the DA client to forward the `WriteValue` message to the DA server in the ProxyHMI (1). Then, it uses the client-side of the BFT library to replicate the message to every ProxyMaster (2). The ProxyMasters run a Byzantine agreement to decide the message order and deliver the message to each own Adapter (3). The Adapter uses the DA client to send the `WriteValue` message to the SCADA Master (4).

The DA server receives the message and places it into the DA subsystem. Before arriving at the DA client, the message passes through the handlers in the AE subsystem. Then, the DA client redirects the message to the ProxyMaster via the SCADA Master DA server and stays blocked (5). The Adapter receives the `WriteValue` message and forwards it to the Frontend via its ProxyFrontend (6), which waits for $f + 1$ matching messages (7). Finally, the `WriteValue` message is transmitted to the Frontend using the DA client (8).

In the other way around, the RTU replies to the Frontend, which sends a `WriteResult` message to the ProxyFrontend DA client (9). Then, it uses the replication library client-side to inform the ProxyMaster replicas (10). These replicas, after running a Byzantine agreement to decide the order of such messages (11), deliver the `WriteResult` to the Adapter that sends it to the DA client. The message is then given to the DA server that places it in the mapped channel to the item with that `WriteResult`. The DA client of that item receives the result and passes it to the DA and AE subsystems (12). A `WriteResult` and `EventUpdate` messages are forwarded to the DA and AE client in the ProxyMaster (13 and 13a). Recall that the latter is sent only if an alarm event is created. The Adapter waits until it receives the result message related to the `WriteResult` message. `WriteResult` and `EventUpdate` messages are sent to the ProxyHMI (14 and 14a), the ProxyHMI waits for $f + 1$ equal messages for `WriteResult` and `EventUpdate` (15 and 15a), respectively. There the `WriteResult` message goes to the DA server (16) and the `EventUpdate` goes to the AE Server (16a). Finally, the HMI receives these messages.

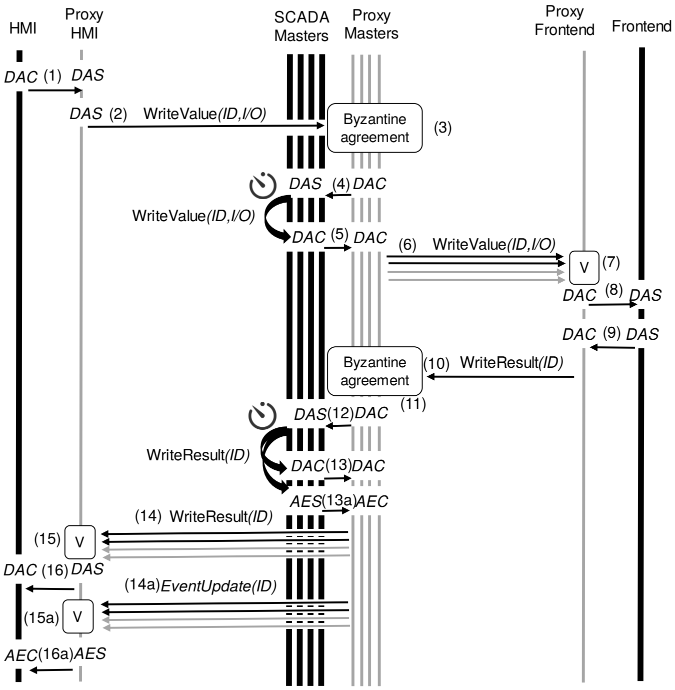


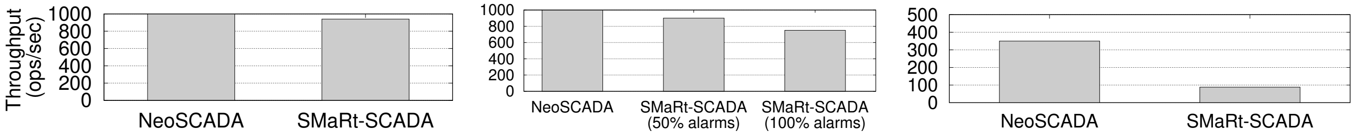
Fig. 7. The messages flow when a write value is executed in SMaRt-SCADA.

During the write operation, the SCADA Master liveness can be compromised. More precisely, when the SCADA Master sends a `WriteValue` message to the Frontend via the ProxyFrontend. The ProxyMaster's DA client stays blocked until it receives a `WriteResult` message. If such message never arrives, the SCADA Master will be blocked forever. To prevent this, we used an approach similar to the one that was prepared in [9]. The Adapter contains a timeout mechanism that is triggered when the `WriteValue` message is forwarded to the Frontend. Each Adapter sends to the other Adapters a timeout message informing that a timeout was exceeded due to a missing `WriteResult` message. When a majority of timeout messages arrive at each Adapter, an empty `WriteResult` message is sent to the SCADA Master informing that a timeout has occurred. This way, we can ensure the liveness of the SCADA Master even if an attacker drops `WriteValue` or `WriteResult` messages.

V. Experimental Evaluation

This section presents a preliminary experimental evaluation comparing the performance of SMaRt-SCADA with the original NeoSCADA.

The machines used in the experiments have two quad-core 2.27 GHz Intel Xeon E5520 with 32 GB of RAM memory and are interconnected by a Gigabit Ethernet switch. The machines



(a) Update value use case.

(b) Update value use case with the AE subsystem.

(c) Write value use case.

Fig. 8. Performance evaluation between NeoSCADA and SMaRt-SCADA.

run Linux Ubuntu 14.04 (64-bit) with kernel version 3.13.0-32-generic and Oracle Java 1.7.0 80-b15.

We deployed the NeoSCADA in three machines, each one containing its component: Frontend, SCADA Master, and HMI. In contrast, we deployed the SMaRt-SCADA in six machines: one Frontend, four SCADA Masters, and one HMI, each one also containing its corresponding proxy.

A. Update Item workload

We started our evaluation by deploying a scenario where the Frontend contains a set of items that are being updated, and then the Frontend sends the updates to the SCADA Master which then forwards them to the HMI. In this experiment, we considered 1000 `ItemUpdate` messages per second to simulate the same workload used to evaluate the intrusion-tolerant logical timeout protocol of Kirsch *et al.* [9]. This workload emulates a scenario wherein every second 1000 RTUs are updated and then propagate their information to the Frontend. We have simplified this experiment by removing the RTUs, as the Frontend generate the messages. We validated this workload with the staff of an electrical company that runs a country-scale SCADA, and they said that it is significantly above to what they typically observe, even in a crisis.

Figure 8(a) illustrates the number of messages processed by NeoSCADA and SMaRt-SCADA. There is a performance drop of 6% in the SMaRt-SCADA due to the additional steps needed to perform the updates (see Figures 3 and 6): In NeoSCADA, each `ItemUpdate` message takes 3 communication steps to go from the Frontend to the HMI, but in the SMaRt-SCADA the same operation takes 9 steps. In SMaRt-SCADA, each `ItemUpdate` message is executed in each SCADA Master after the ProxyMasters execute a Byzantine agreement and later is voted in the ProxyHMI.

We repeated this experiment to understand the overhead associated with the setup of alarms. More precisely, we added the Monitor handler in the SCADA Master to verify whether the items value passes a certain threshold. When that occurs, an `EventUpdate` message is produced, saved the internal storage, and finally sent to the HMI. In this experiment, we exercise both the DA and AE subsystems.

Figure 8(b) shows the number of messages processed. We ran both solutions in two different alarm scenarios. In one scenario, we considered that every `ItemUpdate` message triggers an alarm (100%-alarms), while in the other, half of them do it (50%-alarms). While NeoSCADA was able to process all messages for both percentages of alarms, SMaRt-SCADA presents an overhead of 10% and 25% for the 50%- and 100%-alarms scenarios, respectively. The throughput decrease reflects the additional communication steps introduced by our solution. In particular, in the 100%-alarms scenario the number of events that go to storage is twice what was observed in the 50%-alarms scenario.

B. Write Value workload

In this last experiment, we evaluated the performance of both solutions for the Write value use case. We considered that the HMI performs synchronous writes in a Frontend's item. This means that, for each write operation, the HMI waits until the operation is completed.

Figure 8(c) illustrates the number of writes that can be performed in both solutions. We can observe that the SMaRt-SCADA introduces an overhead of 78%. This significant throughput decrease results from the additional 10 communications steps that our solution needs to perform the write operation (see Figures 4 and 7). Additionally, since in our solution the SCADA Master is single threaded, it does not take full advantage of multi-core CPUs. Moreover, we observed that the BFT-SMaRt is not the bottleneck of our system, as it reaches a throughput of $16k$ requests/sec for a similar message size (1024k bytes) [12]. However, the throughput achieved by SMaRt-SCADA is sufficient to accommodate a real-world workload, as it is virtually impossible for a group of human operators to perform almost 100 commands/second.

VI. Related Work

There are only a few works dedicated to the effort of building dependable SCADA systems. Kirsch *et al.* made the first attempt to build BFT SCADA Master using state machine replication [9]. Although there was an integration of their BFT library with a real SCADA Master product, the authors provided few details about the issues raised during the integration of the system with a BFT library. However, they presented two detailed challenges and solutions. The first challenge concerns the type of communication of a traditional SCADA system, within a replicated environment. The authors proposed a logical timeout to synchronize the replicas polling the RTUs. The second challenge is related to the typical communication pattern of state machine replication, i.e., a client makes a request and waits for the response of the servers. In a SCADA scenario, the communications can be bi-directional and asynchronous, since it is event-driven. The authors developed a communication abstraction between the clients (e.g., HMI and Frontends) and the SCADA Masters. In our work, we used a similar logical timeout and the second challenge is solved by BFT-SMaRt that allows clients to send and receive asynchronous messages.

Spire [10] is the first intrusion-tolerant SCADA system designed to tolerate faults both in the system and the network. Moreover, Spire main goal was to tackle the worst case scenario, whereas highly determined attackers can overpass the existent intrusion-tolerant techniques. Their solution requires a significant extra number of nodes to support availability in the presence of simultaneous intrusions. The system was tested in a wide-area deployment with two control centers (i.e., in site) connected to and two data centers (i.e., external to the SCADA infrastructure facilities). Our work is complementary to Spire as it provides no discussion about the challenges of replicating a SCADA service.

VII. Lessons Learned & Conclusion

In this paper, we presented the challenges in building a BFT SCADA system by integrating two open-source projects. We designed a solution to address these challenges, implementing the SMaRt-SCADA prototype as a result. In this process, we had to make a few design and implementation decisions. We think that some of the learned lessons can be useful for works that need to build a BFT SCADA Master from a single server:

a) SMR determinism: The availability of SCADA systems is a major concern for utility companies. Although primary-backup configurations are more attractive as they use fewer resources, the response-time to failures could compromise the correct functioning of critical infrastructures. Using active replication (e.g., SMR) solves part of this problem, as there is no need to change the replicas' roles upon failures. In this work, we dedicated a significant amount of effort to guarantee SMR properties. In the integration process, we had to make the SCADA Master execute sequentially, i.e., eliminating asynchronous messages and multi-threading execution.

b) Is BFT replication suitable for SCADA systems? BFT replication is a step further from traditional SMR, with additional costs due to the tolerance of arbitrary faults. Our evaluation indicates a performance loss when compared to the original NeoSCADA. However, the overhead was not introduced by the BFT library itself, as its maximum throughput was reported to be several times greater than ours. The main cause for this performance loss was the message serialization bottleneck introduced to guarantee determinism. Nevertheless, we do not dispute alternatives to our implementation in a way to accommodate BFT into a SCADA with a minor performance impact. For example, by using a BFT library that supports multi-threading (e.g., [17], [18]) or by adding parallel execution support to BFT-SMaRt (as recently done by Alchieri *et al.* [19]).

c) The cost of transparent solutions: We decided to minimize the modifications in both SCADA and BFT library code. As they already have a significant amount of code, i.e., BFT-SMaRt has ≈ 15k lines of code, and Eclipse NeoSCADA has ≈ 875k lines of code. Then, we decided to keep the original SCADA and BFT library design, to simplify the software integration. However, it had an impact on the SCADA Master performance, as placing proxies between the SCADA and BFT library introduced additional processing steps. The alternative would be to integrate both projects more deeply. However, this integration would be far more complex and would limit future changes in both projects.

Acknowledgments

This work was partially supported by the EC through project FP7 SEGRID (607109), by the FCT through the project Abyss (PTDC/EEI-SCR/1741/2014), and LASIGE Research Unit (UID/CEC/00408/2013).

References

[1] S. Surisetty and S. Kumar, "Is McAfee SecurityCenter/Firewall Software Providing Complete Security for Your Computer?" in *Procs. of the Int. Conf. on Digital Society*, 2010.

[2] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *Procs. of the IEEE Symp. on Security and Privacy*, 2010.

[3] N. Falliere. (2010) Exploring Stuxnet's PLC Infection Process. http://www.symantec.com/connect/blogs/exploring-stuxnet-s-plc-infection-process. Symantec.

[4] Kaspersky Lab. (2017) The State of Industrial Cybersecurity 2017. https://go.kaspersky.com/rs/802-IJN-240/images/ICS

[5] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems," in *Procs. of the USENIX Conf. on USENIX Annual Technical Conf.*, 2010.

[6] B. Calder *et al.*, "Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency," in *Procs. of the ACM Symp. on Operating Sys. Principles*, 2011.

[7] J. Corbett *et al.*, "Spanner: Google's Globally Distributed Database," *ACM Trans. on Comp. Sys.*, vol. 31, no. 3, 2013.

[8] P. Veríssimo, N. Neves, and M. Correia, *Intrusion-Tolerant Architectures: Concepts and Design*. Springer, 2003.

[9] J. Kirsch, S. Goose, Y. Amir, D. Wei, and P. Skare, "Survivable SCADA Via Intrusion-Tolerant Replication," *IEEE Trans. on Smart Grid*, vol. 5, no. 1, 2014.

[10] A. Babay, T. Tantillo, T. Aron, M. Platania, and A. Y., "Network-Attack-Resilient Intrusion-Tolerant SCADA for the Power Grid," in *Procs. of the IEEE/IFIP Int. Conf. on Dependable Sys. and Networks*, 2018.

[11] Eclipse. (2017) NeoSCADA. https://www.eclipse.org/eclipsescada/.

[12] A. Bessani, J. Sousa, and E. Alchieri, "State Machine Replication for the Masses with BFT-SMaRt," in *Procs. of the IEEE/IFIP Int. Conf. on Dependable Sys. and Networks*, 2014.

[13] J. Rose. (2015) How E.ON uses open source IoT technology for monitoring & control of their renewable energy plants. https://www.eclipsecon.org/europe2015/session/how-eon-uses-open-source-iot-technology-monitoring-control-their-renewable-energy-plants.html.

[14] M. Castro and B. Liskov, "Practical Byzantine Fault-Tolerance and Proactive Recovery," *ACM Trans. on Comp. Sys.*, vol. 20, no. 4, 2002.

[15] F. Schneider, "Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial," *ACM Comp. Surveys*, vol. 22, no. 4, 1990.

[16] M. Garcia, A. Bessani, I. Gashi, N. Neves, and R. Obelheiro, "Analysis of operating system diversity for intrusion tolerance," *Software: Practice and Experience*, vol. 44, no. 6, 2014.

[17] R. Kotla and M. Dahlin, "High throughput Byzantine fault tolerance," in *Procs. of the IEEE/IFIP Int. Conf. on Dependable Sys. and Networks*, 2004.

[18] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, and M. Dahlin, "All About Eve: Execute-verify Replication for Multi-core Servers," in *Procs. of the USENIX Conf. on Operating Sys. Design and Implementation*, 2012.

[19] E. Alchieri, F. Dotti, O. M. Mendizabal, and F. Pedone, "Reconfiguring parallel state machine replication," in *Procs. of the IEEE Symposium on Reliable Distributed Systems*, 2017.