A Tool for Generating Automata of IEC60870-5-104 Implementations

Max Kerkers[1], Justyna J. Chromik[1],
Anne Remke[1,2], and Boudewijn R. Haverkort[1]

[1] University of Twente
m.kerkers@alumnus.utwente.nl,
{j.j.chromik, a.k.i.remke, b.r.h.m.haverkort}@utwente.nl
[2] University of Münster

Abstract. Power distribution networks are often controlled using the communication protocol IEC 60870-5-104 (IEC-104). While a specification exists, not every device implementing this protocol, actually follows this specification. We present *mealy104*, a tool that infers finite-state automata from IEC-104 implementations and use it on a real device implementing IEC-104, comparing it to the protocol standard. We use the tool to show that implementations do deviate from the specification.

Keywords: ICS · power grid · SCADA · Mealy machine · IEC-104

1 Introduction

Implementations of communication protocols should closely follow their specification, as differences or ambiguities might lead to security issues, as recently shown by the vulnerability that was found in the popular Wi-Fi protocol WPA2 [7]. Similar problems might occur with any protocol, if the specification contains ambiguity or if implementations do not follow the standard. Vulnerabilities in industrial control protocols like IEC-104 pose a serious threat to critical infrastructures, such as the power distribution grid. The implementations of industrial control protocols are often not checked against protocol specifications.

To verify whether an implementation follows a specification, both can be represented as finite state machine and then compared. The automaton representing the specification should be part of the standard. The other automaton can be learned from the implementation, e.g., using the tool presented in this paper. It implements a variant of Angluin's $L^\star$ algorithm [1], which produces Mealy machines that can represent more complex behaviour of input/output systems [5]. This algorithm has been applied before, e.g., for determining the correct operation of the ABN Amro e.dentifier2 [2], or implementations of the Transport Layer Security protocol [3]. To the best of our knowledge, we present the first tool to check communication protocols in *SCADA networks* that is made available under the Gnu General Public License. The source code of this tool can be found on GitHub[3].

[3] https://github.com/mkerkers/mealy104

We propose a tool developed for the automated generation of Mealy machines for implementations of IEC-104 [4], which is crucial for the communication between control and field stations in power distribution in Europe. The tool we developed generates a formal representation from an IEC-104 implementation. We tested three IEC-104 simulators and two real-life devices with our tool [4, Chapter 5]. While **none** of the simulators implemented the protocol according to its specification, the investigated hardware, i.e., Sprecher Sprecon-E-C-92 and Datawatt D05-Lite, only partially matched the specification.

This paper describes the most relevant information about the IEC-104 protocol in Section 2 and the tool setup and most significant components in Section 3. Finally, Section 4 presents a case study on a real-life IEC-104 implementation.

2 SCADA protocol IEC-104

IEC-104 [6] describes two different layers: the Application Protocol Control Information (APCI) layer and the Application Service Data Unit (ASDU) layer. The first runs on top of the TCP layer and has three message formats: (i) unnumbered control functions (U-format), (ii) numbered Supervisory functions (S-format), and (iii) the Information transfer format (I-format). **U-format** messages either (de)activate a connection using STARTDT (start data transfer) and STOPDT (stop data transfer) or test whether a connection is still active using TESTFR (test frame). **I-format** messages transfer data. They use TypeIDs to define what kind of message is sent, using, e.g., General Interrogation (C_IC_NA_1) or Single Command (C_SC_NA_1) numbers that range from 0 to 255. **S-format** messages acknowledge previously received I-format messages.

3 Description of *mealy104*: components and set-up

We first describe the main components of the tool before discussing its general setup. As shown in Figure 1, the learner builds the automaton based on input queries from the alphabet that are translated by the mapper to actual IEC-104 messages. The teacher, i.e., the queried device, answers these request. Once an automaton is constructed, the checker tests it against the protocol specification.

The **Alphabet** implements all IEC-104 message format types: three U-format types, the S-format type, and one for each I-format category. The complete alphabet is available in [4, Appendix A]. The **Learner** is built using the framework LearnLib[4]. For each run, a suitable sub-alphabet is chosen to create automata implementing the $L^\star_M$ algorithm [5]. The **Mapper** translates between abstract (human-readable) messages on the learner side and actual messages on the teacher side. For every abstract message in the alphabet, the mapper contains an implementation of a concrete message, structured according to the format as defined in the IEC-104 specification. To implement these concrete messages, OpenMUC j60870[5] is used.

[4] `http://learnlib.de`

[5] `https://www.openmuc.org/iec-60870-5-104/`

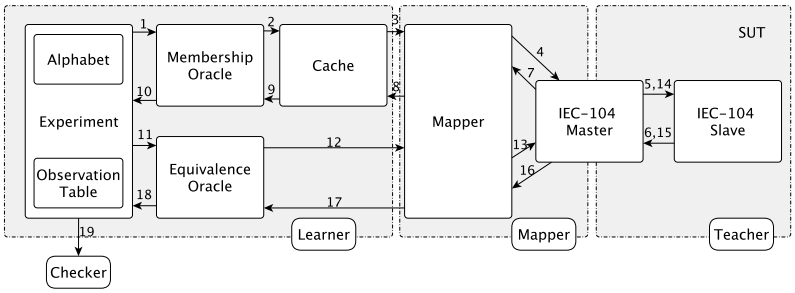

Fig. 1: Block diagram of the *Mealy104* finite-state automata learner

The **Teacher** is implemented as master using OpenMUC j60870 such that all fields in APDUs are adjustable, and sending and receiving of STOPDT and TESTFR messages is included. The **Checker** tests if the automaton of the implementation matches the automaton deduced from the standard [6], as shown in Figure 2; it has been built using AutomataLib[6], and traverses both automata using the same inputs, and comparing the outputs.

In more detail, the learner consists of: (i) the experiment, (ii) the membership oracle and cache, and (iii) the equivalence oracle. The experiment uses the membership oracle (Step 1 in Figure 1) to learn a hypothesis, structured as an Observation Table. During learning, the membership and equivalence oracles send abstract queries to the mapper (Steps 2, 3 and 12), from which they receive abstract answers (Steps 8, 9 and 17). A cache between the membership oracle and the mapper stores each query and its corresponding answer. The mapper translates each abstract query it receives (Steps 3 and 12), into a concrete IEC-104 message which is forwarded to the IEC-104 Master (Steps 4 and 13), and from there to the IEC-104 Client (Subject Under Test). The SUT replies with a concrete answer (Step 15 and 16), which the Mapper translates into an abstract answer and sends to the oracles (Steps 8 and 17). The membership oracle continues sending queries until the hypothesis is closed and consistent. Then, this learned hypothesis is returned (Step 10) and passed to the equivalence oracle (Step 11), which attempts to find a counterexample (Step 18). A counterexample is added to the Observation Table and the learning is restarted. Without a counterexample, the final hypothesis is transformed into a Mealy machine and passed to the checker (Step 19).

The equivalence oracle first checks for inconsistencies with the cache, then it sends random queries to the mapper, checking if the responses match the hypothesis. The tool is configured to send 1000 random queries to the SUT, and it resets the SUT and the hypothesis to the initial starting position with

[6] `https://github.com/LearnLib/automatalib`

probability 1%. These settings provide a traversal that is both extensive and time bound. If a random query contradicts the hypothesis, a counterexample has been found. If none is found after 1000 random queries, the hypothesis is assumed to be confirmed. To compare the hypothesis to the standard specification, the tool contains the standard automaton (cf. Fig. 2) [4, Section 3.5].

4 Case Study and Outlook

We tested several simulators of the IEC-104 protocol and two real devices used in the Dutch power distribution system [4, Chapter 5]. The Axon Test Simulator and the Siemens IEC-Test Simulator generated only one state, where all inputs were accepted; the Mitra Software IEC 870-5-104 Simulator generated two states: one required to initiate the connection by sending U[STARTDT] message, and then it accepted any input. Furthermore, the Sprecher Sprecon-E-C-92 did not match the specification. For example in the UNCONFIRMED STOPPED state, according to the specification the connection should be terminated upon receiving an I-format message. Instead, the device keeps accepting incoming I-format messages.

In the following, we present one result of a DataWatt D05-Lite device used as an IEC-104 RTU in a field station. We run the tool for different subsets of the alphabet. For most of the cases, the obtained Mealy machine matches the one provided by the standard. However, one automaton was learned that does not comply to the standard when sending the I-format message for File Select. Figure 2 shows the automaton from the standard, whereas Figure 3 shows the learned automaton. We used the same name and color for corresponding states

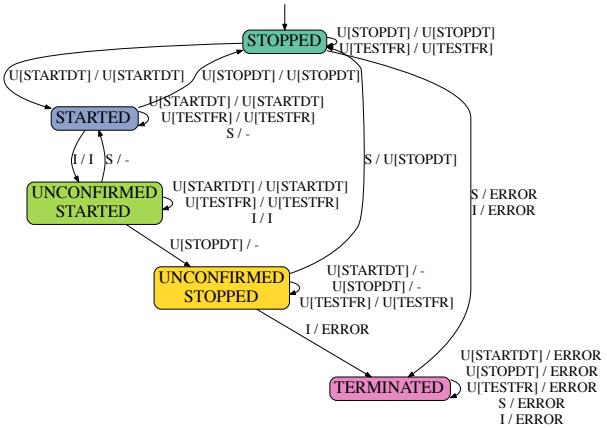

Fig. 2: Automaton derived from the IEC-104 standard for any I-frame

in both automata. While they largely overlap, an additional state, indicated by 'X' (colored orange) can be observed in Figure 3. As the File Select message does

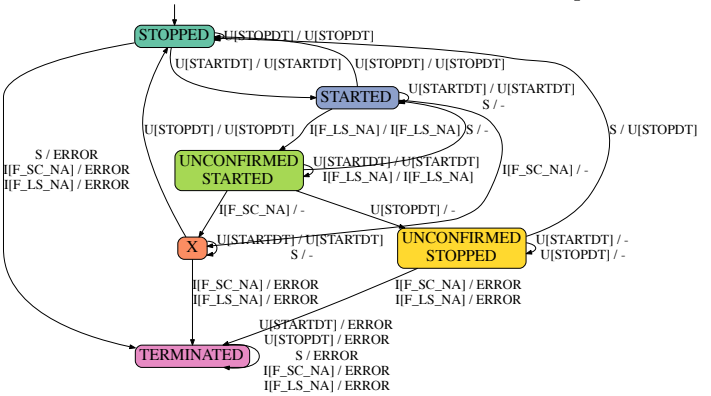

Fig. 3: Automaton learned using alphabet containing File Select I-frame.

not contain a valid address, the RTU is stricter than the standard. It terminates the connection on the next received I-format message, while the standard expects a negative confirmation I-format message. Hence, the behavior specified by the standard is *not fully implemented* by the investigated device.

Outlook. This tool can be used by vendors or users of devices implementing the IEC-104 protocol. The variety in the implementations of the IEC-104 protocol is alarming for both parties. The deviation in the presented implementation was found in a rarely used File Select function. However, such rare scenarios are often exploited [7]. Note that the presented tool can be adjusted to other protocols by adapting its components.

References

1. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
2. G. Chalupar, S. Peherstorfer, E. Poll, and J. de Ruiter. Automated Reverse Engineering using Lego. *Proceedings of the 8th USENIX Workshop on Offensive Technologies*, page 9, 2014.
3. J. de Ruiter and E. Poll. Protocol state fuzzing of tls implementations. In *USENIX Security Symposium*, pages 193–206, 2015.
4. M. Kerkers. Assessing the security of IEC 60870-5-104 implementations using automata learning. Technical report, May 2017.
5. M. Shahbaz and R. Groz. Inferring Mealy Machines. In *Lecture Notes in Computer Science*, volume 5850, pages 207–222. Springer Berlin Heidelberg, 2009.
6. TC 57 - Power systems management and associated information exchange. IEC 60870-5-104:2006. Technical report, International Electrotechnical Commission, Geneva, 2006.
7. M. Vanhoef and F. Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2, 2017.