Compositional Testing of S1AP protocol in Cellular Communication Networks

Prasanth Prahladan and Taeho Kim

Motivation

- Formal specification and testing of QUIC paper
  - verify a complex stateful internet-protocol QUIC using the IVy tool

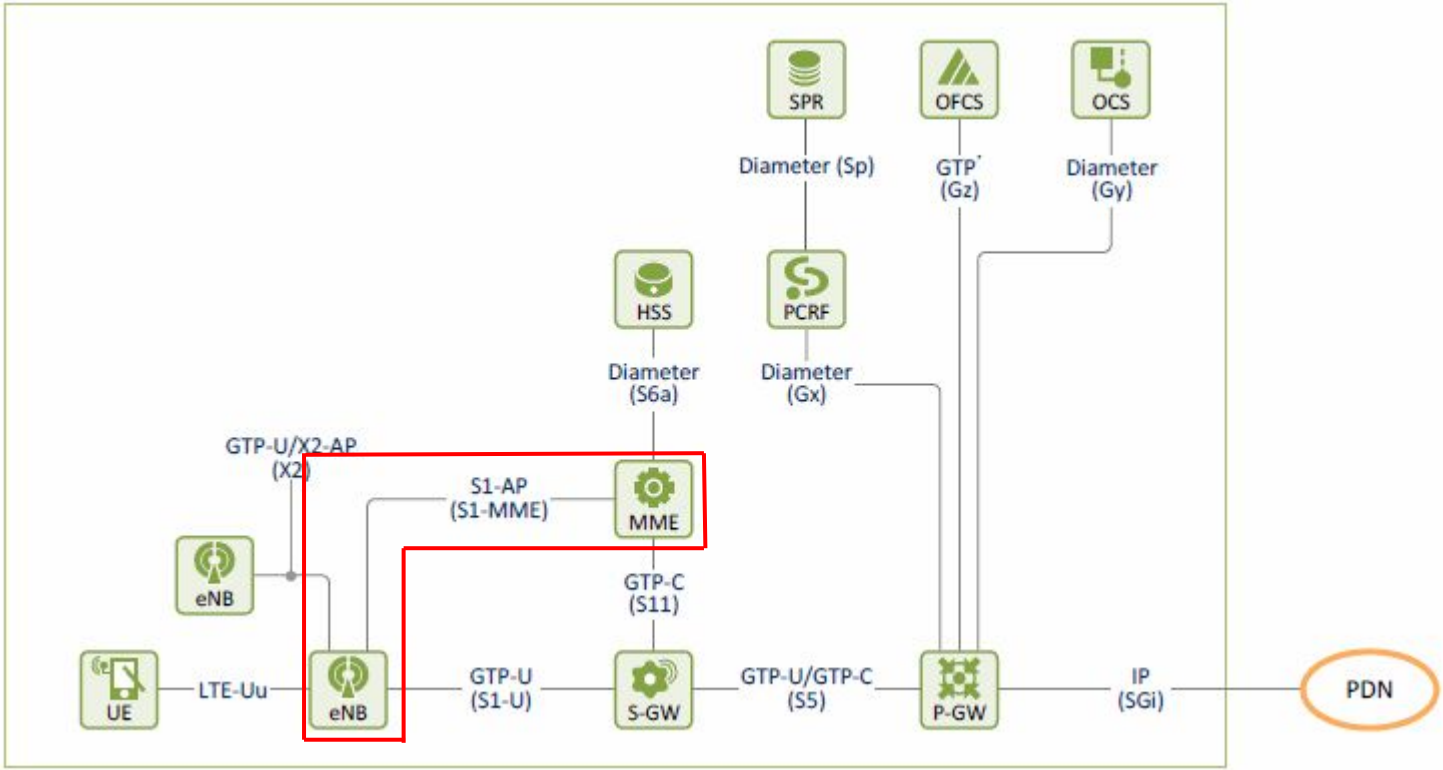
- Goal
  - extend the application of the formal verification of Internet-protocols to the domain of cellular communication networks
  - prove a formal specification of the S1AP protocol using the IVy tool

Challenges

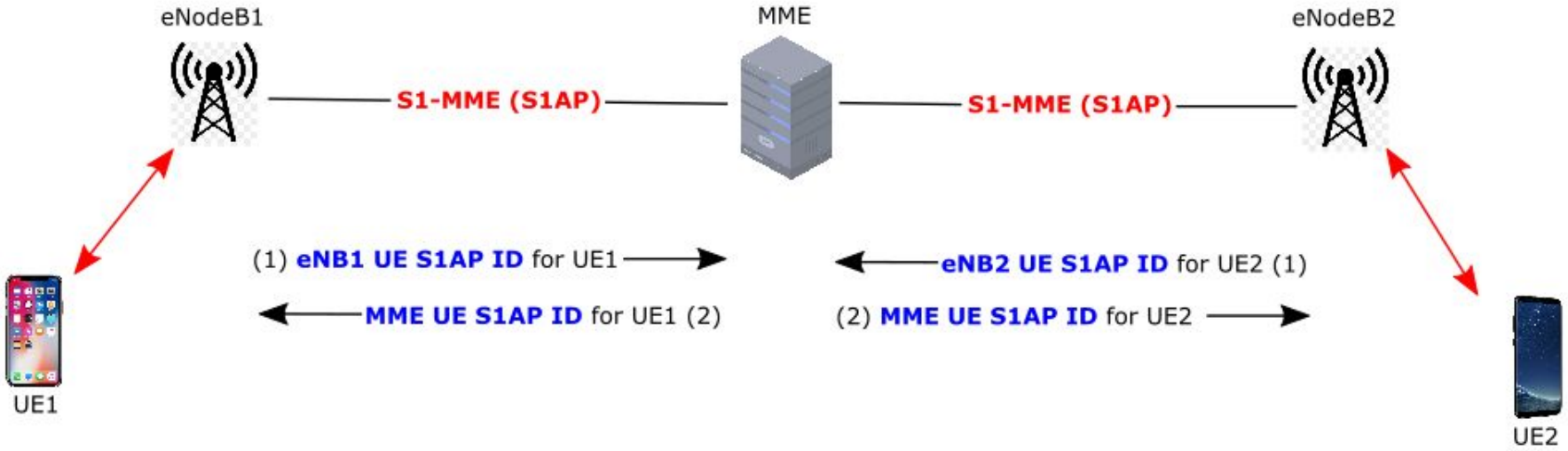
1) Cellular networks are closed system - not readily accessible
    a) SCTP protocol implementation
    b) Serializer/Deserializer implementation

2) Patterns of inter-protocol communication - diverse and complex
    a) Multiple types and messages definition
    b) Test codes for S1AP

LTE network architecture



S1AP protocol



Why do we need formal specifications?

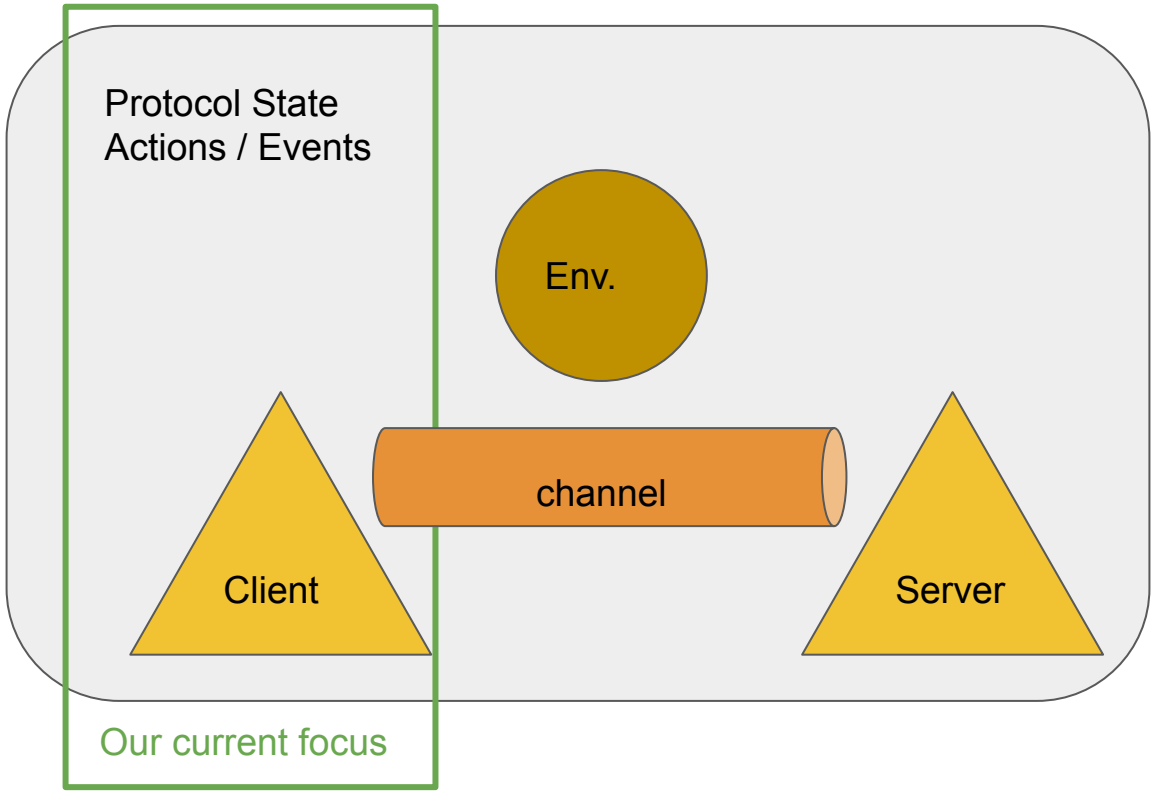
1. Knowledge implicit in implementations
2. Any implementation, is one realization of the protocol. It might only capture a subset of the behaviours permitted by the protocol.
3. Actual protocol-compliance is never tested.

What does it mean to specify a protocol?

1. Behaviours of the protocol, based on what's observable (on the wire)
2. Behaviours of the protocol, based on what we expect happens internal to each constituent sub-system.
3. Properties of the system: Safety, Liveness

De-facto standard: Implementations in the wild!

Entity representation within Ivy

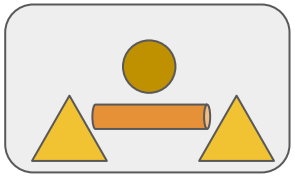


Features:

1. Comopositional
2. No FSM assumption
3. Global Protocol specification based on “Observables”.
4. Generation of testing input: Automated Test-Generation Via constraint solving
5. Checkers of outputs of processes

Method-1

Method-2
Compositional Testing



Application Layer(S1AP)

Transport Layer(SCTP)

Network Layer(IP)

Device Layer

Ivy based verification

Client

Protocol specification
Client (eNB)

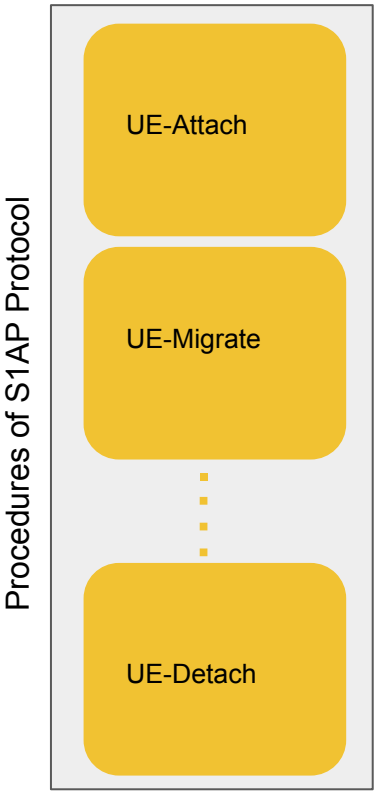
SCTP - socket integration
(Ivy + C++)

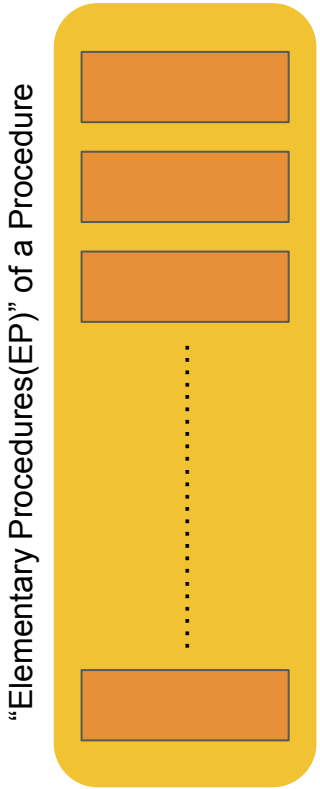
Serializer/Deserializer

Server

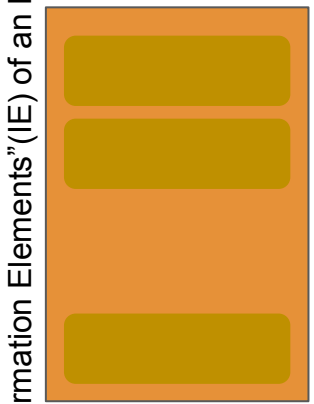
Implementation
(Test-target)
(MME)

Protocol Message Architecture

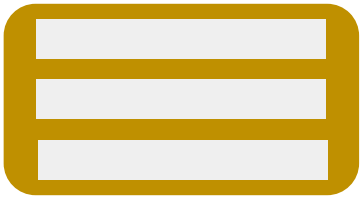




Packet



Frame



Components of an IE

Constructing Serializer/Deserializer

Serializer/Deserializer

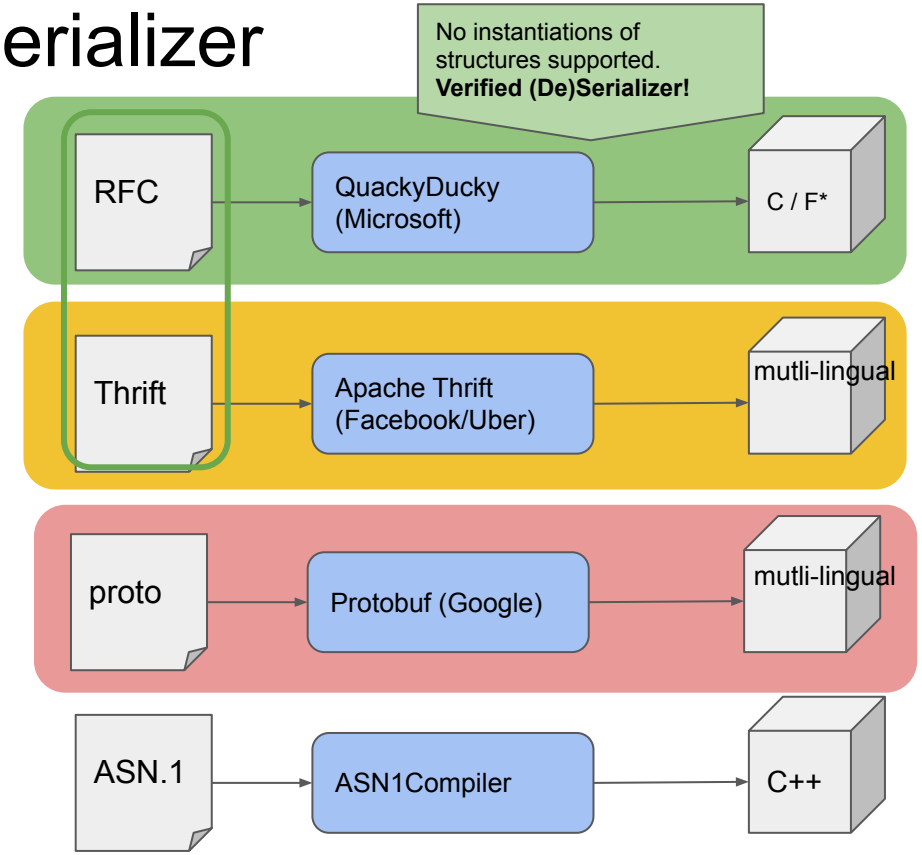
Transport Layer(SCTP)

Network Layer(IP)

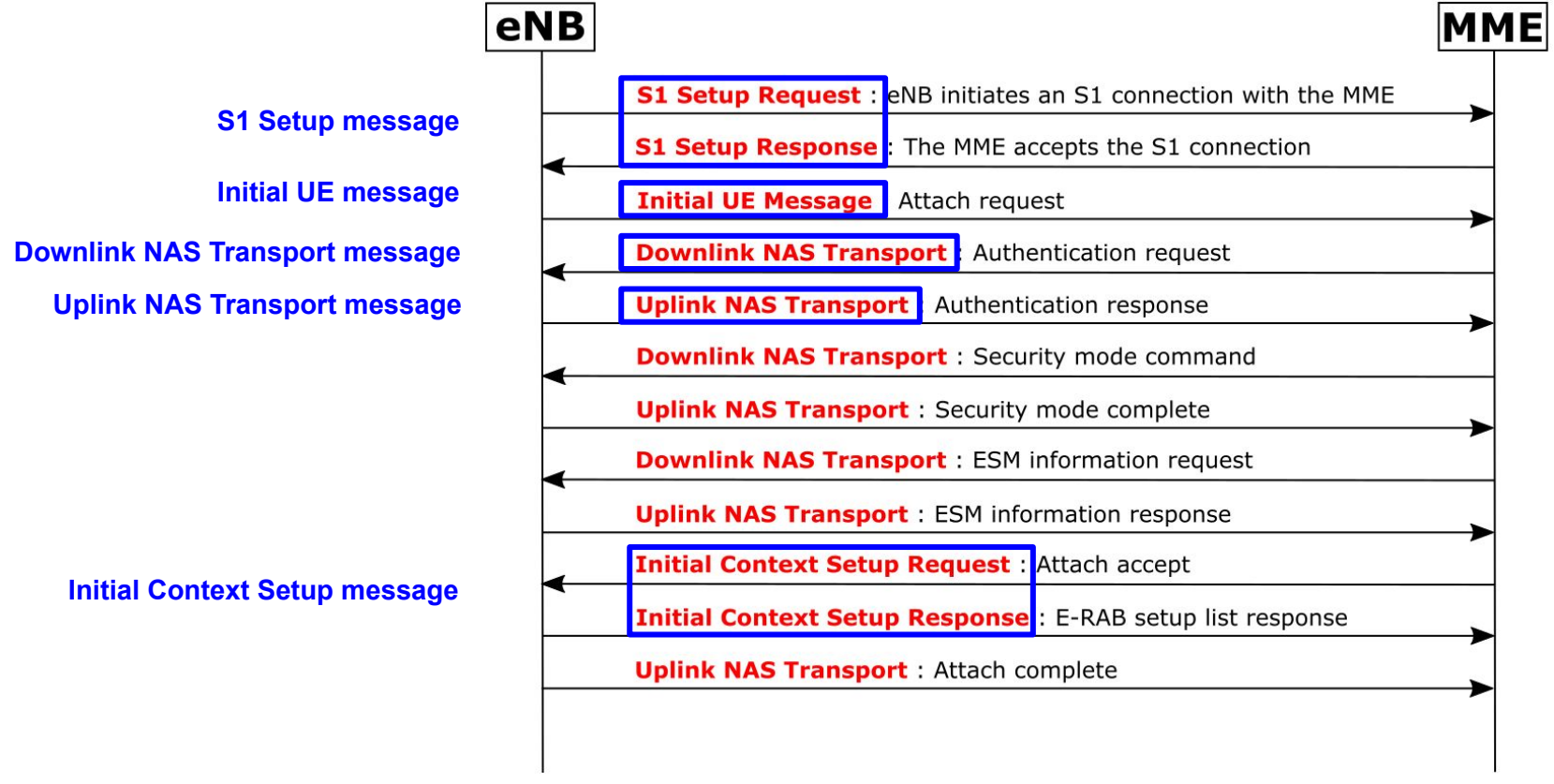
Device Layer

Most versatile in terms of flexibility of language and features.
Has support for Transport Layer of stack.

No **uint8.**
Doesn't support Ordering of message sub-structures.



Initial UE attach procedure



Type definition

- As an example, we'll look at the types of IEs for the S1 Setup message

Packet state

```
type cid
```

Procedure code

```
type pcode_bits
interpret pcode_bits -> bv[8]
```

Tracking area code

```
type tac_octet
interpret tac_octet -> bv[16]
```

Public land mobile network

```
type plmn_octet
interpret plmn_octet -> bv[24]
```

Type of message

```
object message_enum = {
    type this = {initial_message, successful_outcome, unsuccessful_outcome}
}
```

Discontinuous reception

```
object drx_enum = {
    type this = {0, 32, 64, 128, 256} # 0 is error
}
```

Supported tracking area array

```
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)
}
```

Protocol state and Mandatory IEs

- Protocol state transition by action event

```
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 state returns(s:cid) = {
    s := packet_state
}
```

- Checking the values of mandatory IEs

```
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
}
```

S1 Setup test execution

```
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
>
```

Conclusion

Current contributions:

1. Specified a simple manual testing scheme, using Ivy, that can:
   a. Detect the correctness of the received response
   b. Update simplified notion of state of the protocol
2. Specification of the protocol message-format in easily accessible formats: RFC and Apache Thrift

Main task remaining:

1. Specify protocol to enable automated test-generation
2. Develop shim that interfaces the protocol-specification with network interface.

Goals

1. Prepare a formal specification of the protocol
    a. Encode knowledge that's implicit in implementations
    b. Documentation: Keep track of updates introduced in newer versions of protocol
    c. Execution: Automated extraction of test-generators and message-checkers
        i. Test compliance to a common standard
        ii. Test in an adversarial environment
    d. Function: Expose errors via testing
2. Make available the protocol message format specifications in a format that's easily digestible by current web-technologies.
3. Obtain meaningful results from testing:
    a. Test implementations to identify failure to conform/comply with the protocol-specification.
    b. Identify bugs in implementations, or in the specification itself.
    c. Identify system security vulnerabilities

Implementation Details

Verification of the protocol in the abstract

Why is this not truly feasible?

- too many sub-components(analogous to the layers of the stack for QUIC)
-

What have we done? (Taeho - pls describe this? - Done)

- Our test program captures the sent message with mandatory elements.
- Tested specification in the abstract for each message-type individually.

What have we NOT done?

- To conduct the tests for the entire sequence of messages that’s exchanged

Delete Me: Layout of the presentation

1. LTE Architecture
2. What are the challenges/demands of testing?
3. Why can't we do formal verification of systems, and why do we try to handle this as a problem for testing?
    a. State-space explosion
    b. Distributed software-components with diversity of participating entities.
    c. Entire system is complex, due to layering of protocols between entities
4. Tasks involved:
    a. Modelling the protocol in Ivy and Verifying it abstractly (Taeho)
    b. Creating parsers/serializers for transmitting test messages to a physical device on the network (SCTP,
5. Specification Languages:
    a. Protocol Specification for abstract model: Ivy
    b. Message and Format Specification: RFC format; Thrift; ASN.1
6. Modelling the protocol : state and transitions
7. Developing the serializer/deserializer: Describing the messages
8. Experiments we intended to do:
    a. Abstract:
        i. Unlike QUIC, the S1AP protocol has a asymmetric entities at either end of the channel. The server responds via the protocol, but the state-changes depends on its interaction with other entities, using other protocols.
        ii. We model the protocol with a primary focus on the correctness of the client-portion, while assuming a simplified version of the server.
        iii. [Taeho's notes]
    b. On-the-wire:
        i. The Serializer/Deserializer once generated would be used to transmit the randomly generated test-messages, which are compliant with the S1AP protocol, to an Open-source implementation of an EPC like (NextEPC, OpenEPC), etc.
        ii. Develop a Message specification RFC format and Thrift, that's open-source, and available for the community to understand the messages

Contributions

Protocol modelling and verification:

- Ivy based representation of control-plane protocol-dynamics
- explicit description of the state-of-protocol
- explicit description of assume-guarantee relationships between protocol-events
- protocol-specification that's focused on observables, rather than internal-operations of the sub-components.

Message Specification and Formats:

- Message format specification for S1AP protocol using Thrift and RFC
- Generated serializers/deserializers for Control-Plane protocol(s1ap) in cellular-communication network

What does it mean to specify a protocol?

1) What are the messages/formats?
2) What are the sequence of messages that are exchanged between the entities? When is a behaviour considered to be faulty?
3) Does the channel(network) play any role? Can it be an adversary? Can it generate new messages?

Presentation Main Flow

Aim: Why should someone read our paper? What's the motivation of the problem we're trying to address?

1. How have the PL-formalisms we've learned in class, helped for this project?
    a. Taeho: can you describe how you've designed the Assume-Guarantee conditions for the different messages? This way you can connect it with the Hoare-logic based axiomatic approach to verifying software.
    b. {A}c{B} :: {Precondition(Assume)} protocol-event {Post-condition(Guarantee)}
2. Where is "testing" situated, within the context of verification of system performance? How does Compositional Testing help us?
    a. Each sub-component of the system can be modelled separately.
    b. If a system shows a fault, it means some sub-component is necessarily responsible for inducing the fault.
    c. Specifying the protocol in Ivy, signifies the Assume-Guarantee relationship between events. It helps in specifying how events are related to each other