Towards Automated Fuzzing of 4G/5G Protocol Implementations Over the Air

Matheus E. Garbelini, SUTD
Zewen Shang, SUTD
Sudipta Chattopadhyay, SUTD
Sumei Sun, I^{2}R, A*Star
Ernest Kurniawan, I^{2}R, A*Star

***Abstract*—Recent rise in the mobile network communication vulnerabilities highlights the need for systematic security testing frameworks for communication protocols. In this paper, we propose a real-time framework to fully manipulate the 4G and 5G data-link and network communication to the base station (eNB/gNB). This is for experimenting and testing the security of data-link protocols such as Media Access Control (MAC), Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP) and network protocols such as Radio Resource Control (RRC) and Non-access stratum (NAS). Although we focus on the base station, our framework is equally applicable for manipulating the communication to the user equipment (UE). An appealing feature of our framework is that it automatically constructs the protocol state machine during normal communication. This allows us to validate the response from the base station when it is subjected to unexpected packet sequences. Our framework also exposes an application programming interfaces (APIs) for designers to install custom attack scenarios. We have implemented our framework and used it to generate several (adversarial) scenarios that include injection of malformed and out-of-order packets as well as flooding certain packets. Our evaluation revealed crashes in OpenAirInterface (OAI) UE and gNB, as well as in Open5GS core network. Additionally, we guide our validation via the automatically constructed state machine and have caught most adversarial scenarios during our evaluation. We envision our proposed framework to provide the foundation for automated security testing of 4G/5G data-link protocol implementation.**

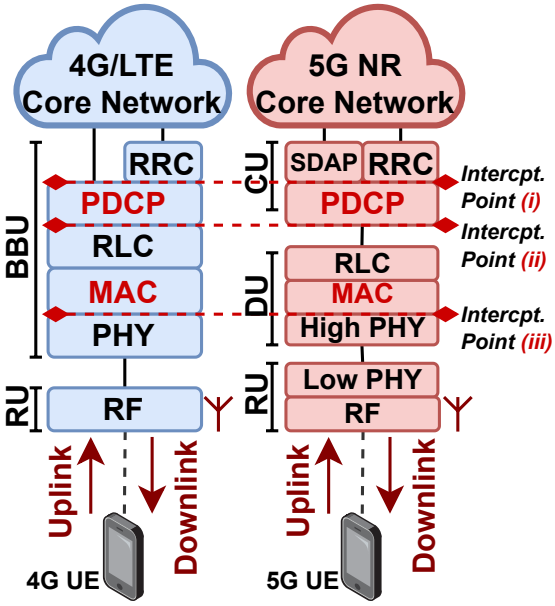

Fig. 1. An illustration of 4G and 5G Radio Access Network (RAN) protocol layers stack targeted by our fuzzing framework. The interception points where the fuzzing takes control of the communication are highlighted.

I. Introduction

Mobile communications have become essential for critical infrastructures as well as for our day-to-day life. However, recently discovered security vulnerabilities in both 4G and 5G protocols [1], [2] undermine the trust in 4G/5G networks. Such vulnerabilities range from design flaws to specific implementation issues on the firmware of modem chipsets that are used in high-end smartphones. Due to the closed nature of 4G/5G protocol stacks, they are not amenable to verification and static analysis. As a consequence, to uncover the hidden vulnerabilities in closed-source 4G/5G stacks, it is of critical importance to build technologies and tools that can automatically interact over-the-air (OTA) with a black-box device under test (DUT) and systematically steer the protocol states to potential implementation vulnerabilities.

In this paper, we propose an OTA fuzzing framework that aims to find vulnerabilities in the implementation of 4G/5G data-link protocols. Our framework focuses on fuzzing the uplink communication from a UE to the mobile base station (i.e., eNB or gNB). Therefore, OTA fuzzing leverages control over the MAC, RLC and PDCP layer protocols (OSI Layer 2) and additionally RRC and NAS (OSI Layer 3) to inject arbitrarily malformed or incorrect sequence of packets or flooding to the base station. This is carried out with the objective to trigger implementation flaws. As a by-product of our framework, we also expose a flexible application programming interface (API) for deep packet inspection (DPI). Designers can use such API to easily create custom 4G/5G attack scenarios from the UE to the eNB/gNB. Additionally, such APIs allow introducing customized fuzzing extensions or attack detection in the MAC layer. In essence, we provide a flexible and extensible framework for a comprehensive security evaluation of 4G/5G enabled systems.

While designing our OTA fuzzing framework, we face several technical challenges. Firstly, fuzzing at 4G/5G MAC Layer (i.e., OSI Layer 2 - Data link) requires a highly efficient algorithm to control the data-link communication. Such control may involve mutating (i.e., fuzz) arbitrary packet fields. Moreover, detecting invalid or non-compliant responses from the base station, during live communication for a complex protocol like 4G/5G, requires a comprehensive fuzzing and validation strategy. Such a strategy needs to be aware of the possible protocol state transitions during mobile communication (i.e., *downlink/uplink* exchange). To resolve these challenges,

we employ an efficient fuzzing and validation methodology guided by the protocol state machine which is also constructed automatically during normal communication. Subsequently, the state machine is used to guide the validation process to indicate anomalous responses.

An appealing and distinguishing feature of our framework is that it takes control of the communication for fuzzing in real-time. As a result, our approach has full access over the contextual information (e.g., security configurations) that are only available during a live communication. For example, 4G/5G protocols reside in different software components as specified in the 3GPP specification [3], [4]. Thus, our fuzzing campaigns intercept protocol packets at different components of the *core network* to have full control of the communication. In addition, protocol-specific behaviors such as *integrity, encryption* and *packet fragmentation* are handled at different *interception points* on the eNB/gNB to ensure full uplink and downlink control. Figure 1 outlines these interception points installed by our framework.

State-of-the-art fuzzing approaches are either not applicable for closed source and stateful protocol stacks [5], [6], or they may require significant manual effort to hand-code protocol state machine and the environment within the fuzzer [7], [8]. In contrast, our approach generates protocol state machine automatically during communication, and it is applicable to effectively test the security of closed-source and stateful 4G/5G implementation. In summary, we make the following contributions in this paper:

1) We present a framework to fully manipulate the lower layers of 4G/5G protocol stack (MAC, RLC, PDCP).
2) We show that the processing time of our packet manipulation outperforms state-of-the-art fuzzers [9], [10].
3) Our work exposes easy-to-use API for downlink manipulation. Such API can access any 3GPP protocol from the *interception points* shown in Figure 1.
4) Our approach is generic and applicable to other wireless protocols in which low-latency is a strict requirement.
5) For both 4G and 5G, we have evaluated the framework by generating several malformed, out-of-order and flooded packets across MAC, RRC, PDCP, RLC and NAS protocols. This revealed crashes at gNB, UE and core network. By having these packets, we have also comprehensively evaluated our validation process which is guided by the automatically constructed protocol state machine. The validation detects our generated attacks at all layers with high accuracy and negligible false positives.

II. Framework Overview

In this section we present an overview of our fuzzing and response validation framework. The framework is outlined in Figure 2. The objective of our proposed framework is to allow fuzzing the eNB/gNB base station down to the lower 4G/5G MAC layer while also capturing potential attacks to the base station launched from the UE.

Attack Model: The attacker model employed by our fuzzing framework exposes an Adversary-Controlled uplink or downlink channel. This translates to a malicious UE or eNB/gNB that is capable to sniff communication or inject and replay packets to the other party without restrictions. Such an attacker model is commonly used in previous security analysis works [2], [11]. Nonetheless, in our evaluation, we focus on the attacker at the UE side, which targets the eNB/gNB. The key components of Figure 2 are discussed in the following.

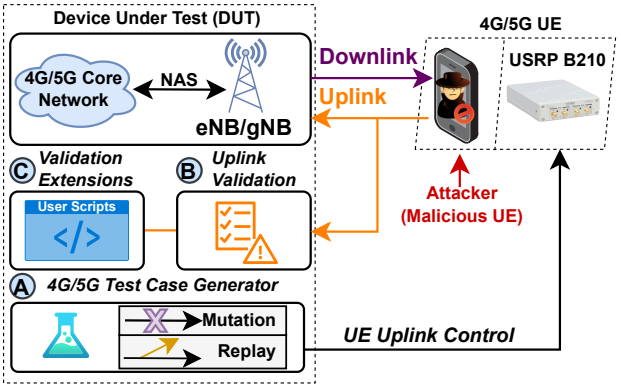

Fig. 2. An illustration of our 4G/5G RAN fuzzing framework

UE Uplink Test Case Generator (A): This component is responsible for generating invalid uplink frames down to the *MAC layer* by using off-the-shelf software defined radios (SDR) such as *USRP B210* [12]. Such invalid frames are generated with the goal to find vulnerabilities in the eNB/gNB stack implementation. To this end, this component emulates a real UE using *Open Air Interface stack* [13] and provides an API to create attack scenarios (i.e., test cases) targeting the eNB/gNB.

The test cases may ***(i)*** inject malformed, ***(ii)*** send out-of-order (i.e., re-played) or ***(iii)*** flooding uplink frames down to the MAC layer using a SDR. The usage of this component is two-fold. Firstly, we can use this component to recreate some attack scenarios targeting the eNB/gNB. Secondly, we can use this component to stress test the UE uplink response validation component, as discussed in the following.

UE Uplink Response Validation (B): In our framework, The validation component informs the fuzzer if the response received from the (malicious) UE is valid according to a protocol state machine, which is automatically generated based on a few simple rules and by decoding the packets during normal communication (see Section III for details). Such a state machine informs the correct order that 4G/5G frames should be received at the eNB/gNB node during a fuzzing campaign. The validation stage is essential to detect attacks or to guide a fuzzer towards triggering an implementation failure. These failures may result in a modem firmware crash (i.e., Denial of Service) or security bypass vulnerability.

In short, currently we consider the following scenarios when validating UE responses. This is generally applicable to both 4G and 5G, amongst other wireless protocols:

1) ***Unresponsiveness***: UE stops responding for an unusual amount of time or indefinitely due to a modem hang.
2) ***State Machine Corruption***: UE responds with an out-of-order frame which should not occur during normal communication with eNB/gNB. This indicates that the UE protocol states were affected during the *fuzzing*.

eNB/gNB Attack Detection Extensions (C): This component extends our framework by providing a DPI API to user-provided scripts which run during uplink reception. Therefore, such scripts can create rules to identify possible attacks to the eNB/gNB down to *network layer 2* (i.e., MAC). Hence, this allows the eNB/gNB designer to disconnect a potentially malicious UE from the RAN. Currently, our framework include sample scripts to detect MAC flooding attacks, reception of out-of-order attacks and malformed uplink frames that can contain invalid fields in multiple RAN protocol layers such as MAC, RLC, PDCP, RRC and NAS. Such extensibility can enable future development of real-time intrusion detection systems at the data-link between UE and eNB/gNB.

III. Design

In this section, we describe the workflow of our framework (see Figure 3) and detail on how downlink and uplink packets, which are generated at the *eNB/gNB* or *UE*, are intercepted and handled. We illustrate this in the context of fuzzing and layer 2 (*L2*) test case generation. The workflow is illustrated in Figure 3 and highlights the downlink and uplink handling whenever the UE connects to eNB/gNB.

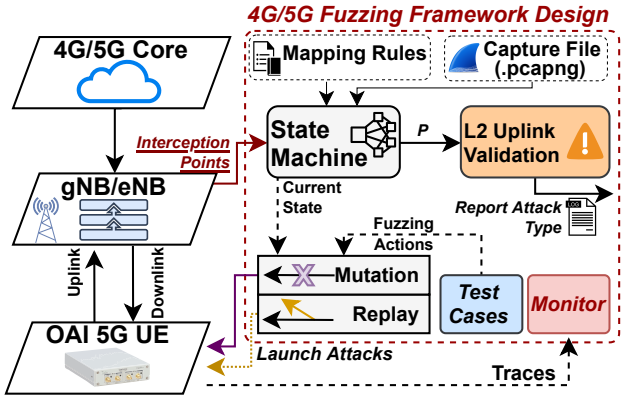

Fig. 3. Design of our 4G/5G Fuzzing Framework for L2 packet interception

Latency Requirements: Recall that downlink and uplink packets are blocked and forwarded to our framework via *interception points* (see Figure 1). These *interception points* yield control over L2 packets which are decoded or generated at the eNB/gNB. With such an interception-based approach, it is crucial to not block MAC frames for more than the time of *downlink/uplink* transmission slot. Otherwise, real-time communication fails and packets cannot be delivered over the air. The time of the *slot transmission* (i.e., *numerology*) is configured at the eNB/gNB, and it can vary between $6.25us$ and $1ms$ [14]. Our interception strategy leverages shared memory to minimize the inter-process-communication (IPC) time between eNB/gNB/UE and the fuzzer.

Nevertheless, it is worth mentioning that 4G/5G *numerology* is adjustable and the throughput is not relevant to a fuzzer. This is because our framework targets finding bugs and improving OTA testability. Therefore, our fuzzing architecture only considers eNB/gNB *transmission slot* configuration that are above $250us$ to ensure real-time communication. This is sufficient for fuzzing MAC or upper layer frames.

Interception Points: As previously shown in Figure 1, three *interception points* have been implemented in our framework: *(i)* After MAC, *(ii)* before RLC and *(iii)* at the PDCP layer. These are not only used to control MAC packets, but also packets that are yet-to-be fragmented by the RLC layer. More specifically, we intercept before and after packets are encrypted or applied integrity protection at the *PDCP layer*. The rationale of these *interception points* is two-fold. Firstly, we ensure that any field manipulation will be received at the base station without being dropped due to encryption problems. Secondly, we aim to have complete control of fragmented packets. The *interception point* at the PDCP layer allows our framework to modify both the PDCP payload and RLC fragments of the packet.

State Mapper: Such component informs the protocol state of eNB/gNB/UE in real-time and it is essential for validating potential problems with the DUT during a fuzzing campaign. The state machine is also used to detect potential attacks at the eNB/gNB. This is accomplished by checking whether the received packets at the eNB/gNB are in line with the state machine.

Due to the complexity of *3GPP* protocols, instead of generating the state machine manually, we use a lightweight method to learn the protocol state machine with the following inputs: *(i)* a set of *Mapping Rules* which indicates the way to identify a state for an arbitrary protocol and *(ii)* the capture traces (i.e., *pcap* file) of the communication that serves as reference of correct sequences during learning. Then, after a learning process, the *State Mapper* outputs the reference sequence of packets (i.e, state transitions) between the eNB/gNB and UE. This is to identify whether a given packet P is *expected* to be received at certain state label S during live communication.

As exemplified in Figure 4, every packet intercepted at the *interception points* is parsed and assigned a unique state label S. Therefore, S is generated from the concatenation of certain packet information such as direction $P.dir$ (***TX** for Downlink or **RX** for Uplink*), packet type $P.type$ and the protocol layer name $L.name$. To this end, the *Mapping Rules* M_u contains a set of rules that tells the *State Mapper* on how to obtain the correct information during packet decoding. This is then used to generate a state label S that corresponds to a unique packet P. Therefore, a different packet P' would consequently generate different state label S'.

The high-level steps to generate S are described in procedure *state_mapping* of Algorithm 1. More importantly, in Line 13, a lookup table converts the raw value of a packet

Algorithm 1 `state_mapping` Procedure

```
1: Input: Packet P, Mapping Rules M_u
2: Output: State label S generated for packet P
3:
4:  Decode packet P to get P.layers and P.fields
5: for each L ∈ P.layers do
6:     ▷ Check if the packet layer L match a rule in M_u
7:     for each R ∈ M_u do
8:         if L satisfies R.filter then
9:             ▷ Check if any field f is found within R.type_fields
10:            for each f ∈ P.fields do
11:                if f ∈ R.type_fields then
12:                    v := value of field f in packet P
13:                    P.type := lookup[v]
14:                    goto Line 20
15:                end if
16:            end for
17:        end if
18:    end for
19:    ▷ Label the state when matched with the rules
20:    if (L ≠ empty ∧ P.type ≠ empty) then
21:        ▷ Create the state label S
22:        S := P.dir ⊕ L.name ⊕ P.type
23:        return S
24:    end if
25: end for
```

field f, matched with rule $R.type\_fields$, to a *type string*. Then S is generated by a string concatenation in Line 22.

This approach is used to validate intercepted packets. For example, consider that packet P' is received during live communication when the current state label is S. If none of the outgoing transitions of S leads to S', then we consider packet P' as an anomaly. Otherwise, the state machine moves to state label S' and the packet is considered valid.

Mapping Rules M_u: The *Mapping Rules* consists of a set of rules for each relevant *3GPP* protocol layer (see Figure 5). As previously discussed, this set of rules is fed to the *State Mapper* for use during validation or fuzzing tasks.

As illustrated in Figure 5, the property *"Filter"* filters a protocol by its layer name (c.f., $R.filter$ of Algorithm 1) and the property *"StateNameField"* identifies each *type-field* of the protocol layer that can be used to map a packet to a protocol state (c.f., $R.type\_fields$ of Algorithm 1). Such rules follow the *PCAP* filtering syntax, which is used in the *Wireshark* packet analyzer program.

We note that while PDCP has only one field that identifies its type, *MAC*, *RLC* and *NAS* can be identified by multiple fields depending on the communication context. This is useful for detecting replayed packets. In our evaluation, by using 6 rules, we had generated a state machine for 4G with 20 states (86 transitions) and another with 14 states (74 transitions) for 5G.

L2 Uplink Validation: The *L2 uplink validation* targets detection of three main attack types: sending *malformed*, *out-of-order* packets and *flooding* attack. The validation process is outlined in Algorithm 2. As shown in Algorithm 2, after a packet P is received (Line 9 to Line 13), we check the validity of the packet by calling the function *expected*. We note that the label of the packet is computed on-the-fly via the state mapper discussed in the previous section. Concurrently, the *expected* function abstracts the details of state machine and informs the validation process if the packet P belongs to any of the outgoing transitions of the current state (i.e., $current\_state$). If the received packet is **not** *expected*, then it is considered an out-of-order attack. Regardless of the packet's validity, the packet summary is appended to a predefined list P_{labels} for later analysis. Specifically, after the timer times out, the list is analyzed (Line 14 and Line 22), and if there exists more than τ instances of a packet, then the validation process detects a flooding attack. In our evaluation we set τ to be 10. Finally, in Line 9-10, the validation detects a malformed packet if any *decoding error* occurs at the *interception points* (see Figure 1).

Fuzzing and Test cases: As captured in Figure 3, our framework exposes an *API* to manipulate the downlink/uplink payloads between UE and eNB/gNB. Such manipulation (i.e., *fuzzing actions*) consists of either mutating protocol fields or replaying packets during uplink/downlink communication. Using this feature, our framework sets up the workflow for 4G/5G fuzzer for both eNB/gNB and UE and it is able to launch *layer 2* attacks. Such attacks are initiated by the component *Test Cases* (see Figure 3) which runs user provided scripts. Particularly, this component serves as a baseline for our framework evaluation, as the component can be used to create over-the-air attacks from the UE to eNB/gNB and vice-versa.

IV. Evaluation

Implementation and Setup Our framework is implemented in C++ (*7879 LoC*). This includes modifications to Wireshark that improve decoding speed for 4G/5G protocols and expose additional information for ASN.1 decoded fields. Additionally, we develop patches for OpenAirInterface5G stack to enable the *interception points* as discussed in Section III. To generate

Algorithm 2 `get_attack_type` Procedure

```
1: Input: Packet label P, Flooding threshold τ
2: Output: Attack Type
3: ▷ global list P_labels
4: ▷ global timer T
5: ▷ global function expected
6:
7: P_labels ← P_labels ∪ {P}
8: ▷ Check for malformed or out-of-order packets
9: if decoding error detected then
10:    Output: Packet malformed
11: else if ¬expected(current_state, P) then
12:    Output: Out-of-order attack
13: end if
14: if T.times_out() then
15:    ▷ Restart timer
16:    T.restart()
17:    ▷ Check if frequency of labels in P_labels exceeds τ
18:    if max(histogram(P_labels)) > τ then
19:        Output: Flooding attack
20:    end if
21:    P_labels ← ∅
22: end if
```

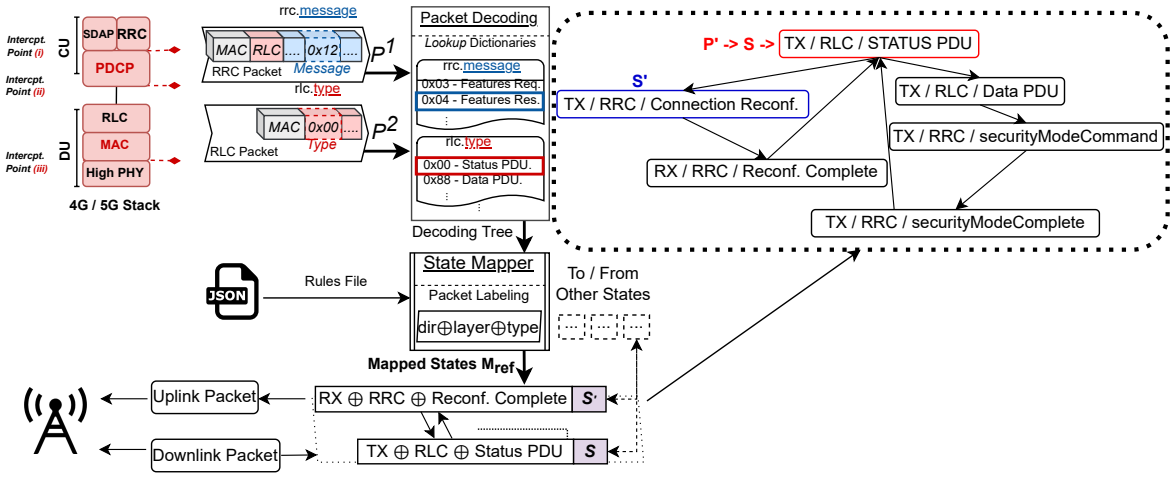

Fig. 4. Overview of State Mapping Generation

```
{ // NAS Protocol Filter
  "Filter": "nas-eps",
  "StateNameField": [
    "nas_eps.nas_msg_esm_type",
    "nas_eps.nas_msg_emm_type"]
},
{ // RRC Protocol Filter
  "Filter": "nr-rrc",
  "StateNameField": "nr-rrc.c1"
},
{ // PDCP Protocol Filter
  "Filter": "pdcp-nr",
  "StateNameField":"pdcp-nr.pdu-type"
},
{ // RLC Protocol Filter
  "Filter": "rlc-nr",
  "StateNameField": [
      "rlc-nr.am.cpt",
      "rlc-nr.am.frame-type",
      "rlc-nr.channel-type"]
},
{ // MAC-NR Protocol Filter
  "Filter":  "mac-nr",
  "StateNameField": [
    "mac-nr.ulsch.lcid",
    "mac-nr.rnti-type"]
}
```

Fig. 5. *Mapping Rules for 5G MAC, RLC, PDCP, RRC and NAS*

reference traces of valid 4G and 5G communication for the *State Mapper* (Section III), we connect the UE modems *Telit 910EU-V2 (4G)* and *Quectel RM500Q-GL (5G)* to OAI eNB and gNB respectively.

Although our framework supports running OAI stack with real hardware using USRP B210 for over-the-air communication, we set up our evaluation to run in RF simulation mode to ensure reproducibility of the results. We observed that interference may introduce random disconnections or unexpected responses during communication. Such *"random"* issues that appear with real UE hardware are expected to be reduced as 4G/5G OAI stack evolves.

Positive Tests: We start testing positive scenarios on a real *Quectel RM500Q 5G* modem, which results in the *L2 Uplink Validation* detecting 232 valid packets out of 243 total exchanged packets. This means that the base reference model employed by the *State Mapper* indicates 5% false positive rate.

Negative Tests: Initially, we start Open5GS as the Core Network and set up the *OAI UE* to launch attacks against OAI eNB and gNB via the *Test Case Generator* as shown in Figure 2. Next, multiple scenarios are evaluated based on this setup to target different 4G/5G protocol layers and fields during UE to eNB/gNB communication. As shown in Tables I and II, we employ negative tests for representative fields in the *MAC, RLC, PDCP, RRC* and *NAS* layers. Such tests may either send malformed packets by randomly mutating a specific field, flooding specific packets or inject an out-of-order packet (*replayed*) to the eNB/gNB. Finally, we also created flooding attacks by consecutively sending certain packets to the eNB/gNB. In the third column of Table I and II, we indicate whether the *L2 Uplink Validator* detected the negative scenario launched from the UE. The column *"Any Consequence?"* informs if a crash happens to the *eNB/gNB*, *UE* or *core network* stacks during the negative tests' evaluation.

Our validation detects the majority of the scenarios listed in Tables I and II. More interestingly, mutating 4G MAC fields such as *"mac-lte.sch.extended"* and *"mac-lte.control.bsr"* causes the OAI eNB to either crash or hang. Similarly, the 5G OAI gNB crashes upon receiving an invalid *"mac-nr.dlsch.lcid"* field as depicted in Table II. Furthermore, the Core Network (Open5GS) has crashed in one instance due to a malformed RRC field *"ulInformationTranser_element"*. In summary, both eNB and gNb are susceptible to DoS attacks in the MAC layer due to improperly handling the reception of malformed or replayed packets. On the other hand, only the gNB is vulnerable to a certain flooded packet.

Nonetheless, our validation misses the detection of certain attacks. Firstly, after we mutate specific fields such as *"mac-*

TABLE I
NEGATIVE TEST CASES AND THEIR IMPACT FOR OAI LTE *e*NB

Protocol	Malformed Packet Fields	Detected?	Any Consequence?
MAC	"mac-lte.sch.extended"	✓	OAI eNB Crashes
MAC	"mac-lte.ulsch.lcid"	✓	
MAC	"mac-lte.control.power-headroom.level"	✓	
MAC	"mac-lte.control.bsr"	✗	OAI eNB Hangs
MAC	"mac-lte.sch.length"	✓	
RLC	"rlc-lte.am.header"	✓	
RLC	"rlc-lte.am.fixed.sn"	✗	
RRC	"lte-rrc.rat_Type"	✓	
RRC	"lte-rrc.accessStratumRelease"	✓	
PDCP	"pdcp-lte.reserved"	✓	
Protocol	**Replay Scenario (Message, State)**		
MAC	(Short/Long BSR, Msg3)	✓	
MAC	(Power Headroom, Msg3)	✓	
RRC	(RRCConnectionReestablishment, Msg3)	✓	OAI eNB Crashed
RRC	(RRCConnectionRequest, After Msg3)	✓	
RRC	(RRCConnectionResumeRequest, After Msg3)	✓	
RLC	(RLC Fragment, After Msg3)	✓	
Protocol	**Flooding Packet Type**	✓	
MAC	Short BSR	✗	
MAC	Long BSR	✗	
MAC	Power Headroom	✓	
RRC	RRCConnectionReestablishment	✓	
RRC	RRCConnectionRequest	✓	
RRC	RRCConnectionResumeRequest	✓	
RLC	RLC Fragment	✓	OAI eNB Hangs

TABLE II
NEGATIVE TEST CASES AND THEIR IMPACT ON OAI 5G *g*NB

Protocol	Malformed Packet Fields	Detected?	Any Consequence?
PDCP	"pdcp-nr.reserved"	✓	
PDCP	"pdcp-nr.direction"	✓	
RLC	"rlc-nr.am.dc"	✓	
RLC	"rlc-nr.seqnum-length"	✗	
RRC	"nr-rrc.dedicatedNAS_Message"	✓	
RRC	ulInformationTransfer_element"	✓	Open5GS Crashed
MAC	"mac-nr.dlsch.lcid"	✓	OAI gNB Crashed
MAC	"mac-nr.subheader.sdu-length"	*N.A*	OAI 5G UE Crashed
NAS	"nas_5g.mm.suci.msin"	✓	
NAS	"nas_5gs.mm.elem.id"	✓	
Protocol	**Replay Scenario (Message, State)**		
RRC	(RRCSetupComplete, Msg3)	✓	OAI gNB Crashed
RRC	(RRCSetupComplete, Connected)	✓	
RRC	(CapabilityInformation, RRCSetupComplete)	✓	
RLC	(Fragment, RRCSetupComplete)	✓	
RLC	(Control Status ACK, RRCSetupComplete)	✓	
NAS	(Auth. Response, RRCSetupComplete)	✗	
NAS	(Auth. Response, SecurityModeComplete)	✓	
MAC	(Short/Long BSR, Msg3)	✓	UE cannot reconnect
Protocol	**Flooding Packet Type**	✓	
RRC	RRCSetupRequest	✓	OAI gNB Crashed
RRC	RRCSetupComplete	✓	
RRC	UE Capability Information	✓	
NAS	Authentication Response	✓	
NAS	Security Mode Complete	✓	
NAS	PDU Session Establishment Req.	✓	
MAC	Short BSR	✗	
MAC	Long BSR	✗	
MAC	Power Headroom	✓	
RLC	Control Status PDU	✓	

nr.subheader.sdu-length", the UE crashes before the packets reach the gNB. Secondly, fields such as *"rlc-nr.seqnum-length"* may be malformed in a given context even if the value of such field falls in a valid range. Since our implementation does not analyse the context (e.g., previous packets) for malformed detection, we may miss attacks that manipulate *"rlc-nr.seqnum-length"* or *"mac-lte.control.bs"*.

Performance: The performance of our framework is measured from the time a packet is intercepted and processed until the packet is released back to the eNB/gNB. The time measurements are shown as a boxplot in Figure 6. Overall, the interception time of our framework makes it suitable for real-time L2 fuzzing in 4G/5G networks. The maximum RTT of $40us$ is well below the latency requirements specified by the *numerology* parameter at the eNB/gNB. Moreover, the decoding speed significantly outperforms recent work [10] that depicts decoding timings between $1-3ms$.

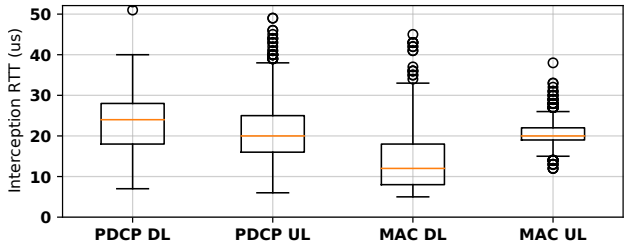

Fig. 6. Downlink/Uplink Interception time for MAC and PDCP. Worst-Case Outlier RTT=$200us$.

V. RELATED WORK

In the last few years, researchers have found a number of design flaws in the 3GPP specification [15], [16]. In contrast to these works, we propose a framework to facilitate automated fuzzing at the implementation level. Current works on discovering implementation level vulnerabilities [2], [17] involve extensive manual effort [17] or the availability of commercial logs [2]. In contrast to these works, our approach constructs the protocol stack machine automatically from a few simple rules. This, in turn, significantly automates the fuzzing framework and packet validation process.

Our approach is inherently more flexible than emulation based fuzzing [18], which requires expert knowledge of the UE hardware architecture. Other notable works include adversarial model testing [19] or mutation and generation based fuzzing of RRC messages [10]. In addition, generation based fuzzing for *only the 4G MAC* has been covered earlier [20]. In contrast to these works, our approach is the first to cover the gap on fuzzing 4G and 5G layer 2 protocols *as a whole* (i.e., PDCP, RLC and MAC) for security testing a black-box UE. Moreover, earlier works do not consider testing encrypted or integrity-protected PDCP packets or downlink RLC fragments.

Alternative approaches such as 5GReasoner [15] aims to discover the vulnerabilities in 5G protocol by modeling the protocol behavior and using verification tools. Such an approach does not target vulnerabilities in real implementation of 5G stack (e.g., 5G UE). In contrast, our proposed framework targets implementation level vulnerabilities.

Recent works have advanced the state of *over-the-air fuzzing* for BLE [21] and Wi-Fi [22]. However, such works do not consider fuzzing fragmented, encrypted and integrity protected packets separately. Instead, such behavior is abstracted during fuzzing. Hence, our work is conceptually different from these works targeting other wireless protocols. Our approach generalizes the state-of-the-art OTA fuzzing for complex 3GPP related protocols [23] by handling fragmentation, encryption and integrity in separate protocol layers as shown in Figure 1.

VI. DISCUSSION

In this paper, we propose a framework to facilitate automated security testing of 4G/5G systems. Compared to existing

works, our proposed framework brings two concrete advantages: *(i)*, our approach is capable to intercept, modify and replay arbitrary packets in both layer 2 and layer 3 communication in real-time. This allows designer to comprehensively stress test different layers. *(ii)* our framework provides capabilities to automatically validate received packets at the gNB. Such a validation is guided by a protocol state machine, which, in turn is learned automatically during normal communication. Consequently, our validation avoids hand-coding of expected responses in communication, thus significantly reducing the manual effort for 4G/5G testing. With low latency fuzzing and automated validation, we hope to significantly improve the state-of-the-art for automated security testing in 4G/5G enabled systems and beyond.

Although our framework is capable to fuzz arbitrary (e.g., commercial) 4G/5G UEs, we acknowledge that more testing and experiments are required for finding new UE vulnerabilities. Nevertheless, our current evaluation demonstrates that the fuzzing framework is capable to trigger crashes in a real base station i.e., OpenAirInterface eNB/gNB. Moreover, our validation process detects the presence or absence of a variety of attacks (e.g., malformed packets, out-of-order packets and flooding) with reasonable accuracy. This provides a platform not only for us to improve our fuzzing algorithms, but also for the community to extend the framework along different dimensions of security through mobile data-link evaluation.

In the future, we plan to improve our fuzzing algorithms, specifically we aim to direct the search process of our fuzzing via efficient heuristics. Such a direction can be formulated as an optimization problem e.g., maximization of protocol state transitions to improve the coverage of protocol states.

Our framework is available open-source upon request.

Acknowledgment: We thank the anonymous reviewers for their insightful comments. This work is partially supported by the Singapore International Graduate Award (SINGA) and NRF National Satellite of Excellence in Trustworthy Software Systems (Project no. NSOE-TSS2021-01 and NSOE-TSS2020-03).

REFERENCES

[1] A. Shaik, R. Borgaonkar, S. Park, and J.-P. Seifert, "New vulnerabilities in 4G and 5G cellular access network protocols: Exposing device capabilities," in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '19. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 221–231.

[2] H. Kim, J. Lee, E. Lee, and Y. Kim, "Touching the Untouchables: Dynamic Security Analysis of the LTE Control Plane," in *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019, pp. 1153–1168.

[3] S. Kanchi, S. Sandilya, D. Bhosale, A. Pitkar, and M. Gondhalekar, "Overview of LTE-A technology," in *2013 IEEE Global High Tech Congress on Electronics*, Nov. 2013, pp. 195–200.

[4] F. Mademann, "The 5G system architecture," *Journal of ICT Standardization*, pp. 77–86, 2018.

[5] M. Zalewski, "American fuzzy lop," https://github.com/google/AFL, April 2017.

[6] V.-T. Pham, M. Böhme, and A. Roychoudhury, "AFLNet: a greybox fuzzer for network protocols," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 460–465.

[7] M. E. Garbelini, C. Wang, and S. Chattopadhyay, "Greyhound: Directed greybox wi-fi fuzzing," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[8] M. E. Garbelini, C. Wang, S. Chattopadhyay, S. Sumei, and E. Kurniawan, "Sweyntooth: Unleashing mayhem over bluetooth low energy," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 911–925. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/garbelini

[9] S. Potnuru, "Fuzzing Radio Resource Control messages in 5G and LTE systems: To test telecommunication systems with ASN. 1 grammar rules based adaptive fuzzer," 2021.

[10] S. Potnuru and P. K. Nakarmi, "Berserker: ASN.1-based Fuzzing of Radio Resource Control Protocol for 4G and 5G," in *2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2021, pp. 295–300.

[11] S. R. Hussain, M. Echeverria, I. Karim, O. Chowdhury, and E. Bertino, "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, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM, 2019, pp. 669–684.

[12] E. Research, "USRP B210 USB Software Defined Radio (SDR)," https://www.ettus.com/all-products/ub210-kit/, 2019.

[13] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 33–38, Oct. 2014.

[14] S.-Y. Lien, S.-L. Shieh, Y. Huang, B. Su, Y.-L. Hsu, and H.-Y. Wei, "5G new radio: Waveform, frame structure, multiple access, and initial access," *IEEE communications magazine*, vol. 55, no. 6, pp. 64–71, 2017.

[15] S. R. Hussain, M. Echeverria, I. Karim, O. Chowdhury, and E. Bertino, "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*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 669–684.

[16] M. J. Lanoue, J. B. Michael, and C. A. Bollmann, "Spoofed Networks: Exploitation of GNSS Security Vulnerability in 4G and 5G Mobile Networks," in *2021 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, Jul. 2021, pp. 1–8.

[17] C. Park, S. Bae, B. Oh, J. Lee, E. Lee, I. Yun, and Y. Kim, "DoLTEst: In-depth Downlink Negative Testing Framework for LTE Devices," in *USENIX Security Symposium*, 2022.

[18] D. Maier, L. Seidel, and S. Park, "BaseSAFE: Baseband sanitized fuzzing through emulation," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '20. New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 122–132.

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

[20] A. Pestrea, "Fuzz testing on eNodeB over the air interface: Using fuzz testing as a means of testing security," 2021.

[21] M. E. Garbelini, C. Wang, S. Chattopadhyay, S. Sumei, and E. Kurniawan, "SweynTooth: Unleashing Mayhem over Bluetooth Low Energy," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 911–925.

[22] M. E. Garbelini, C. Wang, and S. Chattopadhyay, "Greyhound: Directed greybox wi-fi fuzzing," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[23] A. A. Atayero, M. K. Luka, M. K. Orya, and J. O. Iruemi, "3GPP long term evolution: Architecture, protocols and interfaces," *International Journal of Information and Communication Technology Research*, vol. 1, no. 7, pp. 306–310, 2011.