Development of a Simulator for Testing Software Systems Working with a Binary Protocol †

Iliyan Boychev * and Gergana Spasova ‡

Faculty of Computer Sciences and Automation, Technical University of Varna, Studentska 1, 9000 Varna, Bulgaria; g.spasova@tu-varna.bg
* Correspondence: i.boychev@tu-varna.bg
† Presented at the International Conference on Electronics, Engineering Physics and Earth Science (EEPES'24), Kavala, Greece, 19–21 June 2024.

Abstract: In this article, the development of a software system—a simulator for testing software systems using a binary protocol—is proposed. The purpose of the development is to achieve full testing of a wide class of software systems. They are directly related to the management of hardware systems, with which communication is based on a binary protocol. The developed simulator allows different types of messages to be simulated during real communication. In this way, it is possible to check how a given program system would react to correct and incorrect messages in the communication stage. This article presents a GUI system developed in the standard C++ programming language and the Qt platform, which makes it platform-independent. The system provides the ability to select preset commands via an external file or enter/correct commands in real-time. The commands may be different depending on the integration scope of the binary protocol. The connection between the developed simulator and the tested system is through RS 232 interface. Configuring settings and testing binary protocols through the developed application becomes fast and intuitive.

Keywords: binary protocol; communication protocols; program systems; simulator; simulation

1. Introduction

This paper [1] proposes an approach to represent a binary protocol in a state machine, which enables automatic generation of state patterns for binary protocol by listening to the network. A methodology for field alignment in the binary protocol, their extraction, and construction of the protocol model are implemented. The experimental results of ARP and TCP show that the approach is effective.

Binary protocols do not have session flow characteristics. To achieve binary protocol data identification, an unsupervised clustering algorithm based on enhanced principal component analysis (PCA) and density peak clustering (DPC) is proposed. The authors of [2] propose to improve PCA by determining the dimensionality of PCA based on information acquisition. Enhanced PCA can remove redundant information and preserve the features of the original data.

The authors of [3] propose a Biprominer tool that can automatically extract binary message protocol formats of an application from its real-world network trace. The main feature of Biprominer is that it does not need to have any prior knowledge of protocol formats, as Biprominer is based on the statistical nature of the protocol format.

The authors of [4] propose a Veritas system that automatically infers the protocol state machine from real-world network traces. The main feature of Veritas is that no prior knowledge of the protocol specifications is required, and the work is based on the statistical analysis of the protocol formats. The authors define a new model—a probabilistic protocol state machine (P-PSM).
The authors of [5] introduce the RecordFlux toolkit and language. The RecordFlux Domain Specific Language (DSL) is used to specify the protocol version and verify the correctness of the generated code using the SPARK toolkit.

The authors of [6] introduce a method based on the symbolic execution of multiple packets, which can bring the software to deep states to test the entire binary software stack of the network protocol. This paper also presents a prototype system, S2EProtocol-multi [7,8], on the Selective Symbolic Execution (S2E) platform.

In [9], an approach for detecting anomalies and vulnerabilities based on a dynamic binary analysis technique and a semantic-based active testing technique is proposed. The proposed implementation works on the binaries of tested protocol implementations and can detect deviations.

In [10], an automated field semantics inference method for a binary protocol (FSIBP) is proposed. FSIBP aims to automatically learn semantic inference knowledge from known protocols and use it to infer the semantics of each field of an unknown binary protocol. To achieve this goal, a method for extracting the characteristics of the field itself is designed.

A multiple data sequence alignment algorithm [10] based on approximate matching (MSAAM) is implemented to process the data with random error bits. After sequence alignment, a noise-resilient system for extracting protocol formats from binary data is created, combining statistical analysis and a feature library.

2. Purpose of the Development

The program systems using a binary protocol represent systems operating in real time. They send and receive messages mostly via the RS232 interface. Each system sends and receives messages with different content, depending on the application. For development testing, two approaches are used:

- Availability of the physical system to be serviced. It sends and receives real data to the programming system,
- Development of a simulator (software or hardware) that simulates the physical system. It receives and sends real or test data with which operability is verified. The data can be correct or incorrect according to the communication protocol used to fully test the functionality of the application program.

The first approach has the disadvantage that the physical system is not available every time in advance or cannot be used in the application program development process. There are also cases where the two systems are in parallel development and cannot be used together until they are fully completed.

The second approach requires developing a system that can only be used to test one or a few such developments, but not a broad class of applications. This leads to increased costs of time and resources. Moreover, such systems will not be implemented and operated because are intended only for hardware system simulation.

In this article, it is proposed to develop a program simulator that implements a binary protocol and can receive/send messages by connecting via an RS232 interface to the corresponding program system. This simulator is a universal programming system, in terms of messages, because they are performed depending on the respective tested software system and its application.

The software systems check messages for correctness at two levels (Figure 1).

![Figure 1. Message-checking levels.](image-url)
At the physical level, a message is received as a sequence of bytes. It then checks that a correct message has been received (the correct number of bytes and CRC code, which is calculated when the message is received and compared to the one that was sent).

At the application level, it is checked whether the message corresponds to the expected data to be received (for example, a message should be received from receiver 02 and a message is received from receiver 04, or a status message was sent and another message was received, etc.).

After passing the double-check, the corresponding program system proceeds to the subsequent processing of the received data.

During the actual operation, it is possible that unforeseen situations occur and messages are not sent correctly between the physical and the software system (for example, due to cable interference, unplugged cable, incorrect message formation, etc.).

The developed simulator and the tested software systems are based on a binary communication protocol (Table 1).

### Table 1. Message protocol.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6…n−2</th>
<th>n−1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment Receiver</td>
<td>Address Receiver</td>
<td>Segment Sender</td>
<td>Address Sender</td>
<td>Length (from 5 to n−1 byte)</td>
<td>Command</td>
<td>Parameters (Optionally)</td>
<td>Check sum</td>
</tr>
</tbody>
</table>

The message contains the following fields:

- Segment Receiver (byte 0)—segment of the recipient—a one-byte field that contains the segment of the device that will receive the data.
- Address Receiver (byte 1)—address of the recipient—a one-byte field that contains the address of the device that will receive the data.
- Segment Sender (byte 2)—address of the sender—a one-byte field that contains the segment of the device that will send the data.
- Address Sender (byte 3)—address of the sender—a one-byte field that contains the address of the device that will send the data.
- Length (byte 4)—length—the length from the next byte to the end of the message is set.
- Command (byte 5)—command—a one-byte field in which a command number is set.
- Parameters—command parameters. These can be from 0 to 249 bytes, which means it can have no parameters, only one parameter, or multiple parameters, with a maximum of 249, since the maximum message length is 256 bytes.
- Checksum—this field is calculated by the sender and the receiver. The calculation is as follows:
  - On the sender’s side—the sum of all bytes from 0 to n−1 bytes is found, and then this difference is subtracted from 255 and the result is recorded in the CRC field.
  - On the recipient’s side—the sum of all bytes from 0 to the n−1st byte is found. The checksum is then added, and if the result is 255, the message is accepted as true.

The maximum length of the message is 256 bytes, and the minimum is 7 bytes.

According to the developed binary protocol (Figure 2), messages are divided into several types:

- A correct message that has all fields and a valid CRC code.
- A bad message that has all fields and the wrong CRC code.
- Short message with missing fields.
- Long message—a message that consists of more fields. In this case, the simulator finds one full message (correct or incorrect) and one more short message:
If the extra fields are after the CRC and it is correct, the message is correct; if it is wrong, it is incorrect.

If the extra fields are before the CRC, then there is a correct message with the wrong CRC code.

A developed software system must intercept both correct messages and messages that do not correspond to the correct ones according to the communication protocol. As a result of incorrect messages, they should not be processed at the application level because these will contain incorrect and/or partial data. This can lead to unforeseen situations.

The purpose of the simulator is to send different types of test messages to simulate different problem situations and the correct operation of the software system under development. In this way, the behavior of a system can be studied in the presence of correct and incorrect messages on the communication channel.

3. Program Simulator

3.1. Overview

In Figure 2, the simulator home screen is shown. It consists of:

- Fields for visualizing received and sent messages.
- A field for manual input/correction of messages.
- An ability to load commands from an external file.
- An ability to automatically format a message, when entering only the fields with the data (this allows for the formation of large messages and automatic calculation of the data length and CRC code).
- Options to configure the COM port according to the requirements of the specific application.

3.2. State Machine of Simulator

Figure 3 shows the state machine that receives the messages at the physical level. It aims to accept the message from the sender and check it for correctness against the byte-level protocol. Each field of the binary protocol is a state of the machine. When a message is received, there is a set time for which; if the rest of the message is not received, a timeout is generated with an error stating that there is a short message.
This causes the state machine to restart and protects it from hanging in any of the states. If the state machine goes through all the states and finds that the CRC code is correct, it passes the message on to the application layer for further processing.

4. Experimental Setup

The experimental setup consists of the following components.

- The developed simulator (Figure 2). The simulator is developed in the Qt programming system using C++ program language. Thus, it can be compiled for different operating systems.
- Developed software process control system (the developed server application for machine liquid dosing system is used for testing). This application sends and receives messages to/from liquid dispensing machines, processes the received messages at the physical and application level, and records the data in a database. From this database, another application designed for end users shows the status of the machines, allows for the operation of the machines, etc.
- A computer on which both applications are installed to perform the test experiments.
- Two USB–RS232 converters, with which COM ports are simulated for RS-232 interface connection. Each of the applications works with a unique COM port because it resides on one physical machine.
- Cable for two-way communication, with DB9 connectors. To perform the tests, it is sufficient to connect only the Tx and Rx signals of the two couplers for two-way communication. When performing the tests, the settings of the COM ports must be the same.

5. Experimental Results

To run the tests, one of the liquid dosing server application command sets shown in Table 2 is used. That command is a command for machine status.

<table>
<thead>
<tr>
<th>Segment</th>
<th>Address</th>
<th>Segment</th>
<th>Address</th>
<th>Length</th>
<th>Command</th>
<th>Flags</th>
<th>Bottles</th>
<th>Info 1</th>
<th>Info 2</th>
<th>Check sum</th>
</tr>
</thead>
</table>

5.1. Sending the Correct Message Machine of the Simulator

In this test, a correct machine status message is sent from the simulator to the server application (Figure 4).

Result: The server application receives the message correctly (Figure 5 and updates the status of the current machine to active (Figure 6). When sending a status message, the server application only checks that the message was sent correctly. In this case, sending a status message to Machine 1 is simulated. Upon receiving a correct message, Machine 1 is marked as active, i.e., it works and can receive and send messages over the communication network.
5.2. Sending the Message with a Bad CRC Code

In this test, a message with a bad CRC code is sent from the simulator to the server application (Figure 7).

Result: The server application receives the message correctly, calculates the correct CRC code, and prints an error message for the bad CRC code in the corresponding message field (Figure 8).

5.3. Sending the Short Message

In this test, a short machine status message is sent from the simulator to the server application (Figure 9).

Figure 4. The simulator command format for a correct message.

Figure 5. Server console application response.

Figure 6. Server application status for one machine.

Figure 7. Simulator command format with a bad CRC.

Figure 8. Server console application response.
5.3. Sending the Short Message

In this test, a short machine status message is sent from the simulator to the server console application (Figure 9). The test experiments conducted and the obtained results (part of which are shown in this article) show the correctness of the reception of the messages on the communication channel. The simulator developed offers quick and easy configuration of test messages to simulate different communication scenarios in a real environment.

Result: The server application receives the message, but the state machine generates the timeout error and prints an error message for byte timeout (Figure 10). When receiving a new byte that is part of the message, a timer is started to wait for the next new byte. If the timeout expires, a short message is considered received and a timeout error is generated.

6. Conclusions

The test experiments conducted and the obtained results (part of which are shown in this article) show the correctness of the reception of the messages on the communication channel. The simulator developed offers quick and easy configuration of test messages to simulate different communication scenarios in a real environment.

For future work, the system can be expanded with the following functionalities: adding different communication protocols, communication through different protocols (for example, via Ethernet), automatic generation of reports with diagnosed errors, and automatic generation of possible messages for a given system.

Author Contributions: Conceptualization, G.S. and I.B.; methodology, G.S. and I.B.; software, I.B.; validation, G.S. and I.B.; formal analysis, G.S.; investigation, I.B.; resources, I.B.; data curation, G.S.; writing—original draft preparation, G.S. and I.B.; writing—review and editing, G.S. and I.B.; visualization, I.B.; supervision, I.B.; project administration, I.B.; funding acquisition, G.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

References


**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.