Lowcaf: A Low-Code Protocol Analysis Framework

1st Alexander Frank
University of the Bundeswehr Munich
Research Institute CODE
Neubiberg, Germany
alexander.frank@unibw.de

2nd Michael Steinke
University of the Bundeswehr Munich
Research Institute CODE
Neubiberg, Germany
michael.steinke@unibw.de

3rd Wolfgang Hommel
University of the Bundeswehr Munich
Research Institute CODE
Neubiberg, Germany
wolfgang.hommel@unibw.de

***Abstract*—Evaluating a communication protocol's design in terms of compliance and security is a very complex and time-consuming task. It requires the configuration of suitable test environments and test scenarios as well as the evaluation of procedures and handling of network traffic. A plethora of tools and frameworks exist that tackle individual problems of low-level networking but they usually presume robust programming skills and lack visualization. In contrast we propose an approach, derived from node-based processing systems, that conveys a clear logical picture that is still flexible and extensible enough to be adaptable to real-time processing or simulation of arbitrary networking components. In this paper, we present the concept for a visually programmable low-code communication protocol analysis framework (Lowcaf) that can be used either stand-alone to simulate network components or be used together with arbitrary backends. We also showcase our prototypical implementation of this framework and how it can be combined with the popular deterministic network simulator ns-3.**

***Index Terms*—network analysis, protocol analysis, low-code, scapy, ns-3**

I. Problem Description

Secure and reliable operation of communication protocols is crucial for a holistic security approach in information processing. Minimum security requirements for information systems are among others described by the NIST in the Federal Information Processing Standards 200 [1], aiming at assuring of information's confidentiality, integrity and availability.

Ensuring these properties during the life of a protocol – from its design and implementation through enhancements and updates – is challenging and many flaws are publicly exposured like Heartbleed, FREAK, or KRACK [2]–[5]. It is reasonable to assume that the situation for more use case specific protocols is no different, just that the process of adapting and fixing is typically not public information.

As a result continuous protocol testing and evaluation by developers or the security community for public protocols is necessary. Testing relies on network simulation, traffic generation, inspection or manipulation [6]. This functionality is split across a plethora of specialized tools, e.g., wireshark for packet sniffing, nmap for system and service probing, ns-3 for discrete network simulation. The entry barrier for the tools varies greatly, ranging from learning a few command line options to complex APIs and GUIs. Additionally, there are cases where a combination of multiple tools achieves desired results more effectively, e.g., a network simulator manages simulation but traffic visualization and packet processing is realized through other tools. However, the integration of tools is usually use case specific and done manually.

Especially for protocol analysis or prototyping, where tasks are to be performed through various tools based on network traffic provided by a network backend, we recommend a new solution without the intent to replace any of the existing tools. Instead we argue the need for a framework that enables joint usage of different tools in a modularized fashion atop a network backend such as to integrate packet processing and analysis functions provided either natively or through external tools in an accessible and reusable way. We believe to fulfill this idea and to leverage more visual abstraction and simplicity in protocol testing by introducing a graphical low-code protocol analysis framework (Lowcaf). We propose a low-code based solution especially inspired by GUI-driven network simulations like OPNet, OMNet++ or GNS3 (cf. [7]). In Lowcaf, we want to leverage a similar approach for communication protocol analysis and processing with the following key features: *a)* Accessibility for users with less coding experience, *b)* fast reconfigurability and natural code reuse, *c)* usability along with arbitrary network backends.

Lowcaf is loosely coupled to networking backends via a reusable application. Lowcaf and an exemplary connector to the ns-3 simulator are public available on GitHub[1].

In Section II, we describe requirements for a low-code protocol analysis and testing approach. Section III summarizes similar already existing approaches and evaluates their suitability. Section IV describes the concepts of Lowcaf, followed by selected implementation details in Section V. In Section VI we show an exemplary use of Lowcaf, discuss it and point out current limitations. Section VII summarizes our key findings and gives an overview of future work in this area.

This work has been funded by Airbus Defence and Space as part of the Airborne Cybersecurity Enhancement project and by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr – as part of the project DEFINE. dtec.bw is funded by the European Union – NextGenerationEU. Any opinions, findings, and conclusions or recommendations are those of the authors and not necessarily of the funding institutions. We thank our colleague Mario Silaci for providing the LoRa data set for our evaluation.

[1]https://github.com/ubwafr/lowcaf

978-3-903176-66-9 ©2024 IFIP

II. Requirements

With the Lowcaf framework we want to leverage traffic generation, inspection and manipulation (contentual, temporal or quantity aspects). The sections describe general requirements for a suitable tooling approach just as specific requirements for the aforementioned network traffic analysis tasks. Each requirement is prioritized in conformity with RFC 2119 [8].

A. General and User Interface Requirements

Following requirements are based upon the low-code platform design's dynamic perspective and functional perspective defined in [9]. The functional perspective demands for suitable domain-specific functions. An approach MUST provide the ability to break down testing scenarios in visual functional protocol analysis blocks (RG1). These blocks SHOULD be reusable (RG2). They MUST be linkable as processing pipelines (RG3) to provide process modeling. Visual elements SHOULD be programmatically extensible with arbitrary networking backends like simulators (RG4) or bare-metal networks (RG5). Storing and loading processing designs SHOULD be given (RG6). The ability to integrate existing network traffic analysis tools SHOULD be given.

B. Traffic Generation

Traffic generation is crucial for protocol analysis and verification [10]. A low-code protocol analysis framework MUST provide traffic generation capabilities (RTG1). Traffic may be produced via replaying prerecorded PCAP files (RTG2) or by injecting real-time traffic from already set up services (RTG3).

C. Traffic Inspection

Traffic inspection is mandatory for protocol security testing; it is provided by several existing tools. For one to understand traffic, it MUST be able to be visualizable (RTI1) (cf. [11], [12]) and SHOULD be able to be filtered via pattern matching (RTI2). Traffic MUST be able to be recorded and exported (RTI3) then again, for instance to provoke incorrect content handling via traffic injection.

D. Traffic Manipulation

Traffic manipulation helps ensuring that protocol procedures are well designed. We consider five traffic manipulations: Packet duplication (RTM1), packet filtering (RTM2), and field manipulation (RTM3). Besides data centric manipulations, timing relations are interesting: Delaying (RTM4) and reordering packets (RTM5) is necessary. Required functionality is dependent from each protocol analysis scenario. Hence, any function SHOULD be provided. More detailed traffic manipulation characteristics were elaborated in [13].

III. Related Work

To the best of our knowledge no concept for a node-based framework to realize network protocol evaluation exists. However, low-code is a highly active research topic with several tools and a similar logic in different application areas.

A. Related Scientific Research

Lowcaf aims at enabling a low-code (following the definition of [14]) for communication traffic processing. Low-code is used in different areas, e. g., Internet of Things (IoT) [15]–[17], robotics [18], and agriculture [19] for development, design, and testing. For IoT [15], [16] low-code approaches for tasks such as application development, architectural modeling, integration of IoT devices into larger IoT platforms and many more are actively researched. Other work, as presented in [20] investigates what benefits and challenges arise from using low-code approaches for testing purposes in the context of low-code development platforms.

B. Existing Node-based Processing Tools

Low-code (especially node-based programming) is popular for complexity reduction of processing tasks, e.g., in the areas of 3D rendering (e.g., in Blender[2]) and signal processing, which is closely related to our approach. The purpose of nodes varies between use cases. For example considering network simulators (besides those already named in Section I), a node may be a simulation of one specific device or for a category of devices. In general, a node encapsulates specific functionality, which leverages natural code reuse. For our concept we extend this node-based network method with the ability to model the processing behavior of network equipment via a node-based system. This is shown in upper half of Figure 1, the Packet Processing Framework (PPF).

Node-based processing systems are also used for radio frequency (RF) signal-processing. Two popular frameworks in this area are MATLAB/Simulink[3] and GNURadio (GR)[4]. Both feature node editors that allow to create signal processing chains to modify signals (e.g., modulate, interpolate, rasterize, etc.), visualize and analyze them. Such a chain has typically one logical starting point and one logical end.

As existing systems work with streams of different data types (e. g. floats, integers, bits), they are not suitable for packet-based communication on top of. While GR and similarly Simulink offer a low-code design meeting requirements RG1 to RG6, they show restrictions in compatibility with existing protocol analysis tools (RG7). They theoretically support traffic generation, inspection and manipulation but no dedicated nodes are currently implemented.

IV. Low-code protocol analysis framework Concepts

The basic idea is shown in Figure 1 with two major parts: The Packet Processing Framework (PPF) and the Networking Backend (NB). Since the term *node* is used to refer to both nodes in node editors and systems in a network, we differentiate between both by designating the former Processing Nodes (PNodes) and the latter Networking Nodes (NNodes). The PPF is our proposed low-code-based approach for network

[2]docs.blender.org/manual/en/latest/modeling/geometry_nodes/index.html
[3]de.mathworks.com/products/simulink.html
[4]www.gnuradio.org/

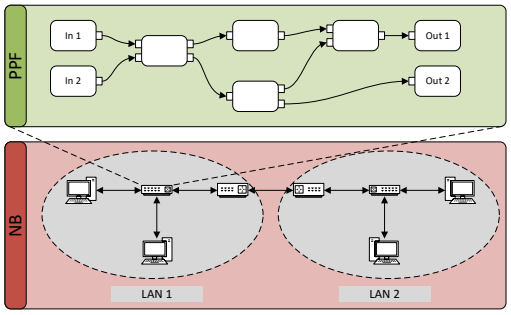

Fig. 1. Modeling the packet processing of single components with Lowcaf.

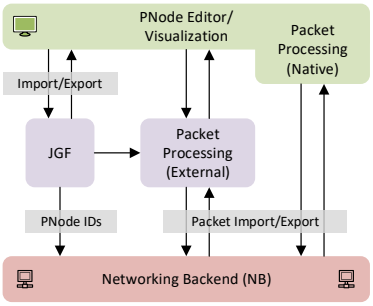

Fig. 2. Relation between the software components.

traffic analysis processes with chainable PNodes. The NB is an arbitrary networking environment. The processing logics defined in PPF can be understood as a processing pipeline residing in a NNode of the NB.

A detailed view of Lowcaf is shown in Figure 2. The PPF consists of the PNode editor and graphical frontend to visually edit processing chains and to visualize their state. The Packet Processor (PProc) is responsible for packet processing. The PProc can either be native (part of the editor and visual components) or external. The structure and state of the processing chain are stored in JSON Graph Format (JGF) files (see Section IV-A2). Its contents can be disseminated to external PProc and the NB. The NB may be connected to the PPF: The primary connection is towards the packet processing component (on the right). Alternatively, the connection can be passed through external packet processing tools (middle interface). Both interfaces are bidirectional. For the NB to properly communicate with the PProc, the IDs of corresponding PNodes are required from the JGF file. The third connection is an optional downstream connection from the PPF towards the NB via JGF. If supported by the NB, network topologies can be loaded from the JGF-based configuration.

A. Packet Processing Framework (PPF)

Packets are exchanged between PNodes. In Lowcaf a packet is a data structure that contains the corresponding data buffer and metadata like timestamps of the original reception. The overall architecture of the Lowcaf framework and its main components is shown in Figure 3. The Lowcaf simulation runs independently of an underlying network backend. The PProc schedules the PNodes for execution and for forwarding packets

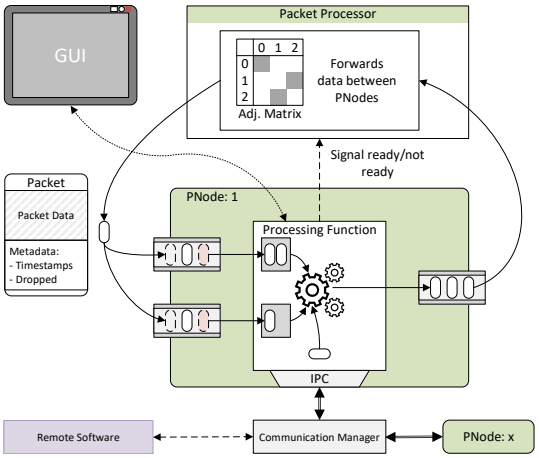

Fig. 3. Relation between the simulation driver and a PNode.

from outputs to the corresponding inputs. The components performing the actual processing are the PNodes. Each PNode can have an arbitrary number of inputs and outputs, a processing function with user-defined code. Running the processing function may require any number of packets from one or more inputs. A PNode signals to PProc as soon as it has enough data available to run at least once.

In addition to processing packets from one of the inputs a PNode can also register an Interprocess Communication (IPC) connection. They can be used to connect to any other system supporting our data exchange format (cf. Section IV-C) via system sockets. External programs can also be attached via shared PCAP files or virtual TUN or TAP interfaces. Each PNode is also connected to a graphical representation of itself. The graphical representation can be used to configure settings of the PNode or to visualize its state or statistics.

1) Packet Processor: We considered two packet processing approaches: First, running each PNode's processing function independently from each other in a loop, where data will be processed as soon as it is available. Second, schedule the execution of each PNode from a coordinating entity. The first approach can be massively parallelized and is thus more suitable for real time network processing. In contrast, the second approach is more suitable for network simulation as it can ensure deterministic processing and repeatability. As we want to leverage protocol understanding and reliable testing, we pursue the latter approach with PProc. It schedules the execution of the processing function of all PNodes. Users may modify the selection order according to their needs. In each step the following tasks are performed in order:

1) Get the next PNode from the selection algorithm. If there is no next PNode terminate.
2) Execute the PNode's function.
3) Move the generated outputs to the respective target PNode's port.
4) Jump to the first step.

As a simple example, we provide a priority selection algorithm that internally maintains a first in, first out (FIFO) list

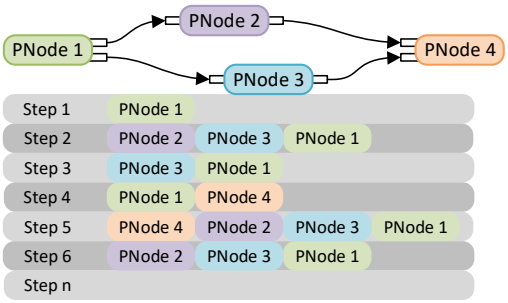

Fig. 4. Operation of the packet processor for a small example.

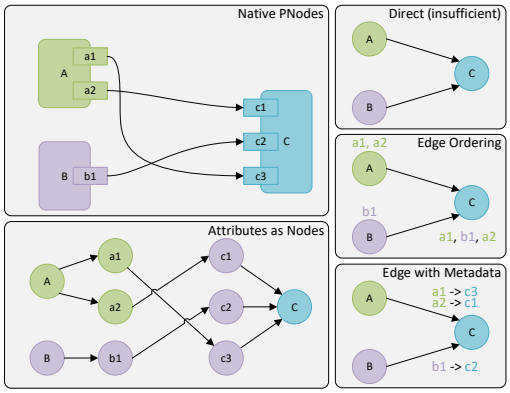

Fig. 5. Options for the representation of our PNodes as generic graph.

of all processing functions that are able to be executed at that moment in time. The list is initially filled with all PNodes that could directly be executed, ordered by each PNode's unique ID. If at some point no PNode is able to run its processing function anymore, the processing graph is exhausted. This does not mean that no more packets are contained in the processing network, since a processing function may depend on more than one input. An small example of this process is depicted in Figure 4. PNode 1 is always ready and for each execution it produces one packet for each output. PNode 2 and 3 each require one input and create one output and PNode 4 requires one packet in each input. Thus initially only PNode 1 is ready and produces two packets, which means that both PNode 2 and PNode 3 become available (in this order because of port priority), additionally PNode 1 is added to the queue.

2) PNodes as Multigraphs: The PPF represents all logical state in the aggregation of PNodes and their connections. We need to represent and save this state, such that it can be executed at a later point in time or a different system. At its core, PNodes and their connections can be viewed as a cyclic directed multigraph where two PNodes may have multiple edges between each other. This is exemplarily illustrated in Figure 5. In this case PNode A has two ports a1 and a2 which both connect to ports of PNode C. If we directly interpreted each PNode as a vertex (see *Direct* in Figure 5) we cannot recover the actual ports. To solve this issue we considered three alternatives: First, one may specify an edge ordering, i.e., store a list of connected edges for each vertex. Second, one could store the port information as part of metadata for each edge. The third option is to not only represent PNodes as vertices but also the ports. The third option is preferred as it can represent the basic relation between PNodes and their connections without having to resort to additional metadata which is stored outside the graph representation.

Having mapped the PPF structure to a graph representation we can make use of existing graph-exchange file formats for storing or transmitting our PNode architecture. An extensive overview of such formats is provided in [21]. We formulate three requirements towards

1) the ability to assign metadata to vertices to store the internal state of PNodes (*integral meta-data* in [21]).
2) a well known and portable format.
3) a public and completely documented specification.

Due to our design – representing ports as vertices – we do not necessarily require multigraph functionality, and also compression does not apply, as for our node networks to be easily understandable for humans are not likely to become extremely large. Under these constraints we chose JGF[5]. It has a public specification, allows to add metadata to vertices and edges, supports hypergraphs and is built on JSON for which parsers are widely available in most programming languages.

B. Networking Backend

The Packet Processing Framework is designed to be usable atop arbitrary networking backends complying with the backend model described in the following section. We envisage bare metal backend, emulation (virtual machines) and simulation backends (e. g. ns-3 simulator) or hybrid backends. The individual backends are connected to the PPF via an Lowcaf Application (LApp), that can be reused for homogeneous backends.

1) Networking Backend Model and Lowcaf Application: The components of the NB are illustrated in Figure 6. A qualified networking backend complies with a generic networking model. It provides a) Networking Nodes (NNodes) and b) communication links connecting two or more (e. g. radio channels) NNodes.

The *a)* NNodes provide a LApp in their networking context each. For bare metal or emulated backends, the LApp is implemented according to the specific operating system running on the machine. Simulation backends usually demand for an integrated approach (cf. Section V-B). The NNode buffers incoming packets and forwards them towards the Lowcaf framework. A NNode can have three different roles in the network. Source Networking Nodes (SoNNodes) only send network packets towards other NNodes. Intermediate Networking Nodes (ImNNodes) receive as well as send network packets from or to adjacent NNodes. Sink Networking Nodes (SiNNodes) only receive packets from adjacent NNodes. The *b)* communication links transmit network packets originating from one NNode to all other connected NNodes. The communications links are not directly affected by the PPF.

[5]https://jsongraphformat.info/

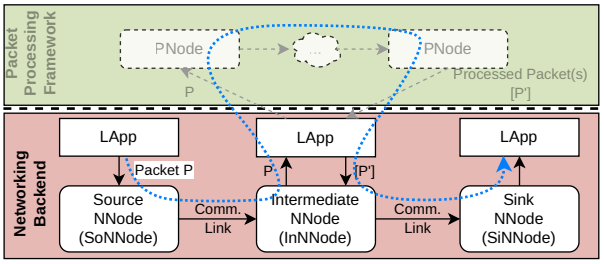

Fig. 6. NB components and interrelations, including an exemplary communication path of packets (blue color).

Figure 6 shows the interplay of the NB and the PPF by an exemplary packet P (blue path): P is infused to a SoNNode from the PPF and is then forwarded via the first communication link to an ImNNode. The ImNNode holds P, forwards it to the PPF via its LApp and waits for the response, which may contain processed packets [P'] of arbitrary count. The ImNNode then forwards every packet in [P'] towards the next node, in this case a SiNNode. This one then passes all packets towards the PPF and does not wait for a response.

The LApp may run on each NNode and connects them to one PNode. It transmits incoming network packets towards the PPF (only ImNNodes and SiNNodes). The LApp also receives packets from the PPF and initiates their transmission towards the next NNode (SoNNodes or ImNNodes). The LApp communicates with the PPF according to a common protocol, which is described in the next Section.

C. Communication

The communication between the LApps and the PPF (cf. Figure 6) provides transmitting received packets from the NB towards the PPF (*upstream communication*) and the other way around, i.e., transmitting processing results (*downstream communication*). Our model currently provides Ethernet-based communication but it is extensible to other protocols analogously.

1) Upstream Communication: The required packet data, needed by PPF is described in Table I upper half. The first field identifies the type of data (ToD). It has three functions, that are differentiated via the ToD's value: Transmit packet data to the PPF, terminate the processing and shut down the connection towards the PPF, or indicate that no further data is sent upstream. Packet data, marked with value *1* has further fields: The PPF is informed about the ID of the source NNode and the ID of the destination PNode. Each packet data also contains a delay or timestamp value, the PPF can access and modify. The *plen* field tells the length of the payload message that is transmitted as byte sequence afterwards.

2) Downstream Communication: The downstream communication has the same functions (ToD) as the upstream communication. The second and third command do no require further parameters. Packet data communications may be chained to provide the receiving LApp with a list of generated packets.

TABLE I
FIELDS FOR THE UP- AND DOWNSTREAM COMMUNICATION

Upstream Communication	
Field Name	**Comment**
Type of Data (ToD)	Packet data; End of Simulation; No Reply
Further fields for ToD = 1 (Packet Data)	
Origin NNode ID	NNode's ID the data is sent from in the NB
Destination PNode ID	ID of the PNode, the data is sent to
Delay/Timestamp	Delay or receipt timestamp at the source NNode
Packet length (plen)	Length of the packet data payload
Packet Data	Actual packet data of length *plen* from the NB
Downstream Communication	
Type of Data (ToD)	Packet data; End of Simulation; No Reply
Further fields for ToD = 1 (Packet Data)	
Delay/Timestamp	Delay or receipt timestamp at the source NNode
Protocol Type	Type of payload protocol frame
Packet Length (plen)	Length of the packet data payload
Packet Data	Packet data from the networking backend

V. IMPLEMENTATION

We implemented the PPF as well as a LApp for the ns-3 network simulator[6] as an exemplary NB due to its practical relevance in academia and research (see [22]).

A. Selection of GUI Framework

For the realization of the graphical frontend we considered using existing node editor frameworks. Our requirements are based on the typical expectations of GR or Simulink users:

- Multiple inputs/outputs: De-/Multiplexing is essential to realize components like switches or routers.
- User interaction: The framework should provide base blocks allowing to add, drag, and remove nodes as well as zooming and panning the current view.
- Arbitrary number of node inputs and data visualization.

Several suitable frameworks exist. For instance the commercial *Nodes*[7] tool or the open source *Node Editor Framework*[8]. *Rete.js*[9] is a JavaScript framework while *QtNodes*[10] builds upon the Qt framework [23]. We decided to built our solution upon *DearPyGui*[11] due to the flexibility of its nodes. It is a general purpose GPU-accelerated GUI toolkit for Python and based on the Dear ImGui framework. Each node in the editor behaves similar to a window and thus can contain nearly all widgets which are provided by the framework.

B. LApp for ns-3

The ns-3 network simulation framework is written in C++ and demands users to write their simulations accordingly.

1) LApp-integration into the ns-3-model: Just like in a *real* system, a NNode's functionality in ns-3 is encapsulated into one or more applications, that can be installed atop of it. We use this circumstance to implement a basic LApp, which can be installed on arbitrary nodes in the ns-3 model. Just like it

[6]https://www.nsnam.org
[7]https://nodes.io/
[8]https://nodeeditor.seneral.dev/
[9]https://github.com/retejs/rete
[10]https://github.com/paceholder/nodeeditor
[11]https://github.com/hoffstadt/DearPyGui

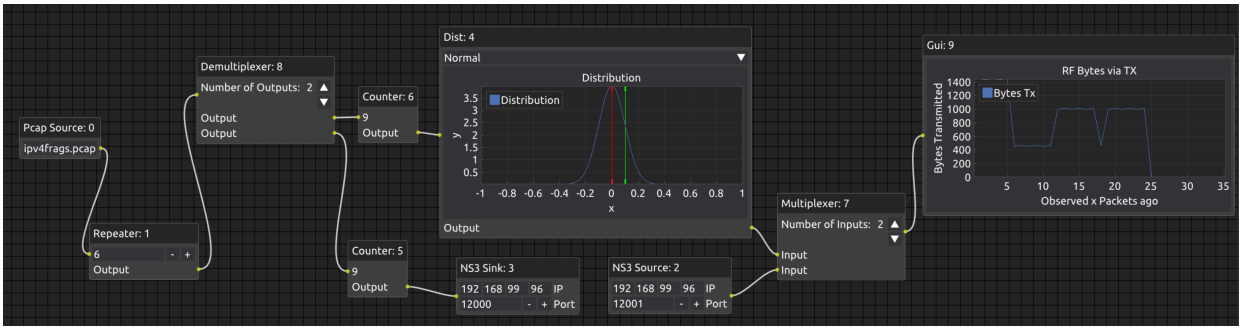

Fig. 7. Screenshot of the Lowcaf user interface.

is usual for applications in ns-3, one can set the point in time in the simulation, where the LApp is active.

2) Integration into the ns-3 process: The ns-3 simulator does not support multi-threading, preventing the LApp to pass packets from the PPF at any time in the simulation. The consequence is, that for ns-3 as a networking backend, the upstream and downstream communication with the PPF must be sequentialized and always works according to the same procedure depending on the role of a NNode: SoNNodes as sources of packets buffer packets until the simulation checks them. For that, we use the *select* function for checking input from selected sockets. ImNNodes only initiates the communication with the PPF in answer to a received packet from a preceding NNode. SiNNodes only send received packets from preceding NNodes towards the PPF and do not get responses.

C. Communication

We implemented the communication protocol from Section IV-C based on UDP and in a binary format. The field sizes are proposed in a sufficient magnitude: We described the ToD field with one byte, the delay timestamp with eight bytes allowing a nanoseconds precision and the protocol type field with two bytes. The NNode and PNode ID and *plen* are described with four bytes each.

VI. Evaluation and Limitations

We practically evaluated our approach in two use-cases which we describe along with our findings and limitations of our approach in the following sections.

A. Test Scenario 1: Basic Demonstration

We tested Lowcaf using ns-3 as a NB and Scapy [24] as a packet parser in the PPF. We modeled a realistic and real-time traffic analysis process in PPF (cf. shown in Figure 7, process from left to right). In the scenario, we read input from a PCAP file (*Pcap Source* node) and repeat its content via the *Repeater* node. We then split the packet stream in two paths: The upper path counts the packets and the *Dist* node manipulates their timestamp according to a given distribution. The lower path sends packets towards the ns-3 backend. In ns-3, we set up a simulated infrastructure with two NNodes that are connected via an Ethernet channel. The first SoNNode receives the packets from the *NS3 Sink* and the second SiNNode sends them on receipt towards the *NS3 Source* node in the PPF. Both streams are then joined in the *Multiplexer* node and the stream bandwidth is visualized in the *Gui* node.

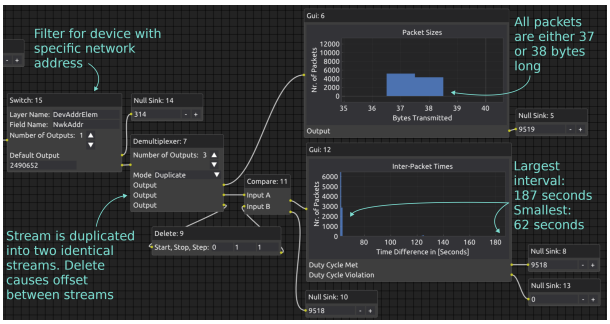

Fig. 8. Excerpt of Lowcaf graph for detection of LoRa duty cycle violations.

B. Test Scenario 2: Duty Cycle Violations in LoRa Networks

LoRa is a wireless technology used, i.a., for IoT applications. The available bandwidth must be shared and to ensure fair access for all participants, rules govern the acceptable usage of bandwidth per time frame per user. This is referred to as duty cycle and in this context a duty cycle of 1% is common [25]. We use Lowcaf to build an application that analyzes the duty cycle of a given LoRa device. The input data is a PCAP file covering one week's worth of LoRa traffic captured by a gateway. We first filter in multiple stages to a) operate only on packet types that we can attribute to participants (those with a network and a device address) b) extract the packets for the device of interest. By first duplicating the resulting packet stream and then removing the very first packet of one of the streams we yield two streams offset by one that serve as input for a comparison node. The difference between each pair's timestamps is computed and attached to the packets' metadata for further visualization and processing. Figure 8 shows the corresponding excerpt of the graph. Our (simplified) assumption is that a duty cycle violation occurs if $t_{diff}[i+1] < 99 \cdot t_{airtime}[i]$, with i as the packet index, t_{diff} as the inter packet gaps and $t_{airtime}$ as the time that each packet occupies the shared medium.

C. Discussion and Limitations

In the Lowcaf framework concepts, we considered the requirements stated in Section II. The general (RG1-RG6) requirements can be met and were shown (except programmatic aspects) in the scenarios. Traffic generation, inspection and manipulation, were shown via PCAP content replay (RTG2) and visualization, pattern matching and export (RTI1-RTI3). We showed packet delaying and re-ordering (RTM4, RTM5). RTG1 and RTG2 were not used in the scenarios; RTM1-RTM3 are not yet implemented, but can be realized analogously.

We also experienced some additional limitations, e.g., some usability and organizational aspects between the PProc and the NB are still unattended. Currently, one must map each PNode and NNode manually by a unique identifier and set service port numbers for communication purposes. While our current concept does not use 1-to-n connections (hypergraph), it would be a relatively minor change enabling transparent copies of packets. Furthermore, as already mentioned in Section IV-C, while the PPF is protocol agnostic, the interface to the NB currently only serves Ethernet-based communication. An extension to other communication standards is feasible but may require minor adaptions of our up- and downstream communication scheme. Our code is currently not optimized for performance, which is unproblematic for use in conjunction with discrete simulators such as ns-3 but could be relevant for real-time applications.

VII. Conclusion and Future Work

In this paper, we presented Lowcaf, a low-code framework for protocol analysis and testing. We argue, that by visually-aided analysis processing, protocols are easier to comprehend and that flaws can be found more reliably. It provides a set of built-in blocks that can be used out-of-the-box but also provides an interface for adding custom blocks for new tasks. Lowcaf can be set up for an arbitrary networking backend. Only a connector application for a backend is needed to communicate with the Packet Processor and can thus be reused for homogeneous network backends without further adaptions. Our solution allows storing and loading designed analysis scenarios by which protocol testing can be accelerated significantly. For the future we plan to approach the limitations from Section VI-C.

References

[1] NIST, "Minimum security requirements for federal information and information systems," Mar. 2006. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.200.pdf

[2] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, "The Matter of Heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. Vancouver BC Canada: ACM, Nov. 2014, pp. 475–488.

[3] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, "A messy state of the union: Taming the composite state machines of TLS," *Communications of the ACM*, vol. 60, no. 2, pp. 99–107, Jan. 2017.

[4] M. Vanhoef and F. Piessens, "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas Texas USA: ACM, Oct. 2017, pp. 1313–1328.

[5] ——, "Release the Kraken: New KRACKs in the 802.11 Standard," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 299–314.

[6] A. Frank, W. Hommel, and B. Hopfner, "An intermediary protocol representation to aid in avionics network development," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–5.

[7] C. Smera and J. Sandeep, "Networks simulation: Research based implementation using tools and approaches," in *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*. IEEE, 2022, pp. 1–7.

[8] S. Bradner, "Key words for use in rfcs to indicate requirement levels," Mar. 1997. [Online]. Available: https://www.ietf.org/rfc/rfc2119.txt

[9] A. C. Bock and U. Frank, "Low-code platform," *Business & Information Systems Engineering*, vol. 63, pp. 733–740, 2021.

[10] O. A. Adeleke, N. Bastin, and D. Gurkan, "Network traffic generation: A survey and methodology," *ACM Computing Surveys (CSUR)*, vol. 55, no. 2, pp. 1–23, 2022.

[11] S.-Y. Ji, B.-K. Jeong, and D. H. Jeong, "Evaluating visualization approaches to detect abnormal activities in network traffic data," *International Journal of Information Security*, vol. 20, no. 3, pp. 331–345, Jun. 2021.

[12] H. Shiravi, A. Shiravi, and A. A. Ghorbani, "A Survey of Visualization Systems for Network Security," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 8, pp. 1313–1329, Aug. 2012.

[13] M. Gadelrab, A. Abou El Kalam, and Y. Deswarte, "Manipulation of network traffic traces for security evaluation," in *2009 International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2009, pp. 1124–1129.

[14] M. Hirzel, "Low-Code Programming Models," *Communications of the ACM*, vol. 66, no. 10, pp. 76–85, Oct. 2023.

[15] F. Ihirwe, D. Di Ruscio, S. Mazzini, P. Pierini, and A. Pierantonio, "Low-code engineering for internet of things: A state of research," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1–8.

[16] S.-G. Pantelimon, T. Rogojanu, A. Braileanu, V.-D. Stanciu, and C. Dobre, "Towards a Seamless Integration of IoT Devices with IoT Platforms Using a Low-Code Approach," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Apr. 2019, pp. 566–571.

[17] K. Panayiotou, E. Tsardoulias, and A. L. Symeonidis, "Defining a Domain-Specific Language for the Verification of IoT-Enabled Cyber-Physical Application," Rochester, NY, May 2023.

[18] R. Brouzos, K. Panayiotou, E. Tsardoulias, and A. Symeonidis, "A Low-Code Approach for Connected Robots," *Journal of Intelligent & Robotic Systems*, vol. 108, no. 2, p. 28, Jun. 2023.

[19] G. Fatouros, G. Kousiouris, T. Lohier, G. Makridis, A. Polyviou, J. Soldatos, and D. Kyriazis, "Enhancing Smart Agriculture Scenarios with Low-code, Pattern-oriented functionalities for Cloud/Edge collaboration," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. Pafos, Cyprus: IEEE, Jun. 2023, pp. 285–292.

[20] F. Khorram, J.-M. Mottu, and G. Sunyé, "Challenges & opportunities in low-code testing," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. Virtual Event Canada: ACM, Oct. 2020, pp. 1–10.

[21] M. Roughan and J. Tuke, "Unravelling graph-exchange file formats," *arXiv preprint arXiv:1503.02781*, 2015.

[22] L. Campanile, M. Gribaudo, M. Iacono, F. Marulli, and M. Mastroianni, "Computer network simulation with ns-3: A systematic literature review," *Electronics*, vol. 9, no. 2, p. 272, 2020.

[23] D. P. et al, "Qtnodes. node editor," https://github.com/paceholder/nodeeditor, 2017.

[24] R. Rohith, M. Moharir, G. Shobha *et al.*, "Scapy-a powerful interactive packet manipulation program," in *2018 international conference on networking, embedded and wireless systems (ICNEWS)*. IEEE, 2018, pp. 1–5.

[25] EN. ETSI, "300 220-2 V3. 2.1 (2018-06), short range devices (SRD) operating in the frequency range 25 mhz to 1000 mhz; part 2: Harmonised standard for access to radio spectrum for non specific radio equipment."