Work-in-Progress: A Formal Approach to Verify Fault Tolerance in Industrial Network Systems

Alessio Sacco
Politecnico di Torino
alessio_sacco@polito.it

Guido Marchetto
Politecnico di Torino
guido.marchetto@polito.it

Riccardo Sisto
Politecnico di Torino
riccardo.sisto@polito.it

Fulvio Valenza
Politecnico di Torino
fulvio.valenza@polito.it

***Abstract*—Distributed systems are extremely difficult to design and implement correctly because they must handle both system correctness and device failures. Most of the work focuses on the first aspect, and in particular, on the correctness of security and network configuration. The large demand for availability and reliability for critical services is actually pushing new architectures that tolerate failures, but a-priori analysis of redundancy and recovery features is still limited. To this end, we present a framework to design and formally verify the persistence of network properties, even in case of failures. The solution considers both nodes and links failure, and it is based on a formal model that takes both network topology and network device configurations into account. In contrast, most of the existing approaches only consider network topology. By analyzing the formal model, the framework can check whether the specified network services are still available after failures, and in case of success, it outputs a possible configuration of the devices to be used for automatic recovery.**

***Index Terms*—fault tolerance, network reachability, industrial network systems**

I. Introduction

Industrial Control Systems (ICSs) are physical and engineered systems whose operations are monitored, coordinated, controlled, and integrated by a computing and communication network. Examples of ICSs include medical devices and systems, aerospace systems, transportation vehicles and intelligent highways, defense systems, robotic systems, process control, factory automation, building, and environmental control, and smart spaces.

ICSs are often considered critical systems. In particular, the dependability of their network (the Industrial Network System) is fundamental [1], as well as its proper management, which often requires a long design phase to avoid most of the potential risks [2]–[7]. In fact, ICSs must interact with the physical world and must operate safely, securely, efficiently, and in real-time. In addition to these properties, reliability and fault-tolerance are crucial for dependable systems, which are spread across many industrial and critical scenarios.

One way to satisfy the requirements and to operate correctly even in the presence of network failures is to introduce redundancy for some of the network elements. Hence, the network must first be designed with a correct topology; second, it must implement the proper logic to react to failure quickly and in the appropriate way. These fault-tolerant and dependable architectures must be able to withstand multiple faults, but the service must continue to satisfy the given requirements, which might be not straightforward. When configuring the network of an ICS, indeed, the administrator might need to guarantee, at the same time, protection against cyber threats, and hard real-time constraints. Sometimes these requirements may be contrasting, because, for example, the introduction of an additional security function in a communication path either may cause higher latency not compatible with the real-time constraints for that path or may reduce the number of available paths by affecting the fault tolerance strategy results. Furthermore, the widening number of functionalities and the complexity of newly deployed networks further increases the effort required for the network configuration process. This makes the configuration activity too cumbersome and error-prone for humans to be managed.

This paper proposes a formal verification approach, able to guarantee that the designed network satisfies the required level of reliability in an automatic way. To address this challenge, our preliminary solution verifies fault-tolerance properties in complex networks by applying the idea that given reachability properties must hold even after a fault. Specifically, to formally verify the reachability and, consequently, the fault tolerance, we make use of First-Order Logic (FOL) to model the nodes of the network and also take into account the configuration of devices. This model is then used by a Satisfiability Modulo Theories (SMT) solver to determine the satisfiability of the problem, i.e., redundancy exists and the alternate path provides the same properties of the initial one. By leveraging the proposed framework, industrial network systems can be designed and constructed to guarantee reliable real-time applications in industrial automation.

Our model is able to deal with the configurations of network devices. Hence the verification process is more complex than a simple path redundancy check since we assure the redundancy can be exploited by nodes and the backup path gives the same services of the original one. To the best of our knowledge, this is the first work that simultaneously verifies reliability in terms of route redundancy and reachability properties. In our view, this is fundamental in industrial networks, where simple topologies are made up of many functionalities and nodes, e.g., Supervision, Control and Data Acquisition (SCADA) servers, Industrial Firewalls, Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs), to cite a few. Indeed, as pointed out by other work [8], [9], the largest sources of failures for complex networks are misconfigurations.

The paper is structured as follows: we start describing the stat-of-the-art and other solutions related to network fault-tolerance in Section II, then, we explain the problem and the methodology in Section III. In Section IV we present a motivating use case in the industrial networks. Finally Section V concludes our paper.

II. Related Work

According to [10], fault tolerance is the ability of a system or component to continue normal operation despite the presence of hardware or software faults. As described above, fault-tolerant networks require some type of redundancy, which can take many various forms, such as hardware, software, or time redundancy. Formal verification can bring benefits to the design process of complex networks by formally verifying the existence of reachability among critical services (e.g., [2], [11]). Nonetheless, formal methods were rarely applied in the design of computer networks and systems where links and nodes can fail.

In [12] the authors presented a new language for writing fault-tolerant SDN programs by means of regular expressions. This solution compiles programs to instructions for OpenFlow switches by leveraging in-network fast-failover mechanisms. This is orthogonal to our approach, where a verification module makes sure the topology is fault-tolerant and can be combined with solutions such as [12] to connect services without disruption.

Ding et al. presented a framework [13] to design fault-tolerant wireless network control systems, particularly suited for industrial automation applications. However, they mainly focus on the optimization of control actions with the development of a reliable Fault Detection and Identification (FDI) algorithm working at different functional layers.

Dynamic Fault Graphs [14] can be used in this scenario as well. Indeed, the authors conduct the overall system dependability analysis through the entire phases of modeling, structural discovery, and probability analysis. Future behaviors are predicted through probability forecast and are then used to verify the required network properties.

Also the redundancy of trees in a graph was already explored in literature [15]. We share with this work the idea to prevent the disruption of services by design, however, our model considers the behavior of each network node to assess if an alternative path exists that can transmit a packet from a source to a destination.

Verdi [16] is a framework for implementing and formally verifying distributed systems in Coq. We share with this solution the target use case (the distributed systems), but we address the problem from the reachability point of view, generalizing their approach.

Finally, in [17] the authors presented an efficient mechanism to detect if a fault-tolerance property holds by enumerating all the possible counterexamples. Their approach is based upon the concept of Stable Path Problems with Faults (SPPFs), and it allows to verify reachability under faults in standard topologies. However, they focus on the verification of distributed routing protocols, neglecting middlebox devices in their model.

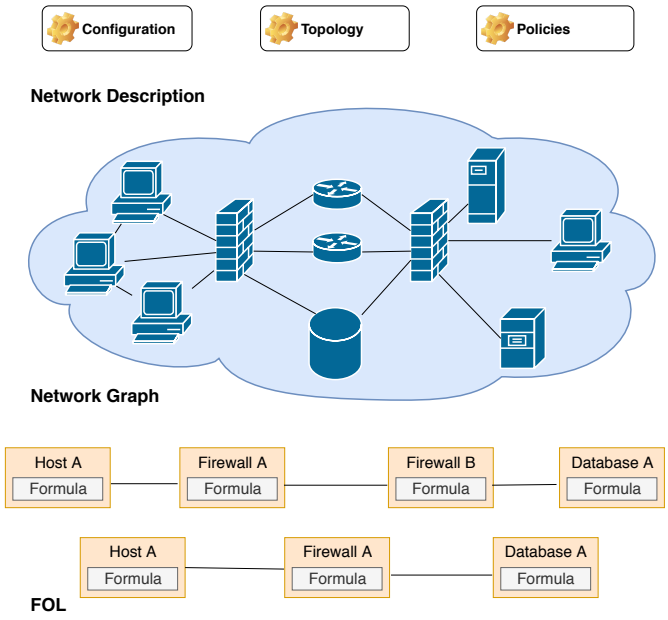

Fig. 1. Workflow of our approach. Logical abstraction as a set of FOL formulas is obtained starting from the physical description. Fault-tolerance property is verified through the FOL representation.

III. General Approach

The goal of the proposed framework is to formally verify that some network reachability properties are satisfied even in the presence of some network failures. The verification process takes into account the forwarding behavior of the network nodes based on their configurations, by leveraging an SMT solver, which, given a set of formulas describing the network, checks whether the reachability condition is satisfiable. To obtain the models required by the SMT solver to validate the network properties, we first need a description of the network. This is usually provided by the administrator who needs to deploy and verify that the proposed design can work properly. This knowledge constitutes the *Network Description*, which includes: *(i)* the network topology, *(ii)* the configuration of network and security devices such as firewalls and VPN terminators, and *(iii)* the required reachability policies.

These policies express the requirements of the administrator, aiming at defining how the network has to withstand the failures of nodes and links. The network administrator can specify that a given service residing on node d can be accessed from node s even if some items, enumerated in a list F, are damaged. Furthermore, the degree of fault-tolerance, k, must be specified as an additional parameter, defining the number of elements that can fail simultaneously. The specification of fault items is done at the physical layer, where the topology is specified as the union of links and nodes, and each node is associated to a network function, e.g., router, switch, firewall.

Once the admin provides the input, the framework extracts the information needed to construct the *Network Graph*, which is an intermediate representation, used to study initial network properties, such as path redundancy. In fact, the graph is examined to find all the paths between the specified s and d, represented as a list of *chains* starting with s and ending with d. However, the presence of multiple paths is not sufficient to guarantee a given level of fault-tolerance, as the configuration of some devices may block the communication on some of them. For this reason, the chains are then converted into First-Order Logic (*FOL*) formulas, where the nodes are expressed by logical conditions modeling their forwarding behavior, also taking their configuration, provided as input by the administrator, into account. This is done by following the same approach described in [2] for network reachability verification. Finally, an SMT solver is used to check whether traffic can actually flow between s and d in each chain, and this result is used to check if fault tolerance policies are satisfied.

In case of a positive outcome, i.e., fault-tolerance guarantee, the output is one or more backup routes available for future use. As we provide this piece of information as a list of nodes, it can be used to set the alternative route when the primary one is down. For example, recent versions of OpenFlow for SDN-enabled networks include support for conditional rules whose forwarding behavior depends on the local state of the switch. When the type on an entry is Fast Failover (FF), each action bucket is associated with a parameter that determines liveness, and the switch forwards traffic to the first live bucket. With these FF entries, a switch explicitly handles the backup case, but the controller program must anticipate every possible failure and pre-compute appropriate paths. Our framework provides the output necessary to properly configure switches in advance, without burdening the network admin in case of failure. Other network protocols offer equivalent concepts and syntax to handle the backup paths; therefore, our output can be exploited in many different contexts according to the specific protocol and architecture. On the other hand, when the result is negative, i.e., fault tolerance is not guaranteed, our tool reports a message stating whether the cause is the absence of a secondary path or the configuration of a device. In the latter case, the tool can detect the node blocking the traffic, and includes this information in the final message.

In Fig. 1 we summarize the aforementioned processing and the workflow required to obtain the final result starting from the administrator input. Although the formality of the model may seem complex to handle for a network administrator, this complexity is hidden to the user who is just required to provide as input a JSON file representing the network description, while the generic FOL formulas for each kind of network node are available inside the framework as a library, as also done in Verigraph [2] for network verification.

It is also worth noticing that, in our approach, we verify reachability on paths rather than on graphs, in line with most existing tools for network verification, e.g., HSA [18] and Verigraph [2]. In such a way, we can deal with simple and easy to treat models. Redundancy can indeed be verified simply by checking whether a secondary path can be considered as an alternative path in case of failure. This procedure allows us to keep complexity low and to provide the response in a short amount of time. In fact, since paths are mutually independent, the SMT solver can verify reachability separately for each path. Solving the problems in parallel provides a speedup over solving all of them sequentially.

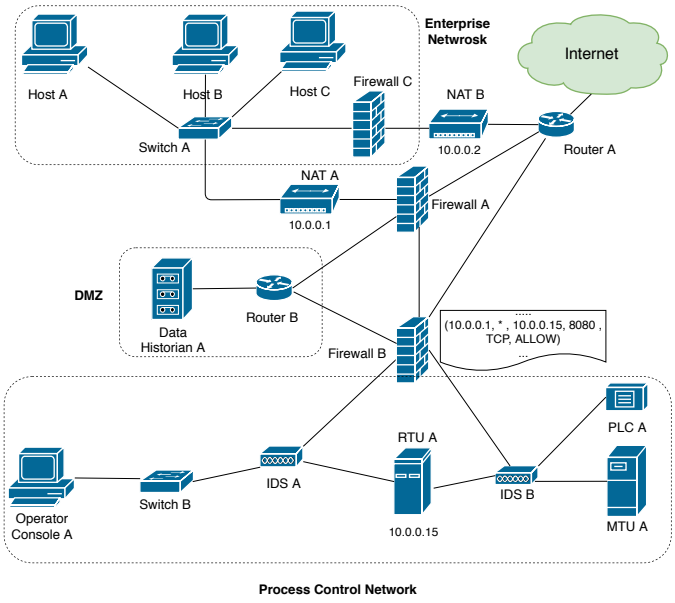

Fig. 2. Example of a typical industrial network where multiple and different functionalities are deployed.

IV. Motivating Example

In order to better motivate our approach that jointly considers both network topology and configuration, we first define a sample use case scenario inspired by previous work on Industrial networks [11], [19], [20]. In particular, we focus on Supervisory Control And Data Acquisition (SCADA) systems, i.e., computer systems for gathering and analyzing real-time data, which are usually deployed to control and monitor entire sites, and, consequently, are often complex systems spread out over large areas. These systems are crucial for industrial organizations since they help maintain efficiency, process data for smarter decisions, and communicate system issues to reduce downtime.

Fig. 2 shows the physical representation of a SCADA network, where fundamental functionalities are deployed, such as Firewalls, Routers, and IDSs. Aside from traditional nodes, more distinctive functions for an industrial scenario are combined together. Remote Terminal Units (RTUs) and the Programmable Logic Controllers (PLCs) automatically perform nearly all the control actions. These hosts execute control functions for a supervisory level intervention, communicate with an array of objects such as sensors, and then route the information from those objects to computers running the SCADA software. For example, the PLC controls the flow of cooling water, while the SCADA system allows any changes related to the flow conditions (such as high temperature,

loss of flow, etc.) to be recorded and displayed. The data acquisition starts at the PLC or RTU level, which involves the reports of equipment status, and meter readings. Data are then formatted to allow the operator of the control room to make the supervisory decisions of overriding or adjusting normal PLC (RTU) controls, by using a remote (or local) console.

Due to the exposure to a wide range of security problems, access to individual sub-networks is secured by firewalls that implement basic network security policies, allowing only the desired traffic. The firewalls separate four different zones: *(i) Enterprise Network,* which includes all the end-hosts and workstations, *(ii) DMZ,* with the database and other services externally available, *(iii) Process Control Network,* where all the industrial services reside, *(iv) Internet,* for the external communications. From the Enterprise Network, it is possible to reach the Internet (except a specific subset of addresses), while from the outside, it is not possible to contact machines in the Enterprise Network (except by administrative roles). Firewall B allows communication just from a certain range of IP addresses. In Fig. 2, Firewall B allows that RTU A with IP address $10.0.0.15$ can be reached at port 8080 from IP address $10.0.0.1$ and from any port. The administrator sets this rule to allow any hosts masked by NAT A (with IP address $10.0.0.1$) to control the RTU A remotely.

This configuration guarantees a working scenario, as the Process Control Network is adequately protected. At the same time, in the case of Firewall A's damage, we can notice how the Enterprise Network can still potentially communicate with RTU A via a secondary route, e.g., Firewall C - NAT B - Router A - Firewall B - IDS A. Nonetheless, NAT B assigns the IP address $10.0.0.2$, and Firewall B, as a consequence, blocks the communication, since RTU A can be accessed only from $10.0.0.1$. This is a pretty basic example where the configuration of security devices impacts the fault-tolerance verification process and motivates the model presented in the following section.

V. Conclusion

This paper presents a novel approach to model networks and formally verify fault tolerance property in an industrial scenario. The proper design and configuration of networks are error-prone and a cumbersome task for the administrator; also tests do not often explore all the possible occurrences, leading to uncertainty and incorrectness. For this purpose, our tool assists the design phase, to achieve a formal verification of fault-tolerance properties, very critical for industrial scenarios.

The preliminary solution presented exploits First-Order Logic to formally verify that reachability holds between specified source and destination. The framework first enumerates all the paths available in the topology, then it analyzes if, even in the presence of faults, the desired services are still reachable. As future work, we further plan to extend this preliminary work by extensively evaluating performance over complex networks, and integrating this framework with the automatic configuration of devices.

References

[1] F. Kargl, R. W. van der Heijden, H. König, A. Valdes, and M. C. Dacier, "Insights on the security and dependability of industrial control systems," *IEEE security & privacy*, vol. 12, no. 6, pp. 75–78, 2014.

[2] S. Spinoso, M. Virgilio, W. John, A. Manzalini, G. Marchetto, and R. Sisto, "Formal verification of virtual network function graphs in an sp-devops context," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2015, pp. 253–262.

[3] C. Basile, D. Canavese, C. Pitscheider, A. Lioy, and F. Valenza, "Assessing network authorization policies via reachability analysis," *Computer and Electrical Engineering*, vol. 64, no. C, pp. 110–131, Nov. 2017.

[4] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2601–2614, Oct. 2017.

[5] A. Panda, O. Lahav, K. Argyraki, M. Sagiv, and S. Shenker, "Verifying reachability in networks with mutable datapaths," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 699–718.

[6] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 300–313.

[7] K. Weitz, D. Woos, E. Torlak, M. D. Ernst, A. Krishnamurthy, and Z. Tatlock, "Formal semantics and automated verification for the border gateway protocol," *NetPL, March*, 2016.

[8] X. Wu *et al.*, "Netpilot: Automating datacenter network failure mitigation," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 419–430.

[9] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb, "A case study of ospf behavior in a large enterprise network," in *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, ser. IMW '02. New York, NY, USA: ACM, 2002, pp. 217–230.

[10] M. Al-Kuwaiti, N. Kyriakopoulos, and S. Hussein, "A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability," *IEEE Communications Surveys Tutorials*, vol. 11, no. 2, pp. 106–124, Second 2009.

[11] G. Marchetto, R. Sisto, J. Yusupov, and A. Ksentinit, "Formally verified latency-aware vnf placement in industrial internet of things," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, June 2018, pp. 1–9.

[12] M. Reitblatt *et al.*, "Fattire: Declarative fault tolerance for software-defined networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 109–114.

[13] S. X. Ding, P. Zhang, S. Yin, and E. L. Ding, "An integrated design framework of fault-tolerant wireless networked control systems for industrial automatic control applications," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 462–471, Feb 2013.

[14] F. Zhao, H. Jin, D. Zou, and P. Qin, "Dependability analysis for fault-tolerant computer systems using dynamic fault graphs," *China Communications*, vol. 11, no. 9, pp. 16–30, Sep. 2014.

[15] M. Médard *et al.*, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM Transactions on networking*, vol. 7, no. 5, pp. 641–652, Oct. 1999.

[16] J. R. Wilcox *et al.*, "Verdi: A framework for implementing and formally verifying distributed systems," *SIGPLAN Not.*, vol. 50, no. 6, pp. 357–368, Jun. 2015.

[17] N. Giannarakis, R. Beckett, R. Mahajan, and D. Walker, "Efficient verification of network fault tolerance via counterexample-guided refinement," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 305–323.

[18] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 113–126.

[19] O. Rysavy, J. Rab, and M. Sveda, "Improving security in scada systems through firewall policy analysis," in *2013 Federated Conference on Computer Science and Information Systems*, Sep. 2013, pp. 1435–1440.

[20] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE industrial electronics magazine*, vol. 11, no. 1, pp. 17–27, 2017.