Compositional Testing of S1AP protocol in Cellular Communication Networks

PRASANTH PRAHLADAN and TAEHO KIM, University of Colorado Boulder

The architecture and design of the cellular-communication-network is fast evolving to handle the needs of IoT and 5G communication. With the increase in scale, we need to update the different protocols used to control-coordinate decision making in the control-plane. While the protocols get updated, the implementations of the protocols need to be evaluated for compliance with older versions. The current methodology of testing doesn't help to address this challenge. The research community has been trying to adopt formal methods to specify and ensure correct implementations of protocols within the domain of software and internet-networks. There hasn't been any prior work that does this within the cellular domain. We explore one method of compositional testing of S1AP protocol, which is used in the control-plane of cellular communication infrastructure. Our contribution is to transfer the tools and techniques that are being adopted for internet-protocols to the domain of control-plane protocols within cellular communications with a particular focus on the interface between the radio part and the core part of the network.

Additional Key Words and Phrases: S1AP, EPC, IVy

ACM Reference Format:
Prasanth Prahladan and Taeho Kim. 2020. Compositional Testing of S1AP protocol in Cellular Communication Networks. CSCI 5535, Spring 2020 (May 2020), 13 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The 5G cellular network deployment, with the promise of enabling an internet-of-things, is an endeavour that has been the target of industrial players in the last few years. The promises of the 5G network architecture arise from the following enhancements over the 4G LTE architecture: (1) improvements in the physical and application layer technologies enabling a large number of devices with higher bandwidth provisions, (2) enhanced security features introduced into the protocol stack. The 5G standard proposes a complex architecture of subsystems that interact with each other at different layers of the communication stack while using multiple sub-protocols between entities.

The cellular network infrastructure may be viewed as large-scale wireless with a wired backend that is designed to support mobile data and voice services. Communication and messaging between the entities may be surmised under two layers of its design-abstraction called control-plane and data-plane. The control plane protocols provide complex signalling functions, which makes it quite different from the network protocols that enable the internet. They follow the layered protocol architecture and run at both the network infrastructure and the end device. The cellular network control-plane consists of a number of critical procedures that are leveraged by the primary cellular services like paging, voice-call, SMS, data and billing. Incorrect implementations of the protocols can have adverse consequences on these services.

Authors' address: Prasanth Prahladan, prasanth.prahladan@colorado.edu; Taeho Kim, taeho.kim@colorado.edu, University of Colorado Boulder, 1111 Engineering Drive, Boulder, Colorado, 80309.

Accordingly, related protocols are frequently updated, and new protocols frequently appear. *User equipment (UE)*, such as a smartphone or a tablet, connects to an *evolved universal terrestrial radio access network (E-UTRAN) node B (eNB)*, which performs radio resource management to UE, to use cellular communication networks. In addition, when the UE is connected to networks, an *mobility management entity (MME)* must be connected to the eNB to manage this connection, as shown in Figure 1. MME provides an evolved packet system (EPS) mobility management (EMM) and EPS session management (ESM) functions to the UE through *non-access stratum (NAS)* signalling. A NAS is a functional layer in the wireless telecom protocol stacks between the core network and UE. This layer is used to manage the establishment of communication sessions and for maintaining continuous communications with the UE as it moves.

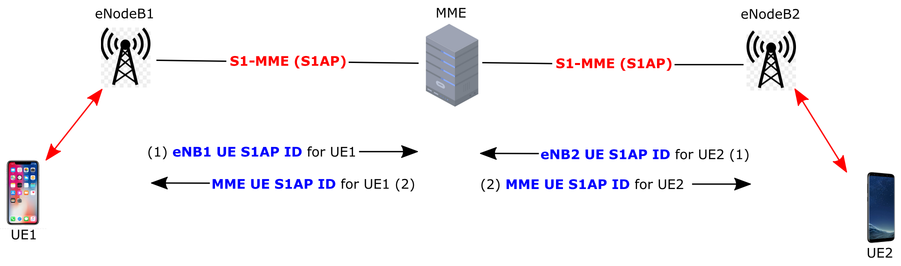

eNodeB1
MME
eNodeB2
S1-MME (S1AP)
S1-MME (S1AP)
(1) eNB1 UE S1AP ID for UE1
MME UE S1AP ID for UE1 (2)
eNB2 UE S1AP ID for UE2 (1)
(2) MME UE S1AP ID for UE2
UE1
UE2

Fig. 1. UE S1AP ID allocation

In this work, we examine the protocol-specific interactions between critical components over the control plane, with the intention of identifying problems/noncompliance of entities to the protocol specifications. Among them, we focus on the interface between the eNB and the Core-Network. To be precise, we intend to address the following concrete research question:

Is it possible to formally verify the correctness of a subset of the critical procedures handled by the control plane interactions between the evolved packet core (EPC) and the MME (cellular core network)?

There are two key challenges to this endeavour - (1) compared to the internet, cellular networks are still closed systems, in other words, signalling exchanges between the entities are not readily accessible during normal operations, (2) patterns of inter-protocol communication over the control-plane are more diverse and complex, in comparison to their counterparts over the internet. Protocol interactions are visible not just at the inter-layer interfaces, as seen over the internet, but also in cross-domain (e.g., data and carrier-grade voice services require signalling for circuit-switching and packet-switching in these networks) and cross-system scenarios (due to heterogeneous deployment strategies inter-system switching between 3G, 4G and 5G systems need to be facilitated).

Instead of translating the 3GPP protocol specification for the *S1-Application Protocol (S1AP)* and the NAS protocol, we intend to demonstrate the methodology for specifying on-the-wire messages that are exchanged between the entities while interacting over a stateful-protocol. We shall focus on implementing one of the primary procedures facilitated by the EPC, the UE-Attach Procedure. In this procedure, the EPC facilities the connection of a UE to the cellular network to permit data transmission over IP.

We intend to extend the application of the formal verification of Internet-protocols [5] to the domain of cellular communication networks. We are using the IVy tool [4] for providing a formal-specification of the S1AP/NAS protocol. This approach has been successfully used to verify a complex stateful internet-protocol QUIC, which runs as an application-layer protocol while using

UDP to send/transmit messages over the network. This use case is very similar to the domain of cellular-communication networks, where the control-plane protocols are executed as application-layer protocols, executing over an underlying stream control transmission protocol (SCTP) as a transport-layer protocol, which is used to communicate messages on-the-wire. In addition, we test the MME component (server) of the core network while generating test-messages that shall be sent by the eNB (client, radio-link).

We plan to demonstrate it by encoding the specification of the protocol in the IVy tool and to adopt its automated-test-message generation process to evaluate some open-source implementations of cellular-core-network (e.g., OpenEPC [1]). The measure of success would be the following:

- Implementation of SCTP transport protocol within IVy, to facilitate message transmission between the IVy system and the EPC-implementation (server). The functionality shall be evaluated by the ability to transmit “S1-Setup Request” Message and also to receive “S1-Setup Response” message.
- Specification of subset of S1AP/NAS protocol that enables the “UE-Attach Procedure”. This shall be evaluated by a successful simulation of a UE-Attach by an arbitrary UE with the EPC.

2 OVERVIEW

2.1 LTE Preliminaries

In the following subsection, we shall cover some aspects of the cellular network system, with an emphasis on the sub-systems that we shall be targeting our testing procedure.

2.1.1 LTE Network Architecture. As mentioned earlier, the LTE network architecture consists of three components: (1) the cellular device (UE), (2) the radio access network (base station (BS) and radio channels), (3) the EPC or the core-network (CN and wired communication channels between the BS and the core).

(1) **User Equipment (UE)**: The end-user communication device which is equipped with a Universal Subscriber Identity Module (SIM) card, serves as a terminal device. The SIM contains unique identification information for each subscriber, of which the following two parameters are vital: *(1) the international mobile subscriber identity (IMSI) number* and *(2) the international mobile equipment identity (IMEI) number*, apart from the associated cryptographic keys that are required to ensure security and privacy protection of various interaction-procedures with the network.

(2) **Base Station(BS)**: The cellular radio access network partitions the geographical space into hexagonal cells to cater to the needs of subscribers within the region. Each geographical cell is serviced by a single BS, located at its ’center’, that acts as an intermediary connecting the geographically dispersed subscribers with the CN. The E-UTRAN refers to the network between an eNBs(Base-stations) and the UEs.

(3) **Core-Network(CN)**: The CN or EPC consists of the following primary entities: *(1) Mobility Management Entity (MME)*: The MME serves as the primary interface between the BS and the CN. It manages the primary procedures of UE attach, UE detach, paging, etc, apart from the vital role of handling the mobility requirements of the UE. *(2) Home Subscriber Server (HSS)*: The HSS component of a service-provider maintains UE identities and subscription details, apart from the cryptographic keys and information required for authentication of the entities. *(3) Gateways*: Once the control-signalling has established the authenticity of the UE, verified their subscription and established a connection with the core-network, the Gateways handle the communication of data between the UE and the internet.

2.1.2 Initial UE Attach Procedure. Of the set of vital control-procedures handled by the MME, we shall be focusing on the functionality and implementation of the Initial UE Attach Procedure. When a UE wants to connect to the EPC, which is automatically initiated by the device often when the device has been rebooted, it scans the radio waves to determine the eNB(base station) with the strongest signal power and attempts to establish a connection with it. The Initial UE Attach procedure establishes a connection with the BS and triggers a subsequent connection to the CN by following the subsequent four sequences of operations:

(1) **Identification**: The UE sends an Attach Request message to the MME via the BS, by providing self-identifying information of its IMSI/IMEI and security capabilities.
(2) **Authentication**: On reception of the Attach Request by the BS, the MME forwards the request to the HSS to obtain authentication of the UE and generates an authentication challenge for the UE via an Authentication Request message, so that it may authenticate the legitimacy of the MME it has established a connection with. The UE uses its own master cryptographic key to authenticate the MME and responds with an Authentication Response message to the MME on success. If the authentication step is successful, the entities progress into the next stage of negotiating the security algorithm to establish a secure channel of connection between them.
(3) **Security Algorithm Negotiation**: From the security capabilities available in the UE, as communicated via the Attach Request message, the MME selects a security algorithm pairs (encryption and integrity) and communicates its choice to the UE along with the Message Authentication Code (MAC). Once the UE verifies the MAC, the UE and the MME establish a shared security context for protecting the secrecy and integrity of the messages transmitted over the channel in the future.
(4) **Secure Temporary Identifier Exchange**: The MME then sends an encrypted and integrity-protected Attach Accept Message, which includes a temporary identity called Globally Unique Temporary Identity (GUTI) that shall be used as an identifier-pseudonym for the UE. This is introduced to reduce the exposure of sensitive information like the IMSI/IMEI to potential eavesdropping and security-vulnerabilities in the system. The UE concludes the Attach Procedure, by transmitting an Attach Complete message to the MME followed, by the establishment of a security context between the BS and the UE.

2.2 Specification Methodology

In this section, we present the specification methodology using examples. To specify the S1AP control-plane messaging protocol between the BS and the MME, we shall use an abstract machine that monitors the protocol events. Conceptually, this machine observes the sequences of message interactions between two communicating entities(tester and target) by observing the protocol-events triggered by the messages on the wire at the interface between the tester and the network. When an event occurs, the abstract machine (which isn't a finite state by design) consults its state-representation to determine whether the event conforms to the protocol specification. If so, it updates its state to account for the event.

The protocol specification and the monitor are encoded in a language called IVy [4]. In IVy, the events associated with message-packets are modelled by an *action*. Constraints may be imposed upon the state and the parameters upon an event, by defining some conditions to be satisfied as a pre-condition or a post-condition. If these constraints are satisfied, the event is legal according to the protocol. Else, the events may be considered to be in violation of the protocol.

The protocol state is represented by a collection of functions or relations. The choice between the functional-representation or the relational-representation is based on how the state-information shall be queried in downstream actions. The two forms of representation are conceptually equivalent. Any element of the relational-set represents a predicate-abstraction of the actual event.

The properties of the protocol can thus be verified by the communication of abstracted messages between the entities. Or it may be used to generate actual messages on the wire, using a *shim* that generates actual network-protocol encoded messages to be sent on the wire, while also capturing the packet events received by the entity by tracing packets on the wire. While doing so, the shim triggers the specific protocol-events that get recorded as state-updates within the monitor. It is important to note that the specification is an executable monitor that observes the behaviour of all protocol nodes, in contrast to being an abstract implementation of the protocol entities.

An important capability provided by the IVy tool is the capacity to obtain a *generator* that produces random sequences of events that conform to the specification. The generator uses an SMT solver to randomly select a valuation of parameters that satisfy the 'require' constraints encoded in the pre-condition of the selected action. Consequently, the generator helps to generate a random sequence of packet-events in compliance with the protocol-specification that is sent to the test-target.

The specification may be encoded at a global level, accounting for all the roles that different protocol-entities may play, or at a role-based level, where only the specifications for a single-role are encoded. In the latter case, one needs to effectively assume that the role-being-modelled generates events that are correct according to the protocol since it's the task of the generator to create the corresponding messages. The task of the verification process is to "guarantee" the correctness of events generated by counterparts of the tester i.e roles played by the be multiple(or single) test targets as required by the protocol design. This framework of Assume/Guarantee specification-based testing has an important formal property: if the composition of roles ever violates the specification, then there must exist a failing assume/guarantee test for one of the many roles within the protocol.

2.3 Specification Examples

At this juncture, we would like to highlight the fact that we shall be using two different object-languages for the purpose of specifying the messages of the protocol.

(1) Ivy Tool: The Ivy language shall be used to represent the formal model of the protocol and the messages. The dynamics of the protocol, including the definitions of its state and the state transitions induced by the occurrence of protocol-events, shall be described using this language.
(2) RFC Message Format: The RFC message format, is a customized representation of messages, adopted by the W3C to describe web-based protocols within the RFC(request for comments) documents. This format shall be adopted to describe the bit-level representation of the message, which shall be used to generate Serializers/Deserializers of abstract messages(represented in Ivy), which need to be sent out on the physical wire.

The test-generator is thus composed of the following subsystems: Protocol-Model, Test-Generator, Serializer/Deserializer, and communication sockets of the operating system. Each of the components is implemented as C++ executables. The Ivy tool is used to describe the protocol-model, the test-generator and the "Interface-layer" connecting the formal protocol-model to the serializer/deserializer and the networking-sockets. The "Interface layer" contains embedded C++ code, which helps to associate C++ libraries describing the networking API of the OS and the Serializer/Deserializer API that may be generated using other tools. The Ivy tool is then used to compile the integrated system into a C++ codebase, which can then be compiled into a monolithic executable for the purpose of testing S1AP-Protocol.

3 IMPLEMENTATION 1: SCTP AND SERIALIZING IMPLEMENTATION IN IVY

For the purpose of conducting the tests in a real-world experimental test-bench, we need to enable the Ivy application to interact with the networking modules of the operating system. Further, we need to develop a mechanism by which the Ivy application may handle the following functions:

(1) translate abstract packets in the application layer to actual binary packet-streams on the wire
(2) parse the binary stream packets received on the-wire to infer application layer events and state-changes of the protocol

To fulfill the needs of the above functions, we need to develop serializer/deserializer(parser) that can translate between binary-bit-streams and the application-layer data structures that are used to represent the abstract messages in the formal model of the protocol. Further, we also need a mechanism by which the SCTP Transport Layer sockets of the operating system may be utilized for the transmission of these packets on the wire. We shall describe our approach to the development of these two capabilities within our system in the following sub-sections.

3.1 SCTP Socket Communication

The S1AP application layer protocol relies on a transport layer that supports the Stream-Control Transmission Protocol(SCTP).

The Ivy language tools have libraries to support UDP and TCP socket-based communication on the wire. However,it does not have a library to support communication via SCTP sockets of the operating system. We have identified two methods to extend the Ivy library to support SCTP communication:

(1) Modify the in-built library functions that support TCP socket based communication, for the purpose of creating a library-module to support SCTP socket usage.
    Implementation We have currently assumed the SCTP protocol to behave similar to TCP protocol, with respect to the number of handshakes needed and the sequence of internal state-transitions that model the functioning of the protocol. We have adopted a naive approach to facilitating SCTP communication, by modifying the socket options of the TCP library in Ivy, to the appropriate values required for SCTP.
(2) Utilize an interface-description language(IDL) with appropriate support for different layers of the protocol interaction.
    Implementation Apache Thrift platform provides language support to model different layers of the protocol-stack, which can then be compiled into appropriate objects of the target-language(C++ in the case of Ivy). The network/socket objects created using this tool, can then be integrated with Ivy, to facilitate communication over the network.

In our current implementation, though we were successful in describing the messages using the Thrift IDL, we couldn't successfully implement the Transport Layer integration within it. Hence, we had to revert back to the naive approach of writing an Ivy based library, to interface with the SCTP sockets of the operating system.

3.2 S1AP Serializer and Deserializer

We have chosen to represent the messages and their formats using the standard format adopted in the RFC within the web-standards community. These messages shall then be compiled using the Everparse[7] library, to obtain formally verified parsers and serializers, in C++. While compiling the RFC specifications that were developed for S1AP, we obtained certain errors, which were then identified to be due to the lack of sufficient support/features in the project everparse project, to handle instantiations of complex message structures. This was confirmed by raising the following issue on their project development-page [6].

```
struct DownlinkNASTransport {
  1: DownlinkNASTransportIES protocol_ies;
}

const list<S1apProtocolIES> DownlinkNASTransportIES = [
  {"id": ID_MME_UE_S1AP_ID_PIEID , "criticality": Criticality.REJECT,
  "type":MME_UE_S1AP_ID , "presence" :Presence.MANDATORY},
  {"id": ID_eNB_UE_S1AP_ID_PIEID, "criticality": Criticality.REJECT,
  "type":ENB_UE_S1AP_ID , "presence" :Presence.MANDATORY},
  {"id": ID_NAS_PDU_PIEID , "criticality": Criticality.REJECT,
  "type": NAS_PDU , "presence" :Presence.MANDATORY},
  {"id": ID_HandoverRestrictionList_PIEID , "criticality": Criticality.IGNORE,
  "type": HandoverRestrictionList, "presence" :Presence.OPTIONAL},
]


struct S1APElementaryProcedure{
  1: InitiatingMessage              initiating_message;
  2: optional SuccessfulOutcome     successful_outcome;
  3: optional UnsuccessfulOutcome   unsuccessful_outcome;
  4: required ProcedureCode         procedure_code;
  5: Criticality                    criticality = Criticality.IGNORE;

}

// EP11
typedef S1apElementaryProcedure DownlinkNASTransport_EP =
    {"initiating_message": DownlinkNASTransport,
    "procedure_code": ID_downlinkNASTransport_PC,
    "criticality": Criticality.IGNORE
    }
```

Listing 1. Downlink-NAS-Transport in Thrift

We finally decided to forge ahead with Apache Thrift project, as the tooling framework to create appropriate socket libraries, and also, to generate the serializer/deserializers for the project.

We shall now explain the protocol message-hierarchy by providing a description of one of the messages that's exchanged during the course of a "procedure" within the control-plane protocol. An illustration of the same is presented in the listing (1). The S1AP control-plane protocol consists of a collection of procedures. For example, when a mobile device is powered-on with its SIM installed, the Initial-UE-Attach procedure is initiated by the user-equipment(UE, mobile device). Each procedure requires the exchange of a series of messages between participating entities, with their associated pre-conditions and post-conditions. If any of these conditions is unsatisfied, the procedure terminates with the appropriate generation of error messages. These messages are visible on the wire, while the decision-making process that triggers these messages is not visible. The messages correspond to "elementary-procedures" according to the reference technical-specification documents. Each "elementary-procedure" in-turn consists of a sequence of 'Information

Elements"(IEs), which are themselves constituted by a set of primitive data-elements. This message hierarchy needs to be understood and represented using both the object-languages for specifying the nature of the protocol-state and for developing the serializer/deserializer component of the testing framework.

4 IMPLEMENTATION 2: S1AP PROTOCOL DESCRIPTION IN IVY

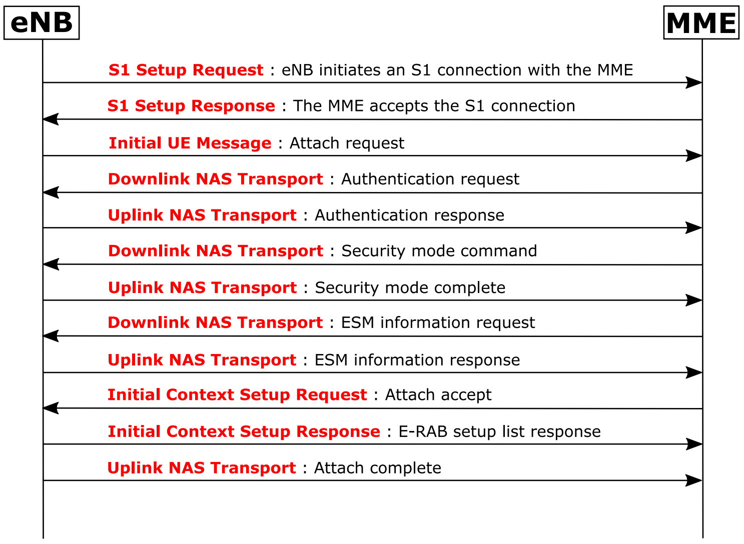

eNB
MME
S1 Setup Request : eNB initiates an S1 connection with the MME
S1 Setup Response : The MME accepts the S1 connection
Initial UE Message : Attach request
Downlink NAS Transport : Authentication request
Uplink NAS Transport : Authentication response
Downlink NAS Transport : Security mode command
Uplink NAS Transport : Security mode complete
Downlink NAS Transport : ESM information request
Uplink NAS Transport : ESM information response
Initial Context Setup Request : Attach accept
Initial Context Setup Response : E-RAB setup list response
Uplink NAS Transport : Attach complete

Fig. 2. Initial UE attach procedure

From the initial UE attach procedure in Figure 2, there are five message types: S1 Setup (Request/Response), Initial UE Message, Downlink NAS Transport, Uplink NAS Transport, Initial Context Setup (Request/Response). Each message has multiple mandatory/optional information elements (IEs) and protocol data units (PDUs). We focused on checking the mandatory elements because we aim to ensure that the tested implementation follows the requirements of the protocol. For example, S1 Setup message includes four mandatory IEs: Message Type (Procedure Code and Type of Message), Global eNB ID (PLMN [1] Identity, Choice eNB ID), Supported TAs [2] (TAC [3], PLMN Identity), Default Paging DRX [4]. Each message is included in the payload of s1ap_packet.

[1]PLMN: Public Land Mobile Network
[2]TA: Tracking Area
[3]TAC: Tracking Area Code
[4]DRX: Discontinuous Reception

```
#lang ivy1.7
include collections
include order

type cid
type pcode_bits
interpret pcode_bits -> bv[8]
type tac_octet
interpret tac_octet -> bv[16]
type plmn_octet
interpret plmn_octet -> bv[24]
object message_enum = {
    type this = {initial_message, successful_outcome, unsuccessful_outcome}
}
object drx_enum = {
    type this = {0, 32, 64, 128, 256} # 0 is error
}
object supported_ta = {
    type this = struct {
        tac : tac_octet,                        # Supported TAs - TAC
        plmn_identity2 : plmn_octet             # Supported TAs - Broadcast PLMNs
    }
    instance idx : unbounded_sequence
    instance arr : array(idx,this)
}
object s1_setup = {
    individual packet_state : cid
    individual pcode : pcode_bits               # Message Type - Procedure Code
    individual msg : message_enum               # Message Type - Type of Message
    individual plmn_identity : plmn_octet       # Global eNB ID - PLMN Identity
    individual global_choice_enb_id : plmn_octet # Global eNB ID - CHOICE eNB ID
    individual default_paging_drx : drx_enum    # Default Paging DRX
    individual supported_tas : supported_ta.arr # Supported TAs
    after init {
        packet_state := 0;
        pcode := 0;
        msg := message_enum.unsuccessful_outcome;
        plmn_identity := 0;
        global_choice_enb_id := 0;
        default_paging_drx := drx_enum.0;
        supported_tas := supported_ta.arr.empty
    }
```

Listing 2. S1 Setup test code (continued)

```
    action send(x:pcode_bits, y:message_enum, a:plmn_octet, b:plmn_octet,
                c:drx_enum) = {
        packet_state := packet_state + 1
        pcode := x;
        msg := y;
        ~
    }
    action recv returns(y:message_enum) = {
        y := message_enum.successful_outcome if plmn_identity > 0 &
                pcode = 17 & msg = message_enum.initial_message &
                global_choice_enb_id > 0 & default_paging_drx ~= drx_enum.0
                else message_enum.unsuccessful_outcome
    }
    action state returns(s:cid) = {
        s := packet_state
    }
}

import action ask returns (x:pcode_bits)
export action ask_and_check_pcode returns(ok:bool) = {
    ok := ask = 17
}

export s1_setup.send
export s1_setup.recv
export s1_setup.state
```

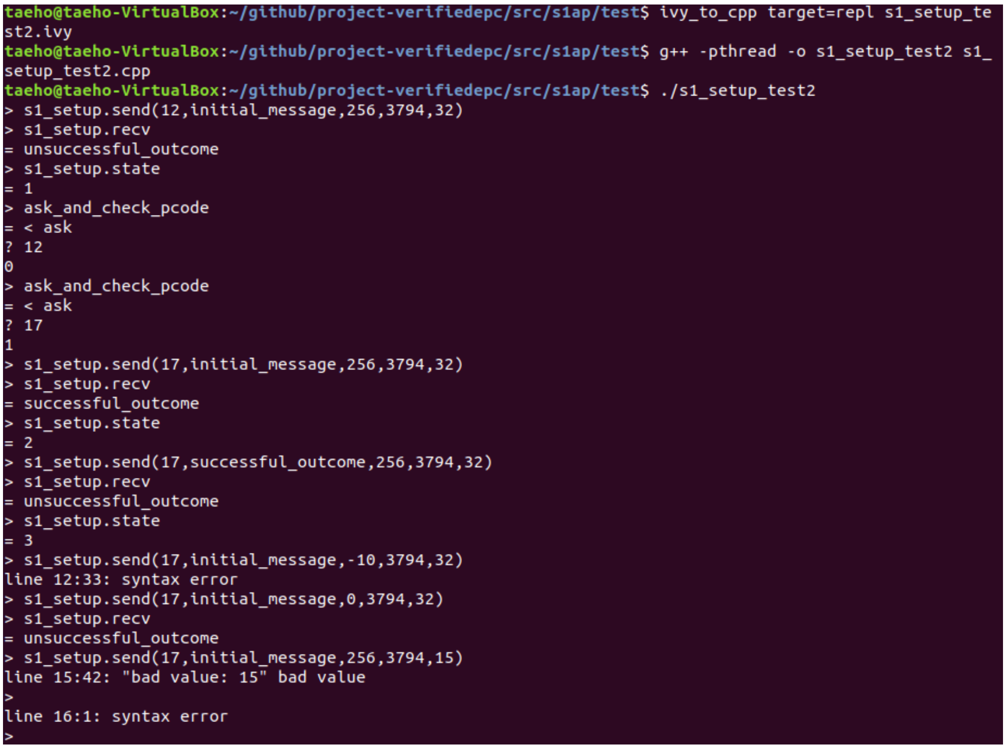
Listing 3. S1 Setup test code

In listing 2, we declare multiple types to follow the protocol at first. pcode_bits, tac_octet, and plmn_octet types have an integer with specific number of bits. message_enum and drx_enum types have enumerated values, and supported_ta type has an array. Our program can check if each element of S1 Setup message is fit with the correct data type In the s1_setup object, each individual element is connected with a predefined type. After the initialization, two action functions send and recv are defined to confirm that the s1_setup message follows the protocol. When the message format is correct, our s1_setup.recv action function returns successful_outcome message. In addition, the user can check the correct procedure code by calling the ask_and_check_pcode action function.

Similarly, we have implemented test codes for other four messages (initial_ue_message_test.ivy, downlink_nas_transport_test.ivy, uplink_nas_transport_test.ivy, initial_context_setup_test.ivy). We can convert ivy codes to .cpp codes by using the ivy_to_cpp command and make executable files by typing g++ -pthread -o [executable file] [ivy code file].

5 EMPIRICAL EVALUATION

After extracting the mandatory elements of message, we can check if the message follows the protocol format correctly by executing our test code. For example, there are five mandatory elements in the S1 setup message: procedure code, type of message, plmn identity, choice eNB ID, and default paging DRX. The correct procedure code of the S1 setup message is 17, so a message with 12 is incorrect. Since the test program captures a message, the current packet state (packet number) increases by 1. Likewise, the program increases the current packet state whenever a message is received. The message sender can confirm the correct procedure code by calling the ask_and_check_pcode action function. The type of message should be "initial_message". If the sent message has a type "successful_outcome", "unsuccessful_outcome", or an unsupported type, our test program returns "unsuccessful_outcome" or syntax error. Likewise, the plmn identity and choice eNB ID must be greater than 0 and less than the maximum values. And the default paging DRX must have one of {32, 64, 128, 256} to pass through our test program. In the same way, we can check the Initial UE message, Downlink NAS Transport message, Uplink NAS Transport message, and Initial Context Setup message with our test programs to verify the Initial UE attach procedure.



```
taeho@taeho-VirtualBox:~/github/project-verifiedepc/src/s1ap/test$ ivy_to_cpp target=repl s1_setup_te
st2.ivy
taeho@taeho-VirtualBox:~/github/project-verifiedepc/src/s1ap/test$ g++ -pthread -o s1_setup_test2 s1_
setup_test2.cpp
taeho@taeho-VirtualBox:~/github/project-verifiedepc/src/s1ap/test$ ./s1_setup_test2
> s1_setup.send(12,initial_message,256,3794,32)
> s1_setup.recv
= unsuccessful_outcome
> s1_setup.state
= 1
> ask_and_check_pcode
= < ask
? 12
0
> ask_and_check_pcode
= < ask
? 17
1
> s1_setup.send(17,initial_message,256,3794,32)
> s1_setup.recv
= successful_outcome
> s1_setup.state
= 2
> s1_setup.send(17,successful_outcome,256,3794,32)
> s1_setup.recv
= unsuccessful_outcome
> s1_setup.state
= 3
> s1_setup.send(17,initial_message,-10,3794,32)
line 12:33: syntax error
> s1_setup.send(17,initial_message,0,3794,32)
> s1_setup.recv
= unsuccessful_outcome
> s1_setup.send(17,initial_message,256,3794,15)
line 15:42: "bad value: 15" bad value
>
line 16:1: syntax error
>
```

Fig. 3. S1 Setup test execution

6 RELATED WORK

The primary research exercise that we undertake is to determine an good methodology for providing a formal-specification for the control-plane protocols used in the cellular communication infrastrucuture. When we think about task for formalizing a specification for a protocol, there are two main use-cases: (1) to use the specification as an input for verifying that a particular model of its implementation satisfies some desriable properties, and (2) to enable the development of correct by construction implementations of the protocol. In our work, we explore the domain of specification as a means to conduct testing of an implementation, by facilitating a mechanism of automated generation of test-messages that shall be used to test the communication interface between two entities.

Most related researches have focused on the issue of formal-verification of correctness and security properties that are provided by the authentication protocols used in this domain. They have attempted a formal methodology of testing a communication protocol. One research team developed a formal specification of the wire protocol [5]. They used the specification to generate automated randomized testers for implementation of QUIC (Quick UDP Internet Connections) that is an Internet secure transport protocol. The testers take one role of QUIC protocol: either interacting with the other role to generate full protocol executions or verifying that the implementations conform to the formal specification. Considering the process of evaluating strict compliance with standards in various communication environments, it is necessary to test implementations in adversarial environments. In addition, they developed and released Ivy Tool [4] to evaluate QUIC protocol. Although the protocols are different, the considerations and suggested approaches covered in this study can also be used to solve our problems. Another interesting research project, that aligns with our endeavour, is the Project Everest [5], which attempts to create a formally verified stack to guarantee verified low-level implementations of the HTTPS stack. Especially, they include cryptographic algorithms and develop new implementations of existing and new protocol standards while formally proving that their implementations provide a secure-channel abstraction between the communicating endpoints.

There are also several tasks related to the verification of cellular network protocols. CNetVerifier [8] is a tool to analyze the inter-layer, inter-domain and inter-system protocol interactions within the control-plane of cellular communication networks. The tool adopts a model-checking methodology within its two-phase protocol-diagnosis strategy to detect issues arising from (1) design problems within the protocol standards specification, and (2) operational mistakes of the service-provider. The verification strategy, however, is user-centric, i.e. the properties that are verified is related to the user-entities and cannot be used to examine interactions between the BS and CN, which would be of interest to improve the operational needs of the carrier/service-provider. In this work, the protocol is modelled as two interacting FSMs, with one representing the UE, and the other representing the network entity(BS, MME, etc), within the model-checking framework, SPIN. The measurement based verification is handled at the UE level.

LTEInspector [2] which employs a property-driven adversarial model-based testing philosophy. LTEInspectortakes the relevant 4G LTE abstract model and a desired property (ϕ), and tries to find aviolation of ϕ in the model. The tool checks for the following properties - authenticity, availability, integrity, and secrecy , all from the perspective of the end-user/customer. The model they develop comprises of as synchronous communicating finite state machines which abstract away the functionality while ignoring low-level implementation details. They adopt an instance of the parameterized system verification problem (i.e., parameterized by the number of protocol participants).Their instantiation of the LTEInspector framework, however,adopts the following constraints: (1) They consider only a single layer of the protocol stack in isolation; (2) For the sake of scalability, they only model packet type and do not model critical data or packet payload, missing out on interesting data-/payload-dependent protocol behavior; (3) Their adversary instantiation cannot handle protocols spanning across different layers of the stack.

In the most recent work, 5GReasoner [3], the authors adopt a modelling procedure which models about five different control-plane procedures, with a FSM modelling each layer of the stack. The state-machines corresponding to different entities communicte via a public, adversary-controlled channel, where the adversary is also modelled as a FSM. The NAS layer protocol packets which

[5]https://www.microsoft.com/en-us/research/project/project-everest-verified-secure-implementations-https-ecosystem/

constitute data-packets of the RRC layer protocols, is modelled using a behaviour-specific predicate-abstraction methodology, in which they do not directly model the data, but only predicates-over the data which are essential to verify the specific properties they are examining. These models are implemented in two infinite-state model-checkers and a cryptographic protocol verifier. They have verified about 187 properties which were extracted either from the standards, or were specified based on domain-knowledge.

7 CONCLUSION

In this paper, we have tried to extend the application of the formal verification of Internet-protocols to the domain of cellular communication networks. We used the IVy tool for providing a formal specification of the S1AP protocol. We have attempted to make our system design flexible and malleable to different protocols by incorporating the use of an interface-description language framework like Apache Thrift, to facilitate the automated generation of serializers and deserializers of the messages. Consequently, we have developed reference specification of the S1AP message-format in RFC and Thrift IDLs. In addition to this, we have developed a partial model of the S1AP protocol within Ivy by defining multiple types, protocol states and protocol-events represented as actions. We finally implemented some test code for S1AP messages. Since the current implementation checks each message for correctness of its structure, our future work involves testing the serialized messages and updating the protocol states. With the steps taken during the course of the project, we have identified the areas we need to focus on, for creating an automated test-generation engine based on Ivy.

REFERENCES

[1] M. Corici, F. Gouveia, T. Magedanz, and D. Vingarzan. 2010. Openepc: A technical infrastructure for early prototyping of ngmn testbeds. In *International Conference on Testbeds and Research Infrastructures*. Springer, 166–175.

[2] S. Hussain, O. Chowdhury, S. Mehnaz, and E. Bertino. 2018. LTEInspector: A systematic approach for adversarial testing of 4G LTE. In *Network and Distributed Systems Security (NDSS) Symposium 2018*.

[3] S. R. Hussain, M. Echeverria, I. Karim, O. Chowdhury, and E. Bertino. 2019. 5GReasoner: A Property-Directed Security and Privacy Analysis Framework for 5G Cellular Network Protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 669–684.

[4] K. L. McMillan and L. D. Zuck. 2019. Compositional Testing of Internet Protocols. In *2019 IEEE Cybersecurity Development (SecDev)*. IEEE, 161–174. https://ieeexplore.ieee.org/abstract/document/8901577.

[5] K. L. McMillan and L. D. Zuck. 2019. Formal specification and testing of QUIC. In *Proceedings of the ACM Special Interest Group on Data Communication*. 227–240. https://dl.acm.org/doi/abs/10.1145/3341302.3342087.

[6] P. Prahladan. 2020 (accessed May 5, 2020). *RFC Parsing Error 11*. https://github.com/project-everest/everparse/issues/11

[7] T. Ramananandro, A. Delignat-Lavaud, C. Fournet, N. Swamy, T. Chajed, N. Kobeissi, and J. Protzenko. 2019. Everparse: verified secure zero-copy parsers for authenticated message formats. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1465–1482.

[8] G.-H. Tu, Y. Li, C. Peng, C.-Y. Li, H. Wang, and S. Lu. 2014. Control-plane protocol interactions in cellular networks. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 223–234.