Using Digital Twins as an Upgrade Path for Critical Infrastructure Control Systems

Huzaifah Nadeem
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA, USA
h.nadeem@pitt.edu

Amy Babay
School of Computing and Information
University of Pittsburgh
Pittsburgh, PA, USA
babay@pitt.edu

***Abstract*—The threats to critical infrastructure systems, such as power grid control systems, are increasing. Existing fault tolerance and security best practices are no longer enough and we will see more and more successful attacks. Prior works have shown how to build intrusion-tolerant systems, however, these system designs have not made it into practice because performing upgrades of critical infrastructure systems is difficult and costly, and operators typically prioritize stability and availability over resilience to emerging threats. Operators do not want to perform upgrades that may cause downtime, and they want to work with well established and tested systems. In this work, we present a new approach to this problem, using intrusion-tolerant digital twins as a path to disruption-free transition towards more resilient systems.**

I. Introduction

Threats against critical infrastructure control systems, such as power grid Supervisory Control and Data Acquisition (SCADA) systems, are increasing [1]. There is a need to upgrade these systems to make them resilient against newer, more sophisticated attacks as prior works have shown [1], [2]. However, system operators face challenges in upgrading SCADA systems. Due to the reliance of other infrastructure on these systems, any disruptions must be minimized. Currently, upgrading these systems to proposed attack-resilient architectures will require a complete reset of the systems, which will incur downtime. Moreover, in the future, the same process will likely have to be done when the systems are further upgraded to address even newer emerging threats. Therefore, it is important to have a built-in capability that allows a quick, disruption-free upgrade path now and in the future. We aim to address this problem in this work.

The high-level idea of our approach is to run multiple live-running systems in parallel. We refer to each of these running systems as an 'instance'. Only one of these instances is the *active* instance, whereas the rest are *digital twins* that serve as shadow system instances. The twins work alongside our active instance towards processing the application's inputs. The difference between the twins and the active instance is that the final outputs sent to the SCADA clients are only from the active instance, while those from the twins are only logged. This will allow us to use, say an alternative (more resilient) architecture at the twin, while our current architecture runs as the active instance. The system operators can run the system this way and collect logs to analyze any important considerations before upgrading to use this new architecture as the active instance. In other words, the digital twin will provide us with a simple upgrade mechanism to improve the ease of transition towards newer architectures by giving more confidence to the system operators. This is a long-term benefit that will help us upgrade these systems now, as well as in the future.

Having multiple running instances also provides us with an additional, more immediate, benefit. The twin instances can be significantly different from our active instance. For example, they may use different architectures, perhaps with the active instance using a primary-backup based approach for crash tolerance, while the twin instance uses a Byzantine Fault Tolerant consensus protocol to replicate the servers. Another possibility is that the twin can be differently-diverse at the code level from the active instance, similar to [3]. Therefore, running the twin instances in parallel to the active instance provides us with a richer set of data that can analyzed to detect intrusions or anomalies. Thus, if an instance is compromised and does something that it is not supposed to do, we should be able to judge by comparing what the other twin instances are doing with that particular input. If a server is compromised and the attacker tries to introduce commands that the other servers have not seen, it will be detected fairly easily. Similarly, our design can be extended to have an additional system component that processes our live instance logs to do even more complicated analyses to detect more subtle clues. However, we leave that for future work.

To summarize, as a result of following our approach, systems will be able to:

1) Operate intrusion-tolerant digital twins in parallel to existing SCADA systems
2) Compare outputs of intrusion-tolerant digital twins to existing system instance logs to support anomaly detection, including both intrusions and benign faults
3) Use intrusion-tolerant digital twins to transition to intrusion-tolerant architectures with minimal interruption
4) Quickly transfer the control to digital twins as a form of dynamic reconfiguration

Our goal with this work has been to develop intrusion-tolerant digital twins of SCADA systems, and create frameworks to shift control to the digital twins to enable seamlessly transitioning to more resilient system architectures without interruptions, and without needing to redesign and replace that infrastructure all at once.

This work will contribute towards improving the security of critical infrastructure by providing a path to adopting intrusion-tolerant architectures that can withstand sophisticated attacks that succeed in partially compromising the system. Despite previous research on intrusion-tolerant architectures, the need for clean-slate redesign of existing systems has been a barrier to adoption. Using intrusion-tolerant digital twins provides a path for seamless adoption of these techniques, as well as for future upgrades. Utilizing our approach of intrusion-tolerant digital twins and control transfer strategies will make systems more resilient to emerging threats, making power grid control systems that can operate through successful attacks and help transition today's vulnerable SCADA systems to intrusion-tolerant architectures.

This paper is organized as follows. Section II gives an overview of SCADA systems and Section III discusses how we use Digital Twins as an upgrade path for SCADA systems. We describe some of the challenges we faced while implementing a prototype using our approach in Section IV. That is followed by a discussion of related works and some future directions in Section V and Section VI, respectively. Section VII concludes this work.

II. An Overview of Supervisory Control and Data Acquisition (SCADA) Systems

SCADA systems control and operate power grid infrastructure and are therefore critical to other systems as well. SCADA systems, in general, have the following clients:

- Remote Terminal Units (RTU)
- Programmable Logic Controllers (PLC)
- Human Machine Interfaces (HMI)

RTUs and PLCs communicate with the substation infrastructure to collect data and control the substation equipment. The HMIs are where the operator sends any commands to the substations and the HMI also displays the current status of the infrastructure.

The clients talk to the servers, called *SCADA Masters* (SMs), which are responsible for processing 'updates' and 'commands'. RTUs and PLCs send *updates* reporting the status of power grid equipment to SMs, whereas the HMIs send *commands* to change the state of the equipment (e.g. to trip or close circuit breakers). A SCADA system can have one or more SMs depending on the specific architecture being used.

A. Resilience in SCADA Systems

Typically, to make a SCADA system tolerant to crashes, a primary-backup-based approach is used. This includes having two SMs, where one is considered the primary and the other is the backup. If the primary crashes, then control is automatically transferred to the backup. For site-level crash-resilience, there can be an additional backup site which can then have its own primary and a backup server. With such a deployment, if the primary site becomes unavailable as a whole, such as due to a network fault, the backup site can keep the system functional. Further, the idea can be extended by using a Crash-Fault Tolerant consensus protocol to tolerate a configurable number of server crashes.

In practice, however, crash-fault resilience is not enough because these SCADA systems are routinely attacked [1]. Therefore, intrusion-tolerance against Byzantine Faults [4] is required. To make a SCADA system intrusion-tolerant, a number of SMs are deployed. These SMs communicate with each other and a Byzantine Fault Tolerant (BFT) consensus protocol is used to order all the servers' inputs. The distributed systems research community has already designed such intrusion-tolerant SCADA systems based on BFT replication to withstand cyberattacks [2], [5]–[7].

B. Spire – an Intrusion-Tolerant SCADA system

In this work, we use Spire as our base SCADA system for developing our approach. Spire [2] is an open-source Intrusion-Tolerant SCADA system that is resilient to both system-level compromises and network-level attacks and compromises. Spire uses Prime [8] as its BFT protocol and Spines overlay networks [9] for network-attack resilience. Figure 1 shows the layout of Spire.

Spire, being a SCADA system, has the same components as described earlier. The clients are the HMI, RTUs, and PLCs. The RTUs and PLCs use DNP3 and Modbus protocols for communication. Therefore, there is a proxy that is used to translate the protocols for the rest of the system. The proxy may have one or more RTU/PLC clients that it is responsible for. Another role of the proxy is to verify signatures of the messages going to its clients and sign messages going out to the SMs. The clients communicate with the rest of the system over the Spines *External* Network. The system instance consists of a number of 'replicas'. Each 'replica' has a SM and a Prime daemon. The SM handles the application logic, while the Prime daemon handles the consensus protocol logic. We usually refer to the SM and its associated Prime daemon, collectively, as a *replica*. The replicas use the Spines *Internal* Network for consensus protocol related communication with each other.

The required number of replicas depends on the threat model. For Spire, you need a total of $n = 3f + 2k + 1$ replicas to tolerate f Byzantine faults and k otherwise unavailable replicas. The replicas can be distributed across sites for site-level resiliency. The number and distribution of the replicas is referred to as the *configuration* of the system instance.

III. Our Approach – Digital Twins as an Upgrade Path

A. Digital Twins

The general concept of a digital twin is to have another (digital) version of a system running alongside the main system. However, the exact implementation of the 'digital

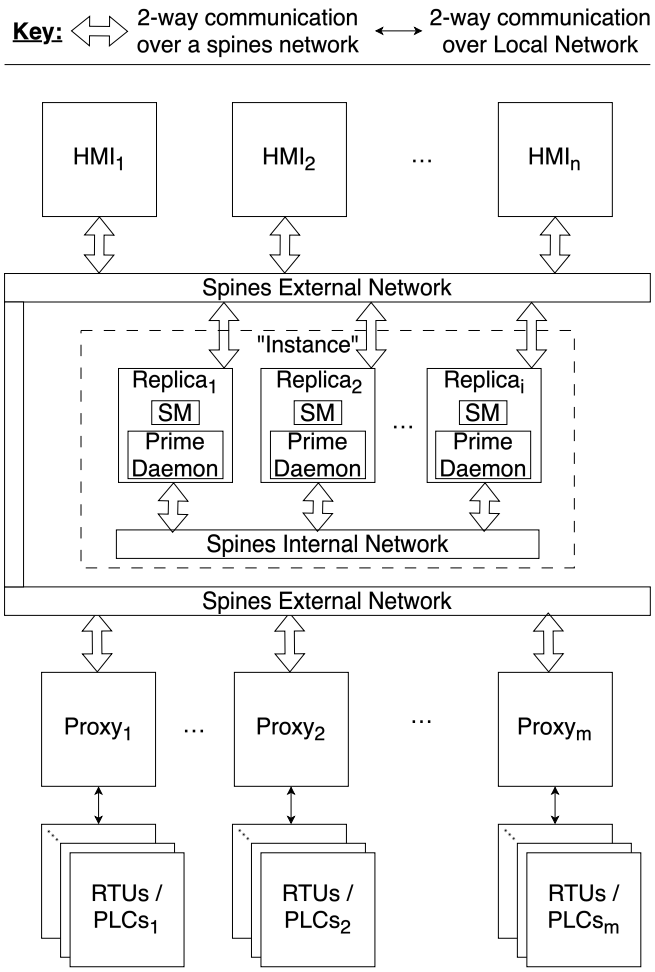

Fig. 1. Layout of the SCADA System Spire

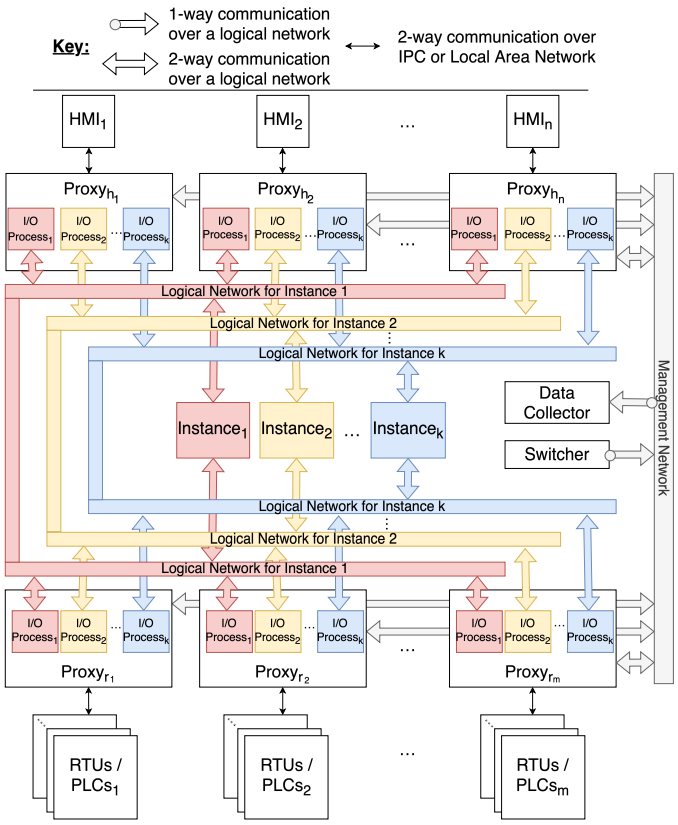

Fig. 2. The layout for our design of a SCADA system with Digital Twins. For our prototype with Spire, internally, the "Instance" components are exactly the same as the "Instance" in Figure 1. The management network and the logical networks for the instances are deployed as an overlay network using Spines, as does Spire.

twin' concept can vary. For instance, in some works, the twins are created using data-driven models or models created using the laws of physics and emulate hardware processes in software [10], [11]. In other cases, especially when the main system is already software-based, the twin is exactly the same system as the main system except that it is not directly in control [12]. We consider the latter kind of digital twins in this work.

B. Technique Overview

Our main idea is to have parallel live-running digital twins of our system. In our case that means to have multiple instances running at the same time as the active instance. For this, we want to avoid modifying the core components of the system (such as the SMs, HMIs, PLCs, etc) as much as possible. To allow us to mirror our incoming messages to go to the twins, we need a component to act as a 'gateway' for all our instances. Apart from this, we need a place to store our instances' logs, and a way to communicate with the 'gateway' component to transfer control between the instances, or to add or remove twins. We implemented our SCADA digital twin prototype in Spire as described below. An overview of Spire is given in Section II-B.

C. Modifications to Spire

In our prototype, we can have a number of instances, say $k \in \mathbb{Z}^+$, running at the same time. These k instances include a single active main instance and $k-1$ digital twins. As before, we can still run a number of HMIs, say n, and RTUs/PLCs, say m. To allow this to happen, we modified or added the following new components, as shown in Figure 2, to the Spire codebase:

- Proxy
- Input/Output (I/O) Processes
- Data Collector
- Switcher
- Management Network

Apart from the modified Proxy and the new I/O Processes, the rest of the new components are all optional in the sense that we can run the system with twins but we would not have the complete benefit of doing so. Note that, all these changes are backwards-compatible and our version of Spire can still function as it did before without digital twins. We can do so by running it with $k = 1$ instances, corresponding to a single

active instance with no twins. The newly added or modified components are described next.

1) Proxy: For our prototype, we repurposed the Proxy to have the additional responsibility of acting as the 'gateway'. The Proxy in Spire is naturally suited for this task because all the RTUs/PLCs' messages pass through it and we only need to add some additional logic to manage multiple instances. Additionally, we also use this new Proxy component to take over the responsibility of communicating with the SMs from the HMI as well. This makes client-system communication consistent across the whole system as well as limits the places requiring the additional logic to handle multiple instances. The HMI, now, is only responsible for displaying the infrastructure status and receiving commands from the operator to send to the SMs, via its Proxy. Therefore, after adding the new functionality to the Proxy, it is the only required component to run the system with the digital twins. This simplifies the deployment and avoids requiring SMs to track information about the digital twin configuration. This is beneficial, since we want to avoid modifying the SMs, and if there are any intrusions, having this information at the SMs could be useful for an attacker to understand the system layout. Similar to our incoming messages, the Proxy also receives all the outgoing messages from all the instances, but forwards only the active instance's messages to the clients.

2) I/O Processes: Another major change to the proxy is the introduction of our new I/O Processes. These are not new components per se but rather, a part of the Proxy itself. These are processes that are in charge of communication with a particular instance and we will have one for each instance. These are required because we would like to keep the system live during any control transfer, or the addition or removal of any new instances, and therefore it is important that there is a mechanism to support that at the Proxy as that is what all the instances 'connect' to. Moreover, the twin instances do not necessarily have to have the exact same protocols as the active instance. To allow such a possibility, the I/O processes can manage any necessary translation that may be required. This translation capability allows us to deploy twins using new communication protocols without modifying the clients; for example, an I/O process can translate an insecure legacy communication protocol to a new secure protocol used by an intrusion-tolerant twin instance. Since the communication with each instance is completely isolated as a process at each Proxy, we can keep the proxies online while we are adding or removing an instance – only an I/O process has to be added or removed. Sections III-E and IV-A discuss these points in more detail.

3) Data Collector: We also include a new 'Data Collector' component. The Proxy has an additional responsibility of sending a copy of all its incoming and outgoing messages to the Data Collector for logging purposes. These logs are useful to find out whether or not all the instances are working correctly and allow the operator to confidently upgrade the active instance to a different configuration by comparing it to a live-running twin in that configuration. An intrusion-detection algorithm can also run at, or with, the Data Collector.

4) Switcher: To allow the operator to transfer control from one instance to another, there is a new component called the 'Switcher' through which the operator can send messages to the Proxies to transfer the control. This control transfer can be useful if or when the operator would like to upgrade the system. To avoid any disruptions during the upgrade process, one of twins can keep the system functioning. Having the Switcher also allows us to add or remove twins from our overall system by sending messages to the Proxies about any relevant information for this purpose.

5) Management Network: To allow the necessary communication between the Proxies, the Data Collector, and the Switcher, we add an additional network to the system called the *Management* network. The management network can be a separate physical network, or a logical overlay network on top of the existing physical network. For our prototype, we use Spines [9] to deploy the management network as an overlay. On this network, the Proxies send the copy of the messages to the Data Collector and the Switcher uses multicast to send messages from the operator to all the Proxies.

D. Control Transfer

The goal of this is work to allow seamless system upgrades from one active instance, that uses a particular configuration, to a new instance using a different and potentially more resilient configuration. For SCADA systems it is essential that we do these upgrades without any disruption as a lot of other infrastructure relies on functioning SCADA systems. Therefore, over the course of running multiple instances to observe how a different configuration would behave, when it is time to transfer the control to a twin, the operational need to do upgrades without disruption has to be considered. This process works as follows: to transfer control, the operator can send a message to the Proxies, via the Switcher, for this purpose. Upon receiving this message, the Proxies update their local data structures to designate a twin instance as the new active instance. For the rest of the incoming and outgoing messages, the Proxies then use this information to route the correct instance's messages, that it receives through that instance's specific I/O Process, to the clients.

Note that since the Switcher sends these messages over a network, it is possible for a Proxy to be left behind briefly. This, however, is not a problem as our ground truth state is always reflected by the clients' state and the servers have the necessary logic to handle any missed packets. The logic includes updating the server state upon receiving acknowledgments in the form of client state updates, or polling the clients when necessary, such as during state transfer or initialization. The same logic will eventually update the servers' state should a situation like this occur.

E. Addition and Removal of Twins

It is important to note that the threat models for these SCADA systems are always evolving and we can expect to observe newer threats in the future. Therefore, there is a need to

have a functionality of evolving our defenses alongside these new threats. Prior work highlights some of these emerging threats, such as in [1], where the authors consider a well-timed cyberattack in the aftermath of a natural hazard, such as a hurricane. To defend against such attacks, the authors gave their insights into possible configurations that are more useful to defend these threat. Our work keeps this in mind and we allow the functionality of adding or removing additional twin instances as well. We do not intend this digital twin functionality to be a one-off process that is only used once to upgrade the systems but rather to have this functionality as a core part of the system to allow analyzing newer, more resilient architecture, that do not exist yet, as research on the topic progresses.

The process of adding or removing additional instances, the twins, looks as follows. The operator uses the Switcher to send messages to all of the Proxies that an instance is to be added or removed. If the operator would like to add a new twin, then based on Switcher's message, all the Proxies in the system fork off a new I/O Process as described in the message that will then set up its communication with the new twin instance. One reason of having these I/O Processes separate from the core part of the Proxies was to allow having a separate binary that can be used for each process that is distinct from the Proxy binary and may not be the same as the other I/O Processes. This can be useful in the situations where a twin instance is different enough from active instance to require some additional processing of client messages, such as protocol translation, or perhaps the twin instance has compiled-in configuration parameters that cannot be dynamically added to a running Proxy. Therefore, when an operator is sending a message through the Switcher, the operator can include all the necessary information in Switcher's message, such as including a binary to use for this I/O Process, or describe where to find a binary on the Proxy's disk to use, or reuse if it already exists. In any case, the Proxy will not have to be turned off, only the new I/O Process will have to be forked. This allows us to add a new twin instance without any disruptions to the existing instances.

Similarly, if the operator would like to remove a twin instance, a similar type of message is sent by the Switcher, based on which, the Proxies gracefully shuts off the relevant I/O Process.

IV. Challenges

During the implementation of our prototype, we faced a number of challenges. Some of the challenges were system-specific issues related to the Spire codebase, whereas others were more general in nature. The system-specific issues are discussed next in Section IV-A. Section IV-B discusses the more general challenges regarding the state divergence between the instances. We discuss several approaches that we considered to resolve this. However, it remains an open question as to which approach is the right one in general, and which are preferred in a specific context. The approach we went with is described in Section IV-B4.

A. System Implementation-specific issues

Based on the exact implementation of the system, it is possible that it may not be directly compatible with the clients. This can happen due to protocol difference or a more codebase-level detail such as compile-time linking of various files. We came across one such issue with using Spire. Spire uses macro defines in its C/C++ codebase for certain configuration parameters. This effectively means that Proxy would have references to various data structures in other parts of the codebase as it was not designed with such an extensibility in mind. If we wanted to support running multiple instances or to add or remove instances, this would cause the whole system needing to reset for this task. To help resolve this issue, the I/O process are now used. Having I/O processes help us isolate the instances and any resolutions are handled at the I/O process.

B. Application State Divergence

Because Cyber-Physical systems can have components deployed over a wide-area network, delays and packet reordering are possible. This can result in slight deviations in the order of incoming commands from a client to the instances. Therefore, as a result, it is possible that the instances end up processing the commands in different orders (e.g. for Spire, due to Prime ordering the commands differently), resulting in a deviation of the instances' state relative to each other. We considered the following approaches to address such issues:

1) Naive Approach: Forcing ordering of incoming messages: We can have a queue for the outgoing messages at each proxy and an established deterministic merge procedure that can order the updates from different proxies. However, this approach will cause additional issues as these queues can act as a bottleneck for our instances' messages and can impact performance. SCADA systems generally require application guarantees such as that a command has to be processed within 100ms [2]. Such an approach can make it hard to provide such guarantees. Moreover, if we rely on the proxies for the orderings, then our scope of threat model will also have to take into account proxy compromises on top of server compromises. Therefore, this approach may not be useful in an intrusion-tolerant environment without rethinking the complete threat model and application guarantees.

2) Using Inputs, Outputs, and Orderings to establish correctness: An alternative approach could be to log all of our inputs and outputs as well as the way our consensus protocol ordered the updates. If we have these logs of all the inputs, outputs, and the orderings from each of the instances, it is possible to verify the correctness of any instance's state by checking the possible input-output mappings of commands based on the consensus orderings. Using the orderings alongside the inputs and outputs can help us verify whether or not a state divergence between any two instances is benign or not. However, the main issue with this approach is that it is very invasive and the instances' code will need to be significantly modified which goes against our overall idea of making the upgrade process simpler.

3) Utilizing Complete History and Possible Permutations in Ordering: Another possibility is to utilize the complete history of all of our updates and commands for each instance. The data collector already logs all the commands and updates and therefore, this would not require any additional changes to the system. Using this history, we can confirm the validity of states by checking possible permutations of orderings. A process can work at or with the data collector to confirm whether or not a particular state is possible by verifying that the required inputs exist for the outcome, as well as, that there is at least one particular correct ordering that can take the state, given the inputs and the orderings, to the specific outcome state. This, however, will be computationally expensive especially considering that these systems are long-running. However, depending on the specific application, it may possible to have some cut-off for the history to keep the number of possible permutations limited.

4) Validating State Correctness through Application-specific Knowledge: Another possible, and likely more practical, approach is to consider application-specific knowledge to confirm the validity of states.

In our digital-twin-based architecture, if an attacker controls an instance, they can only order operations differently if they want to remain hidden. If they introduce any new commands, then that would be easily recognized by comparing with the other instances.

For our specific SCADA application, an attacker who causes updates to be ordered differently cannot cause divergence between systems' states. In our application, all the possible commands are binary in nature and there is no 'flip' command. That is, for example, we can have a number of circuit breakers at the substation, each of which may be commanded to be turned on or turned individually, but there is no single command to flip it from on to off, or the other way around. This observance is an important consideration for our system and it makes validating states easier. This is because, reordering a number of 'flip' commands around other 'on' and 'off' commands can result in a completely different outcome as a 'flip' operator depends on the previous state whilst an 'on' command on an already 'on' breaker does nothing. For our specific application, the only thing an attacker can do, without getting caught, is to reorder operations for a breaker or for operations of different breakers relative to each other.

Another key thing to note for our application is that only the active instance's updates/command are applied to the clients and even if a twin orders something differently: first, that ordering is a valid state as well; secondly, no real long-term divergence will occur because only the active instance's state will be applied to the clients, which is our ground truth. To elaborate more on this, an instance gets regular updates from the clients regarding what the actual state at the substation infrastructure is. Therefore, even if the sequence of commands issued by this instance would have resulted in a different state, it still adopts the actual state received from the PLCs and RTUs. So, the twin will be at the same state as the active instance eventually. The servers, i.e. the SMs, can handle such momentary deviations as they are made to keep in mind that the messages may get lost over the network and hence it only applies the final update upon receiving a confirmation from the client as noted in Section III-D.

Another key piece of information is that it takes some time for the proxies to receive messages from the switcher and a proxy may still, for a time, consider a particular instance to be the active one even though the switcher has already issued a command to make a different instance the active one. However, that, again, is not a problem because: first, as previously mentioned, SMs can tolerate messages getting lost; secondly, we know the ground truth at the clients, which in the case of a SCADA system are the PLCs and the RTUs, and eventually the updated state will be reflected across the instances.

V. Related Works

The work [3] explores some similar ideas. They consider running variants of a system that have different diversity techniques applied to help verify that the systems are all functioning as they should. Our approach can be extended to incorporate system-diversity as well.

The work in [13] is similar in spirit to our work, although the approach is very different. It uses cloud resources to offload some parts of the system to make it more manageable to deploy larger systems. Therefore, if you have a simpler architecture and you want to upgrade it to a more complex system and you are fine with some downtime, then that is one possible way of doing it. Our approach is to have digital twins run alongside your active system instance to help gauge the instance you want to upgrade to and build more confidence in the new architecture, and then finally transfer the control without disruption. Our work can be extended to have a twin partially use cloud resources as [13] does.

In principle, we can think of our approach as a live 'beta' testing or in some sense a version of A/B testing from software engineering. However, there are key differences, such as the fact that A/B testing is generally understood as a user-facing testing of different system variants, and the users may not be aware of the variant. In our case, we do not have users as such but rather operators who are purposefully testing live variants.

VI. Future Work

In the future we would like to have a real-world case study where we use this technique and approach as an upgrade path for a deployed system.

Another natural direction for future work is to have virtual Cyber Digital Twins (CDTs) at the client level as well. In that case, these intrusion-tolerant digital twins will enable applying intrusion tolerance techniques to a larger number of system components via virtualization. This may be especially useful in cases where adding additional physical clients leads to cost scaling issues and having virtual twins will allow us to test with a larger number of possibilities and placement algorithms. Figure 3 shows an example of how this could look like.

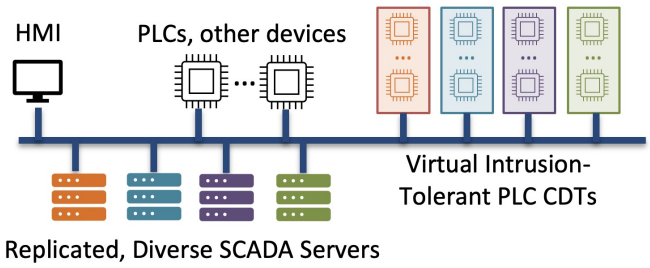

Fig. 3. Intrusion-Tolerant SCADA system with Intrusion-Tolerant PLC virtual Digital Twins

As referred to in earlier sections, we can have more complicated intrusion detection techniques applied live on the instances' logs at the data collector. Additionally, the detection may be used along with control transfer strategies to automatically transfer control if some sort of anomaly is detected. This may be considered a form of dynamic reconfiguration.

Lastly, in this work we only considered SCADA systems. In the future, the scope can be increased and the ideas applied to other control systems or Cyber-Physical systems more broadly.

VII. Conclusion

In this work, we have presented our approach and experience on using digital twins to make system upgrades easier and more manageable. We introduced our approach on a real SCADA system, Spire, and the changes we needed to make to make it possible to run digital twins alongside the active system instance. We also described the issues and challenges we encountered and some possible ways to address them.

Acknowledgment

This material is based upon work supported by the Department of Energy under Award Number DE-CR0000039.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

[1] S. Bommareddy, M. Khan, H. Nadeem, *et al.*, "Tolerating compound threats in critical infrastructure control systems," in *2024 43rd International Symposium on Reliable Distributed Systems (SRDS)*, 2024, pp. 66–79. DOI: 10.1109/SRDS64841.2024.00017.

[2] 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)*, 2018, pp. 255–266. DOI: 10.1109/DSN.2018.00036.

[3] S. D. Stoller and Y. A. Liu, "Algorithm diversity for resilient systems," in *Data and Applications Security and Privacy XXXIII*, S. N. Foley, Ed., Cham: Springer International Publishing, 2019, pp. 359–378, ISBN: 978-3-030-22479-0.

[4] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982, ISSN: 0164-0925. DOI: 10.1145/357172.357176. [Online]. Available: https://doi.org/10.1145/357172.357176.

[5] W. Zhao and F. E. Villaseca, "Byzantine fault tolerance for electric power grid monitoring and control," in *2008 International Conference on Embedded Software and Systems*, 2008, pp. 129–135. DOI: 10.1109/ICESS.2008.13.

[6] J. Kirsch, S. Goose, Y. Amir, D. Wei, and P. Skare, "Survivable scada via intrusion-tolerant replication," *IEEE Transactions on Smart Grid*, vol. 5, no. 1, pp. 60–70, 2014. DOI: 10.1109/TSG.2013.2269541.

[7] A. Nogueira, M. Garcia, A. Bessani, and N. Neves, "On the challenges of building a bft scada," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 163–170. DOI: 10.1109/DSN.2018.00028.

[8] 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, 2011. DOI: 10.1109/TDSC.2010.70.

[9] D. Obenshain, T. Tantillo, A. Babay, *et al.*, "Practical intrusion-tolerant networks," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016, pp. 45–56. DOI: 10.1109/ICDCS.2016.99.

[10] F. Akbarian, E. Fitzgerald, and M. Kihl, "Synchronization in digital twins for industrial control systems," eng, in *16th Swedish National Computer Networking Workshop (SNCNW 2020)*, 2020. [Online]. Available: https://lup.lub.lu.se/record/a1021dfd-c7f8-464c-8f42-f7027423f71a.

[11] S. Haag and R. Anderl, "Digital twin – proof of concept," *Manufacturing Letters*, vol. 15, pp. 64–66, 2018, Industry 4.0 and Smart Manufacturing, ISSN: 2213-8463. DOI: https://doi.org/10.1016/j.mfglet.2018.

02.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2213846318300208.

[12] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE Access*, vol. 8, pp. 108 952–108 971, 2020. DOI: 10.1109/ACCESS.2020.2998358.

[13] M. Khan and A. Babay, "Making intrusion tolerance accessible: A cloud-based hybrid management approach to deploying resilient systems," in *2023 42nd International Symposium on Reliable Distributed Systems (SRDS)*, 2023, pp. 254–267. DOI: 10.1109/SRDS60354.2023.00033.