CellScope: Automatically Specifying and Verifying Cellular Network Protocols

WUHAN UNIVERSITY

Yinbo Yu, You Li, Kaiyu Hou, Yan Chen, Hai Zhou, Jianfeng Yang

Northwestern University

Introduction

Towards a secure cellular network, researchers are spending efforts to manually identify vulnerabilities (*human investigation*). Such a practice demands both time and patience. It is only suitable to be applied on a few procedures of large cellular network protocols. *Dynamic testing*, on the other hand, invokes test cases to check if the actual network behavior meets security criteria. It identifies vulnerabilities during execution. This approach, however, depends on the quality of input test cases. As a result, its completeness can never be proved.

The use of formal methods brings a systematic and solid approach to cellular network security research. Nevertheless, researchers have encountered a common and critical problem: *specification*. Protocols of cellular network are documented in natural languages. Human efforts are required to convert them into formal models. Such *manual-crafted specifications* are error-prone and only capable of describing small pieces of protocols.

Table 1: Existing methods vs. CellScope.

Methodology	Generalizable	Trustworthy	Complete	Automated
Human investigation	○	●	○	○
Dynamic testing	◐	◐	◐	●
Manual specification	◐	◐	◐	◐
CellScope	●	◐	●	●

Being different from existing methods, we present CellScope, an automated framework for specifying and verifying cellular network protocols based on software model checking. With CellScope, we aim to efficiently inspects cellular network protocols in large scale.

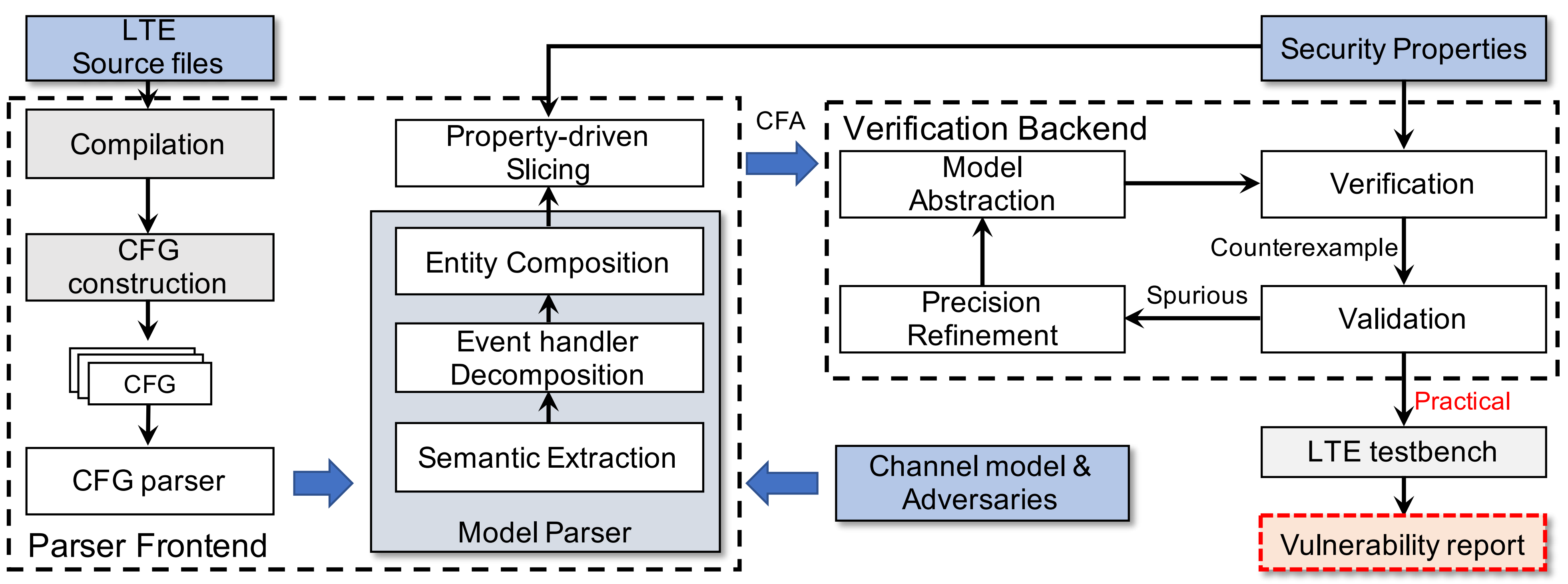
Challenge

- **Size Explosion**: The state space in an implementation of cellular network is too large for a model checker. For instance, a prevailing open source implementation of LTE, OpenAirInterface (OAI), has more than one million lines of code.
- **Independent software entity**: There are multiple software entities (including User Equipment, eNodeB, and the Core Network) run independently in a cellular network.
- **Multi Agent Interaction**: In the cellular network, each of the entity is driven by messages sent by each other. Such a model is neither a sequential program, nor a classical multi-thread program.
- **Temporal Logic Checking**: Software model checkers focus on practical implementation issues. To ensure the correctness of cellular network protocols, more complex properties with temporal logic is needed. For example, we need to ensure that if a cell phone requests to attach to the core network, it will succeed eventually.

CellScope

CellScope automatically extracts control flow graph (CFG) from software implementations of cellular network. We do optimization on CFGs for model simplification and adversary injection. By applying improved counter-example-guided abstraction refinement (CEGAR), CellScope inspects cellular network protocols in large scale and validates identified counterexample in a real cellular network testbench.

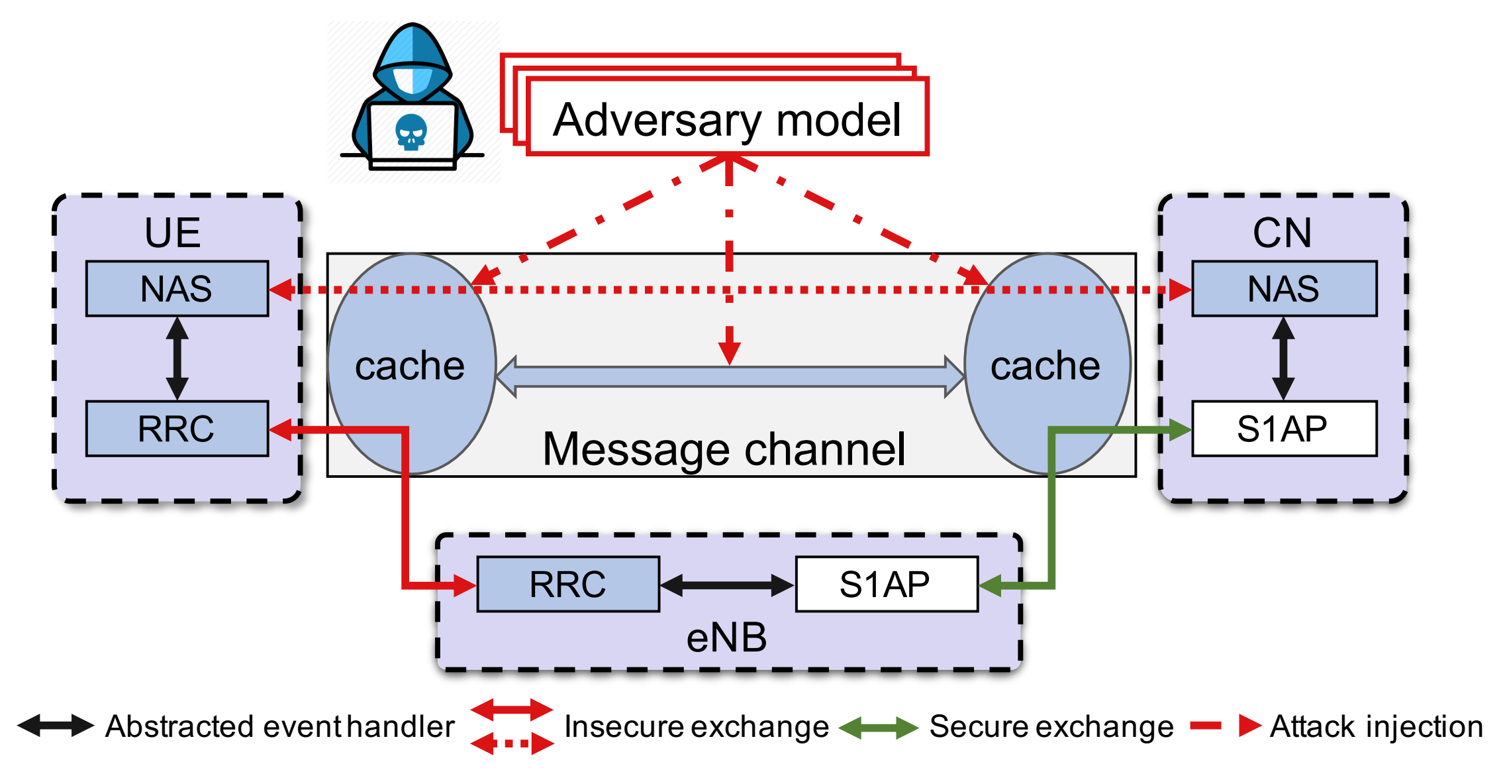
Architecture



The architecture of CellScope:

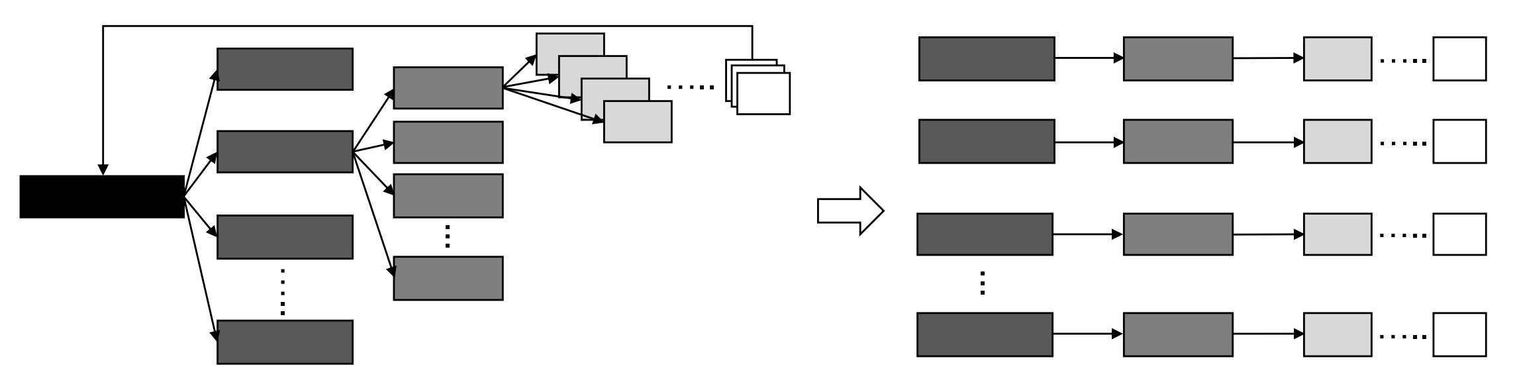
1. **Parser Frontend**: It automatically translates LTE source codes to formal models and instruments channel and adversary models into LTE models. By slicing the model depending on a property, it extracts a small but sufficient model for verification with the property.
2. **Verification Backend**: we design a prioritized counterexample guided abstractionrefinement (P-CEGAR) verification and model decomposition with weakest precondition. With this two techniques, CellScope efficiently inspects cellular network protocols in large scale. Once a counterexample is identified, it will be validated on a real OAI testbed platform.

Message Channel



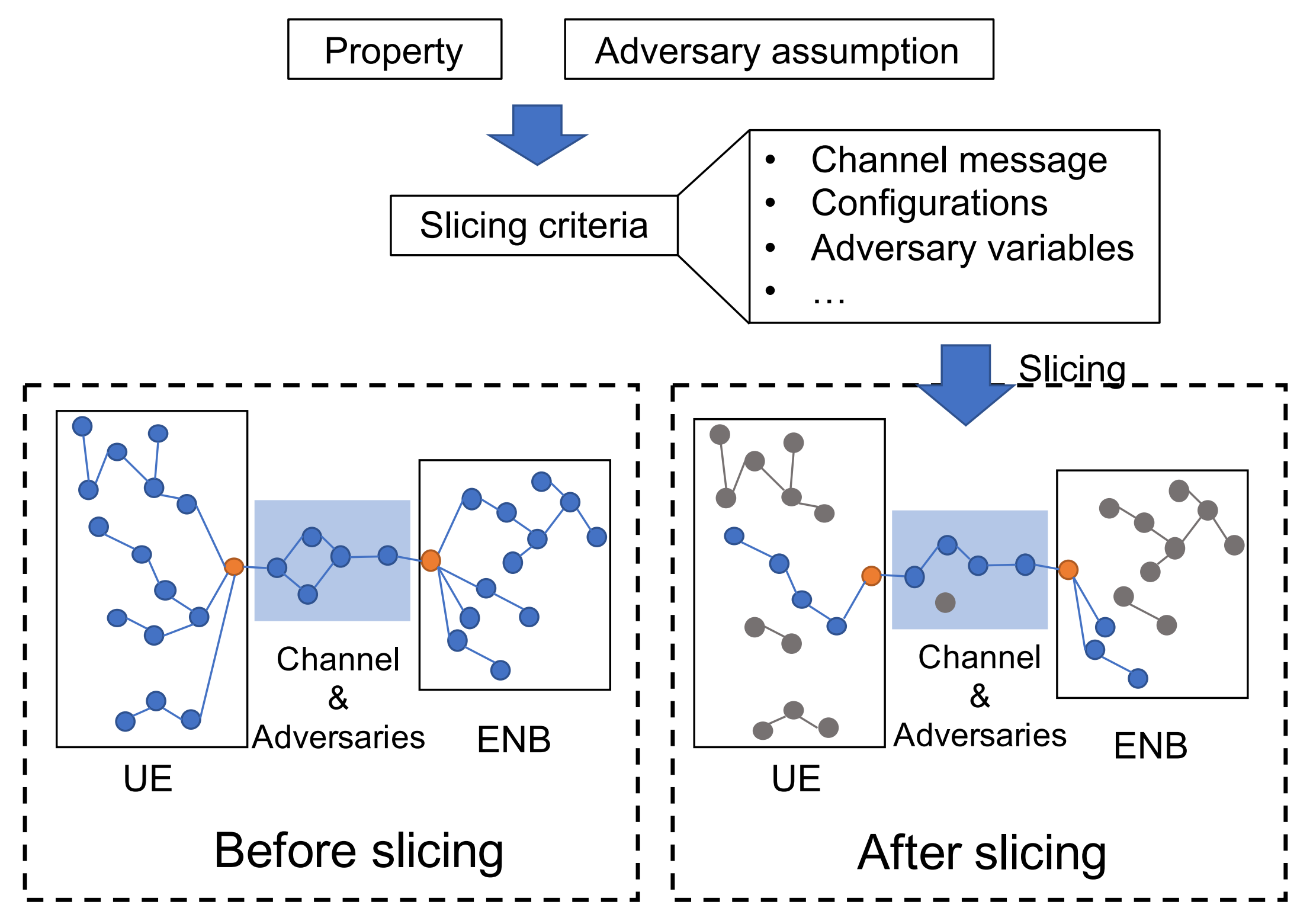
1. Mock up program behaviors in low layers;
2. Compose independent software entities with formal message exchange models;
3. Enable the injection of Dolev-Yao-style adversaries into commmunication channel;

Decomposing Event Handler

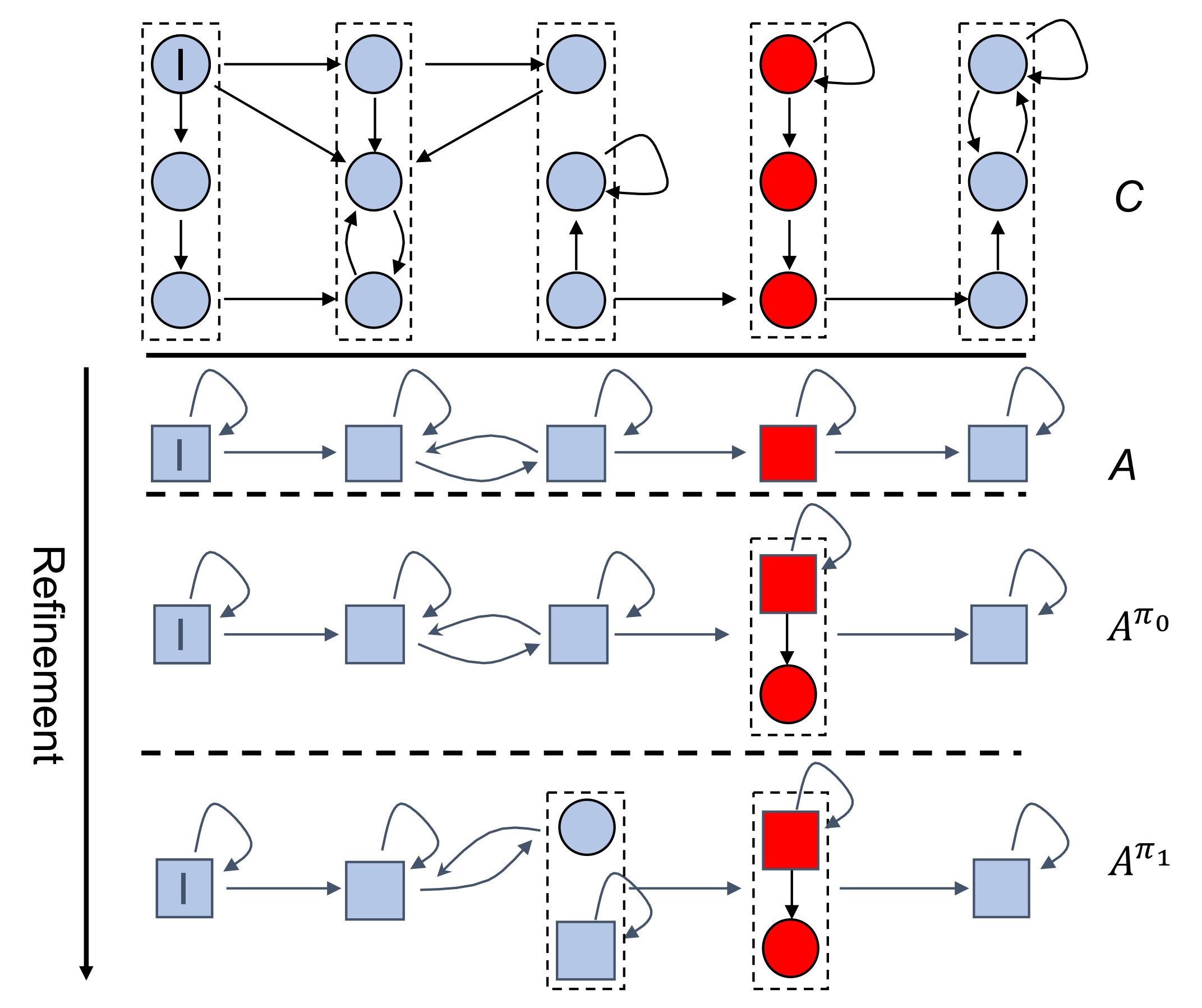


In the cellular network, there are multi-layer event handlers to process messages from different entities, network layers, protocols, components, or entity statuses. In software, these handlers are implemented in a comprehensive manner to cover all possibilities. These implementations, however, expose a large but unnecessary state space for model checking.

Property-driven Slicing



P-CEGAR



Prilimary Results

Vulnerability	Adversary	Attack	Protocol	Root cause	New attack?
No EPS services	Malicious eNB	DoS	NAS	Malicious *attach_reject*	Known
Forbidding PLMNs	Malicious eNB	DoS	NAS	Malicious *attach_reject* with #11 or 14 cause	Yes
Forbidding TAIs	Malicious eNB	DoS	NAS	Malicious *attach_reject* with #12, 13 or 15 cause	Yes
Barring cells	Malicious eNB	DoS	RRC	Malicious *SIB1* with a *cellBarred* flag	Yes