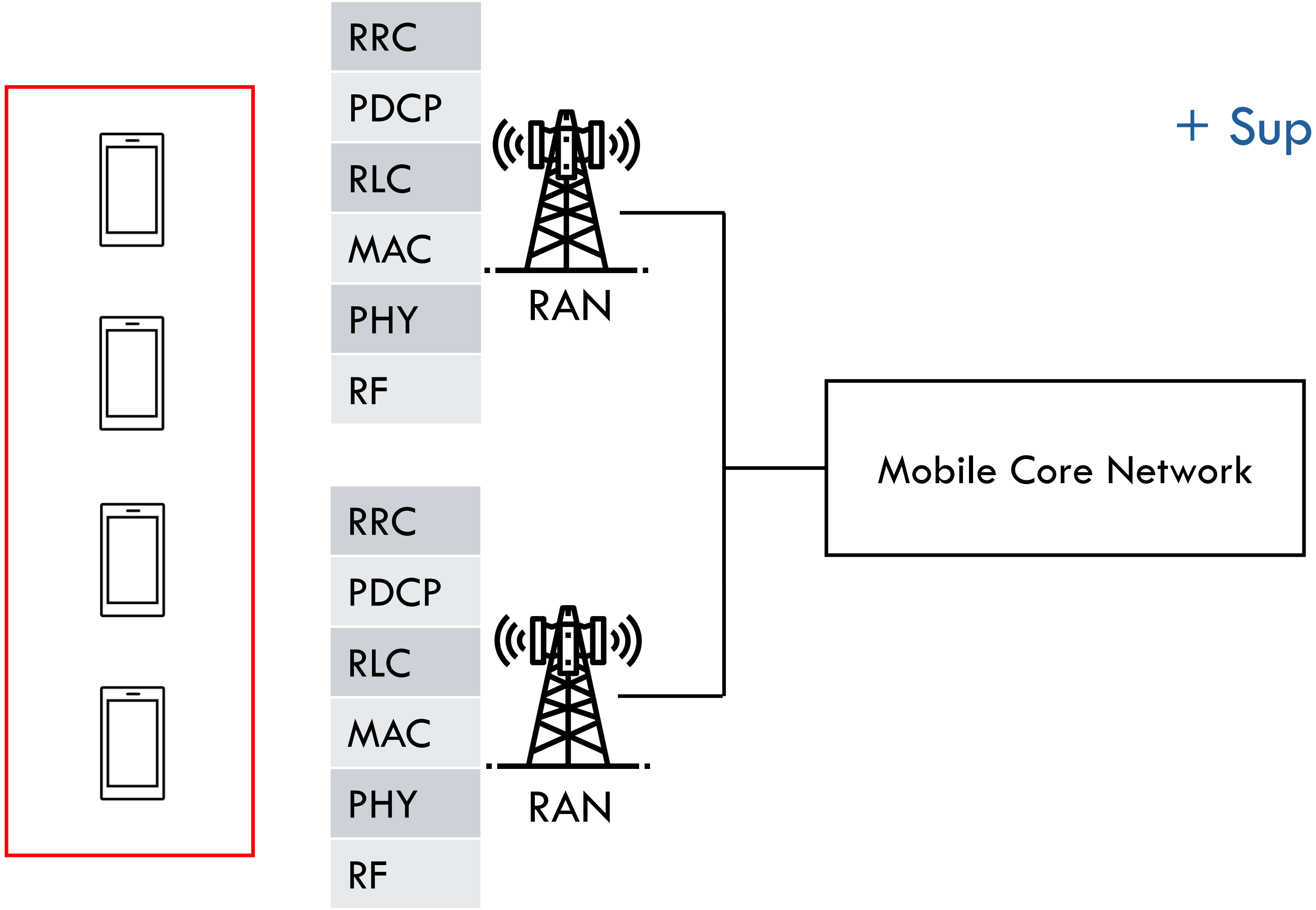
ORANalyst: Systematic Testing Framework for Open RAN Implementations

Tianchang Yang, Syed Md Mukit Rashid, Ali Ranjbar, Gang Tan, Syed Rafiul Hussain

Systems and Network Security (SyNSec) Lab
Department of Computer Science and Engineering
The Pennsylvania State University

Mobile Network's Transition to 5G



RRC
PDCP
RLC
MAC
PHY
RF

RAN

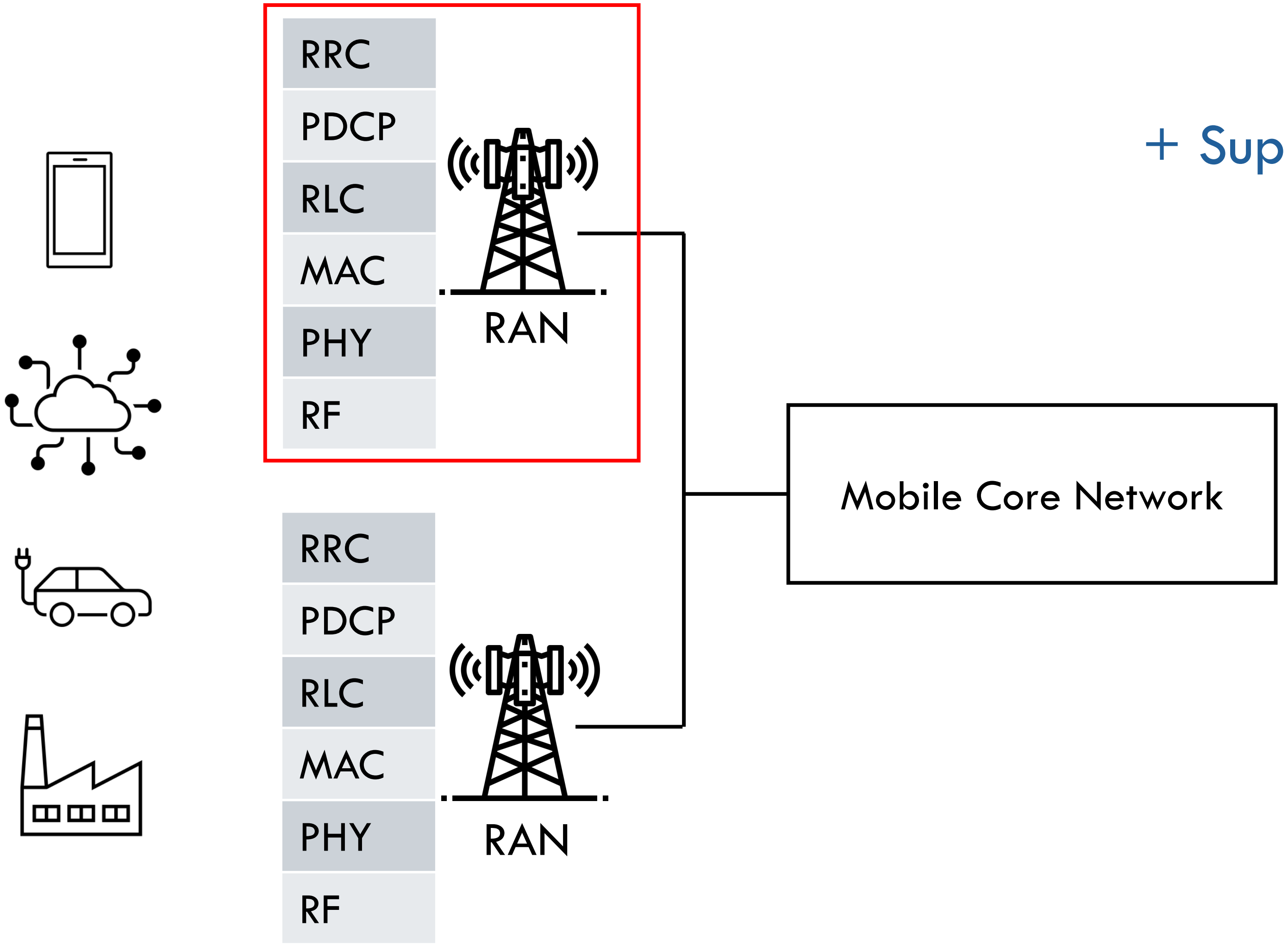
RRC
PDCP
RLC
MAC
PHY
RF

RAN

Mobile Core Network

+ Support diverse devices and use cases

Mobile Network's Transition to 5G



RRC
PDCP
RLC
MAC
PHY
RF

RAN

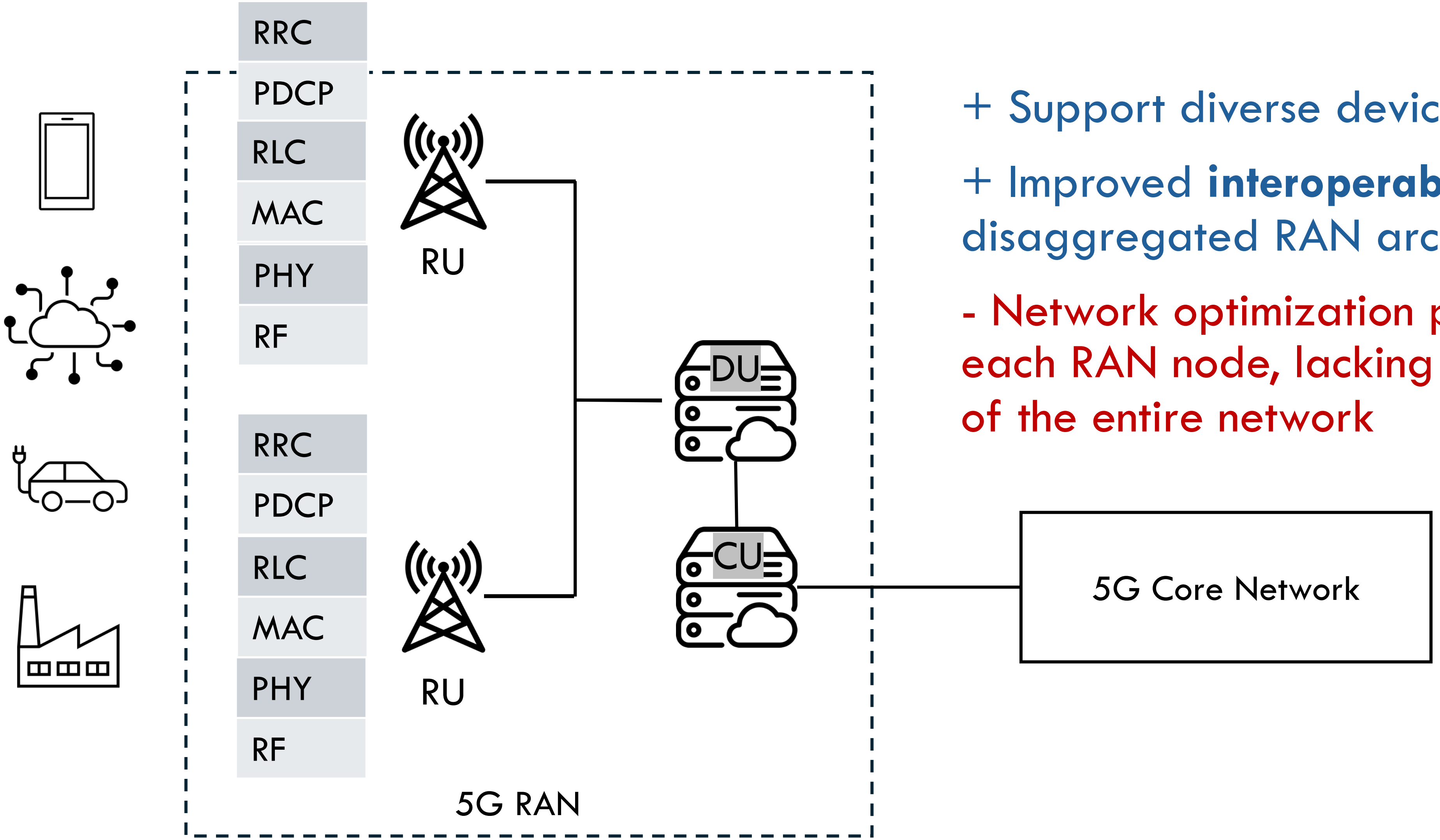
RRC
PDCP
RLC
MAC
PHY
RF

RAN

Mobile Core Network

+ Support diverse devices and use cases

Mobile Network's Transition to 5G

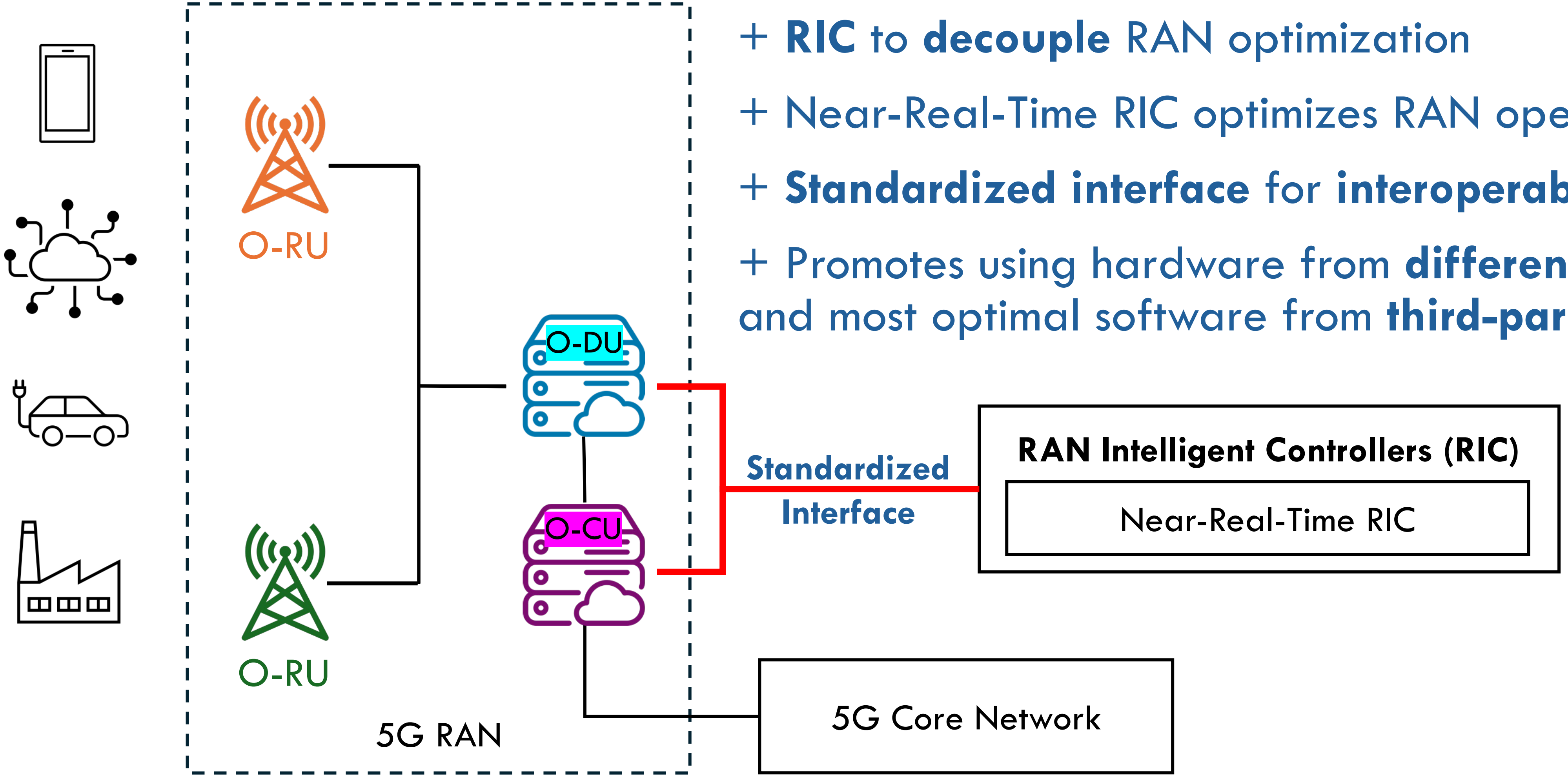


+ Support diverse devices and use cases

+ Improved **interoperability** through disaggregated RAN architecture

- Network optimization performed at each RAN node, lacking high-level view of the entire network

Open RAN



+ **RIC** to **decouple** RAN optimization

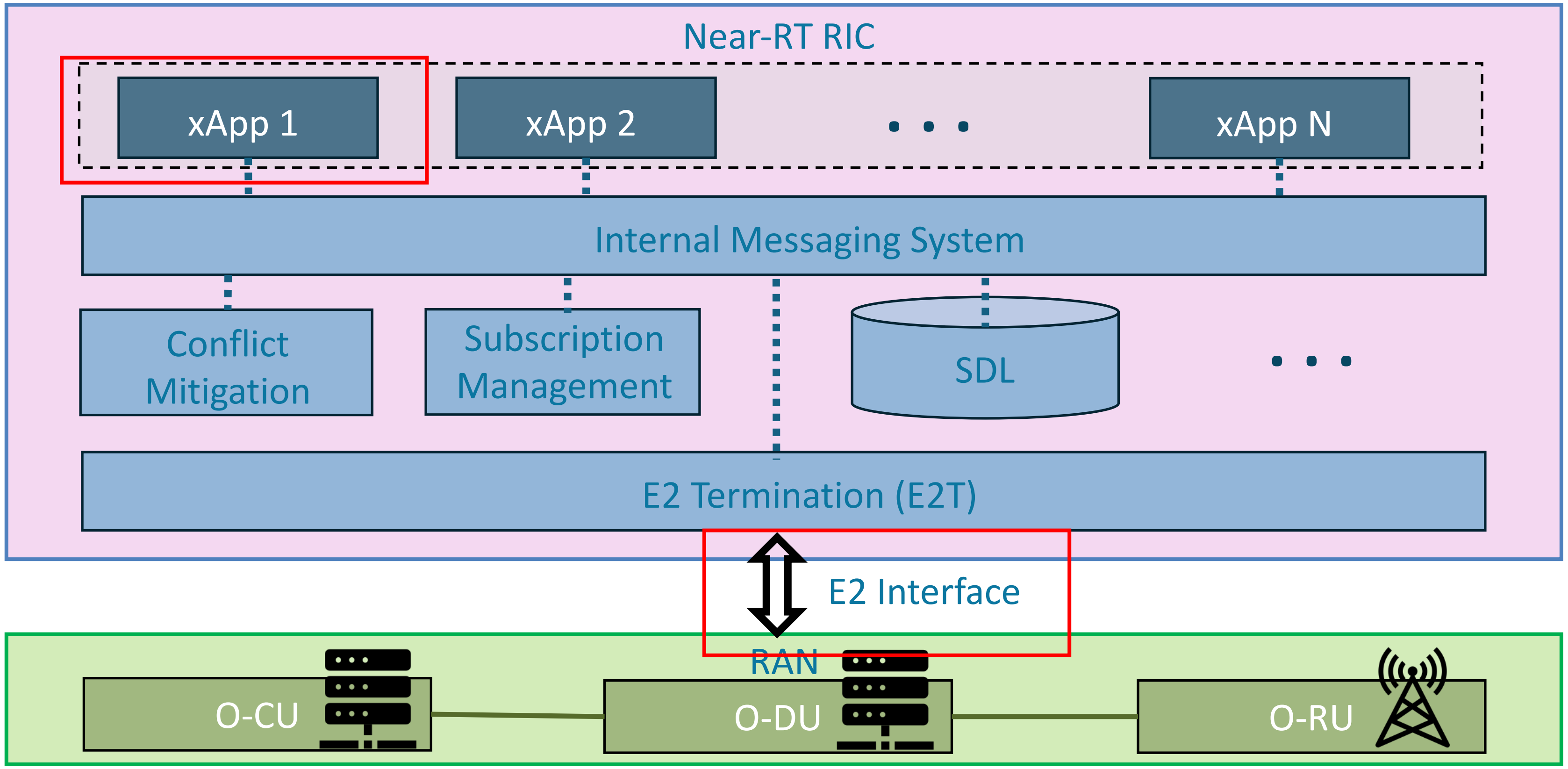
+ Near-Real-Time RIC optimizes RAN operations

+ **Standardized interface** for **interoperability**

+ Promotes using hardware from **different vendors,** and most optimal software from **third-parties**

RAN Intelligent Controller (RIC) Architecture

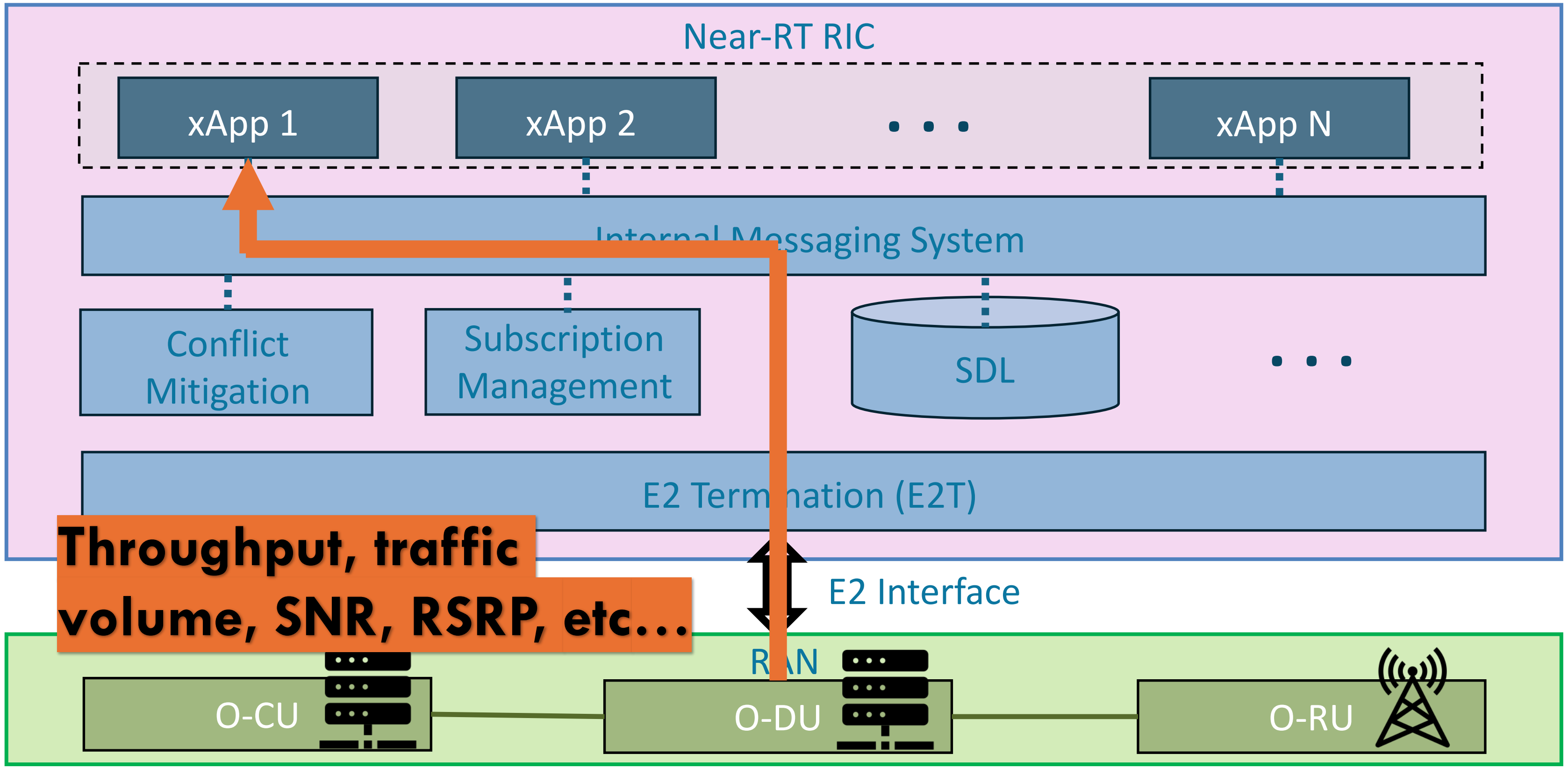
Traffic steering, power optimization, network slice management …



Service-Based Architecture

RAN Intelligent Controller (RIC) Architecture

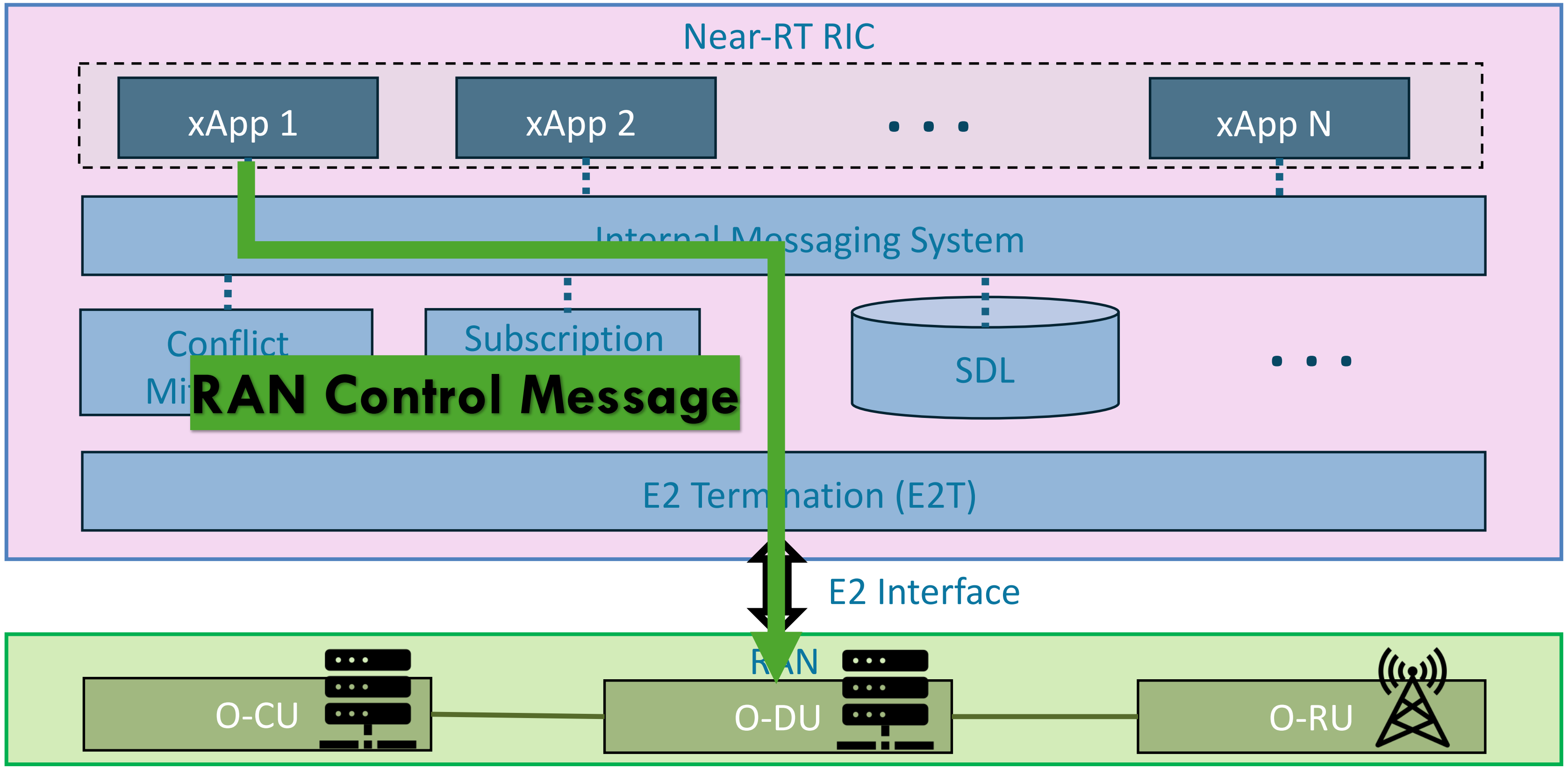
Traffic steering, power optimization, network slice management …



Service-Based Architecture

RAN Intelligent Controller (RIC) Architecture

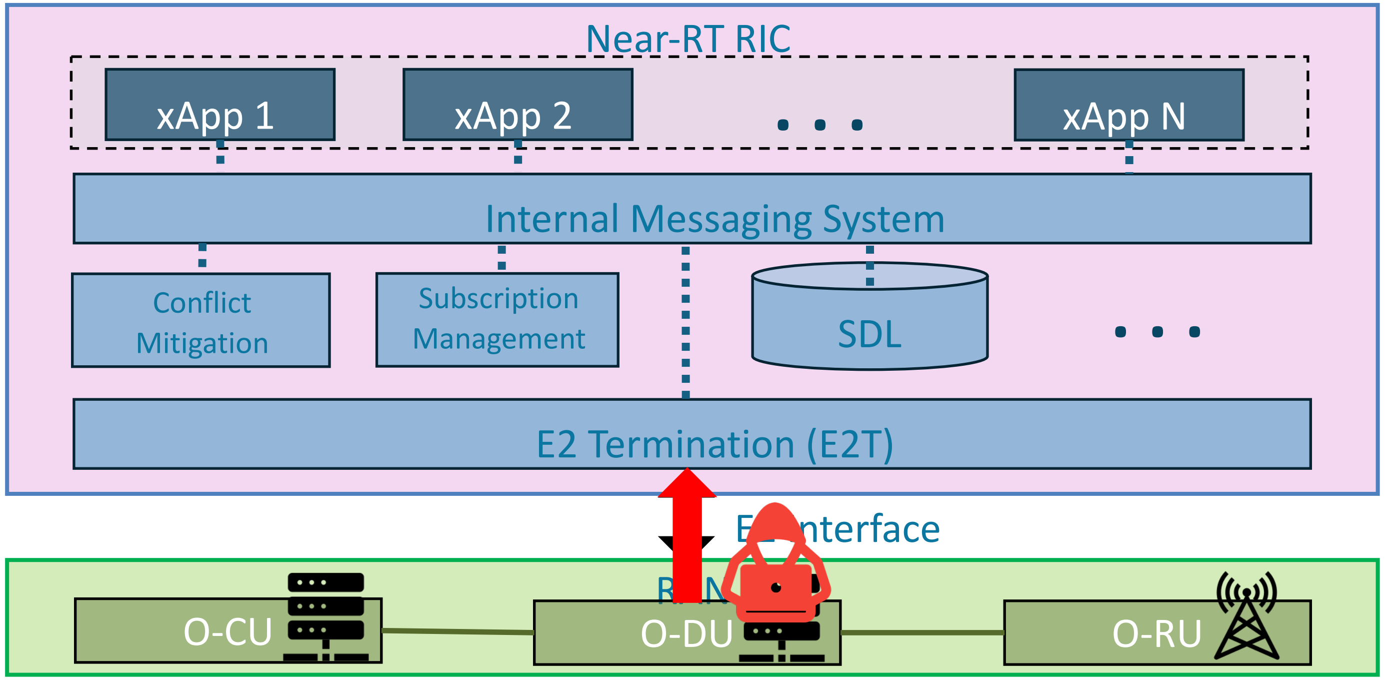
Traffic steering, power optimization, network slice management …



Service-Based Architecture

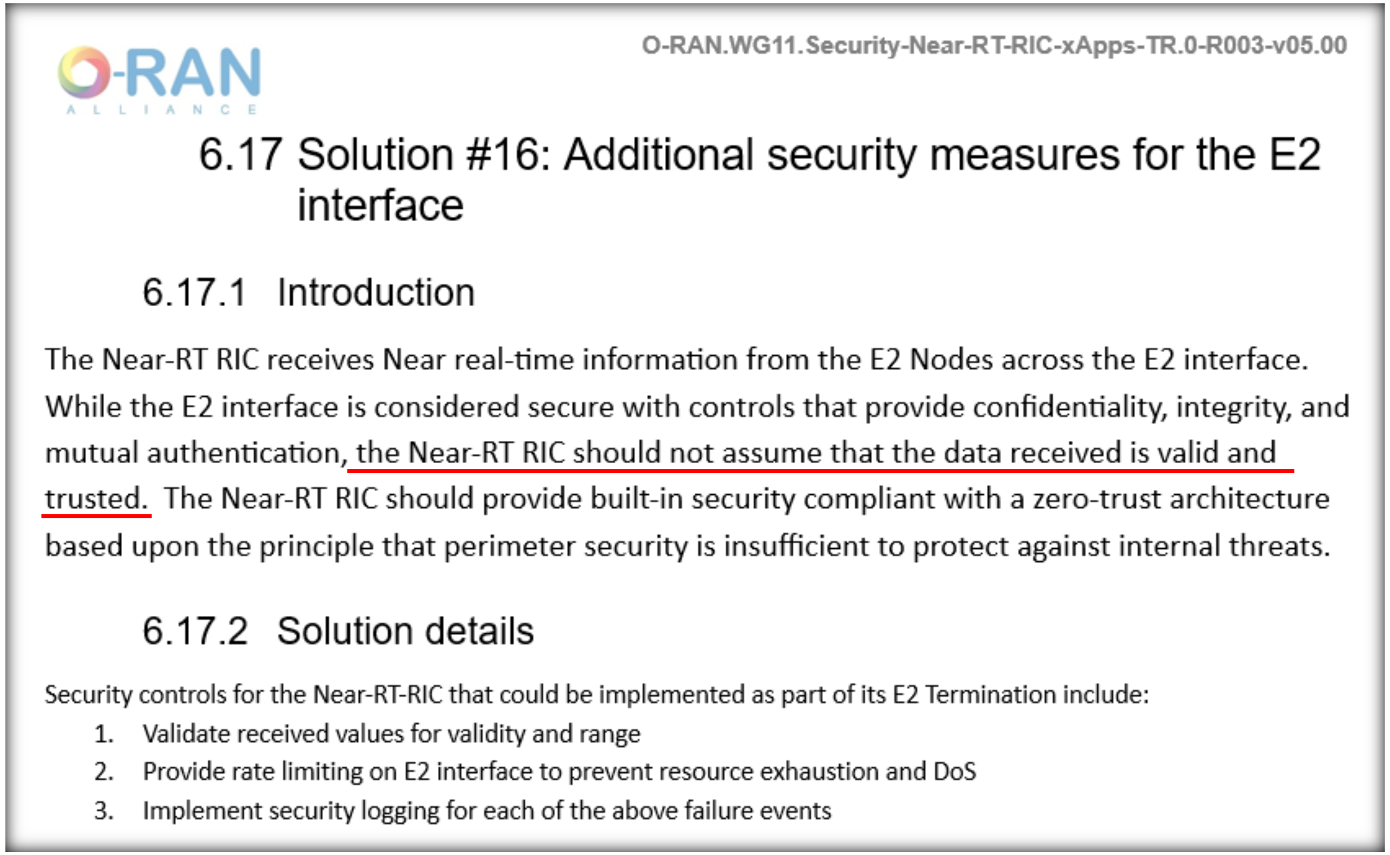
Attack Surface of O-RAN RIC

- **Software-centric** RIC with **third-party providers**
  - More likely to contain software bugs/vulnerabilities
  - Misconfiguration, dependency vulnerability, insufficient checks
- **Heterogeneous** RAN nodes & user devices
  - RIC faces unpredictable, possible malicious data
  - Unexpected/unsanitized traffic from RAN node, malicious UE behavior



Attack Surface of O-RAN RIC

- **Software-centric** RIC with **third-party providers**
  - More likely to contain software bugs/vulnerabilities
  - Misconfiguration, dependency vulnerability, insufficient checks
- **Heterogeneous** RAN nodes & user devices
  - RIC faces unpredictable, possible malicious data
  - Unexpected/unsanitized traffic from RAN node, malicious UE behavior



O-RAN ALLIANCE

O-RAN.WG11.Security-Near-RT-RIC-xApps-TR.0-R003-v05.00

6.17 Solution #16: Additional security measures for the E2 interface

6.17.1 Introduction

The Near-RT RIC receives Near real-time information from the E2 Nodes across the E2 interface. While the E2 interface is considered secure with controls that provide confidentiality, integrity, and mutual authentication, the Near-RT RIC should not assume that the data received is valid and trusted. The Near-RT RIC should provide built-in security compliant with a zero-trust architecture based upon the principle that perimeter security is insufficient to protect against internal threats.

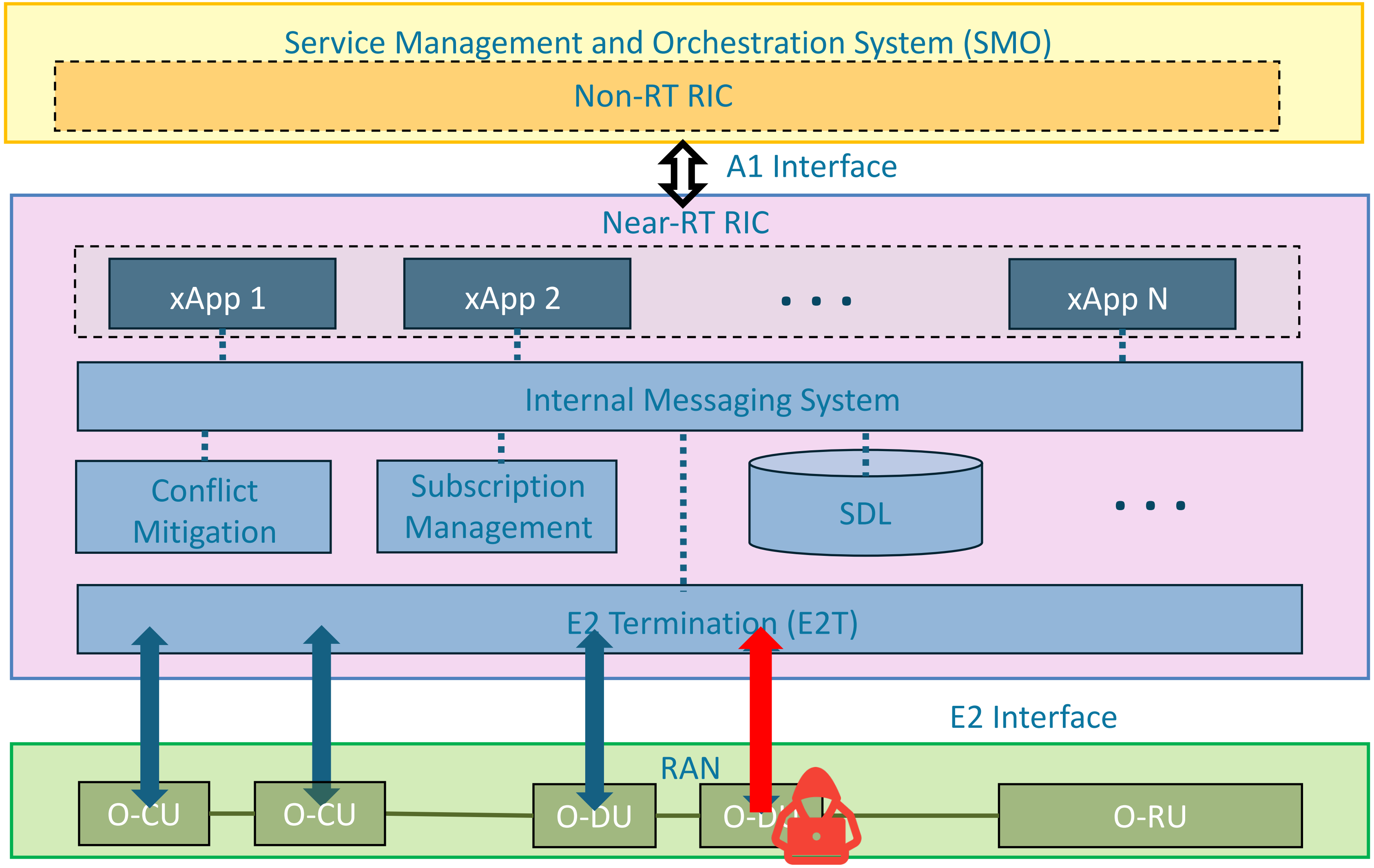
6.17.2 Solution details

Security controls for the Near-RT-RIC that could be implemented as part of its E2 Termination include:

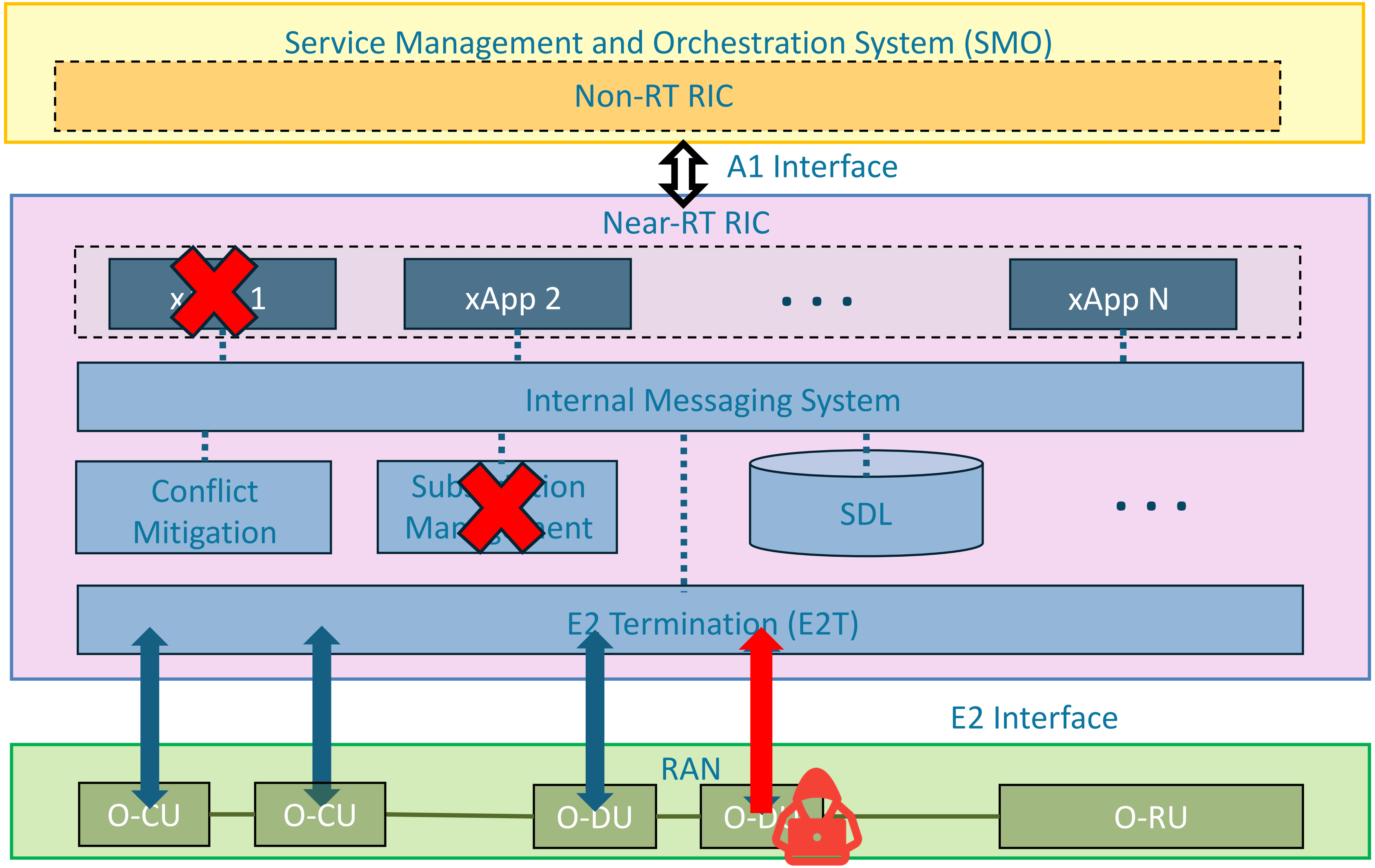
1. Validate received values for validity and range
2. Provide rate limiting on E2 interface to prevent resource exhaustion and DoS
3. Implement security logging for each of the above failure events

O-RAN Study on Security for Near Real Time RIC and xApps 5.0

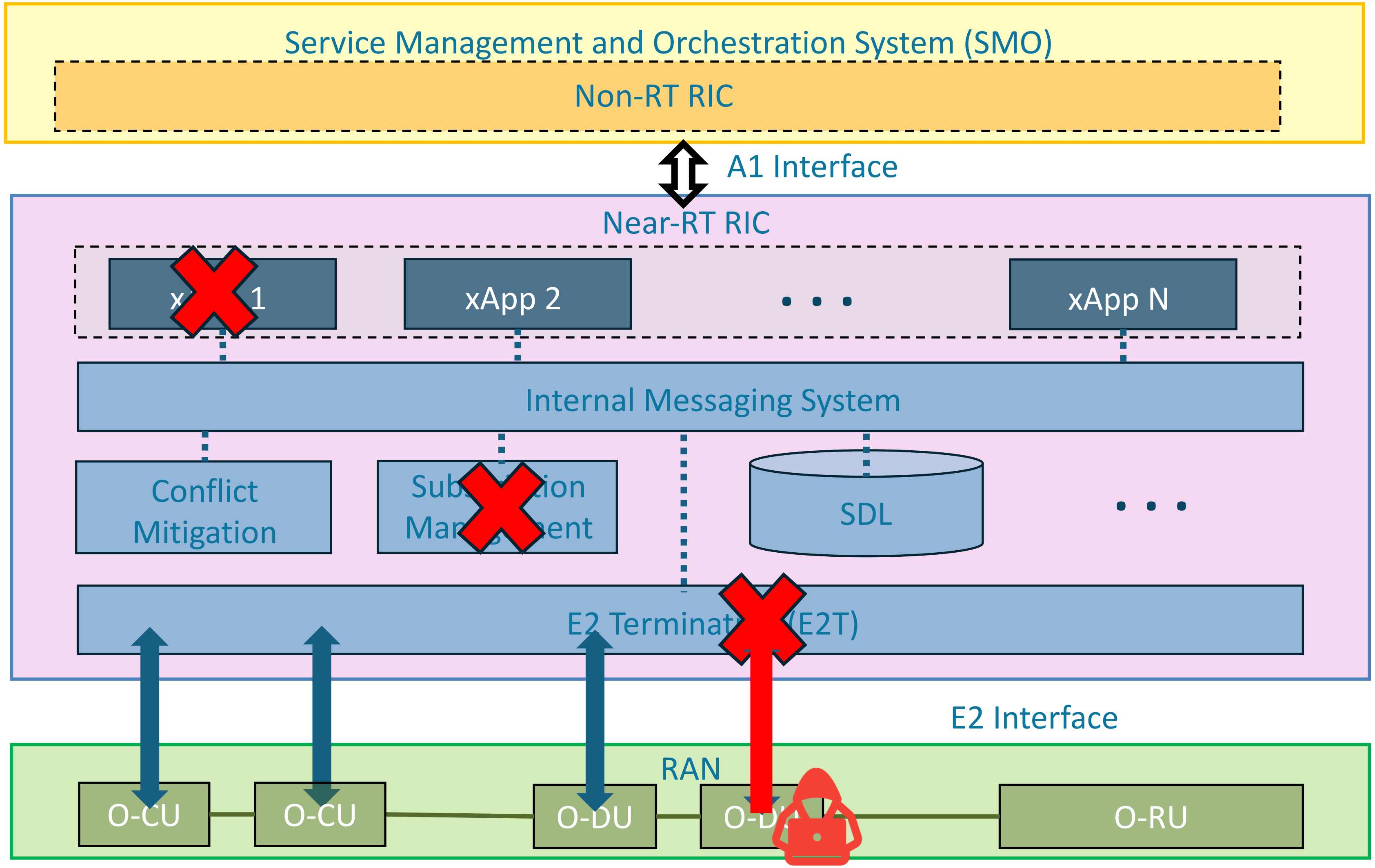
Potential Vulnerabilities in RIC



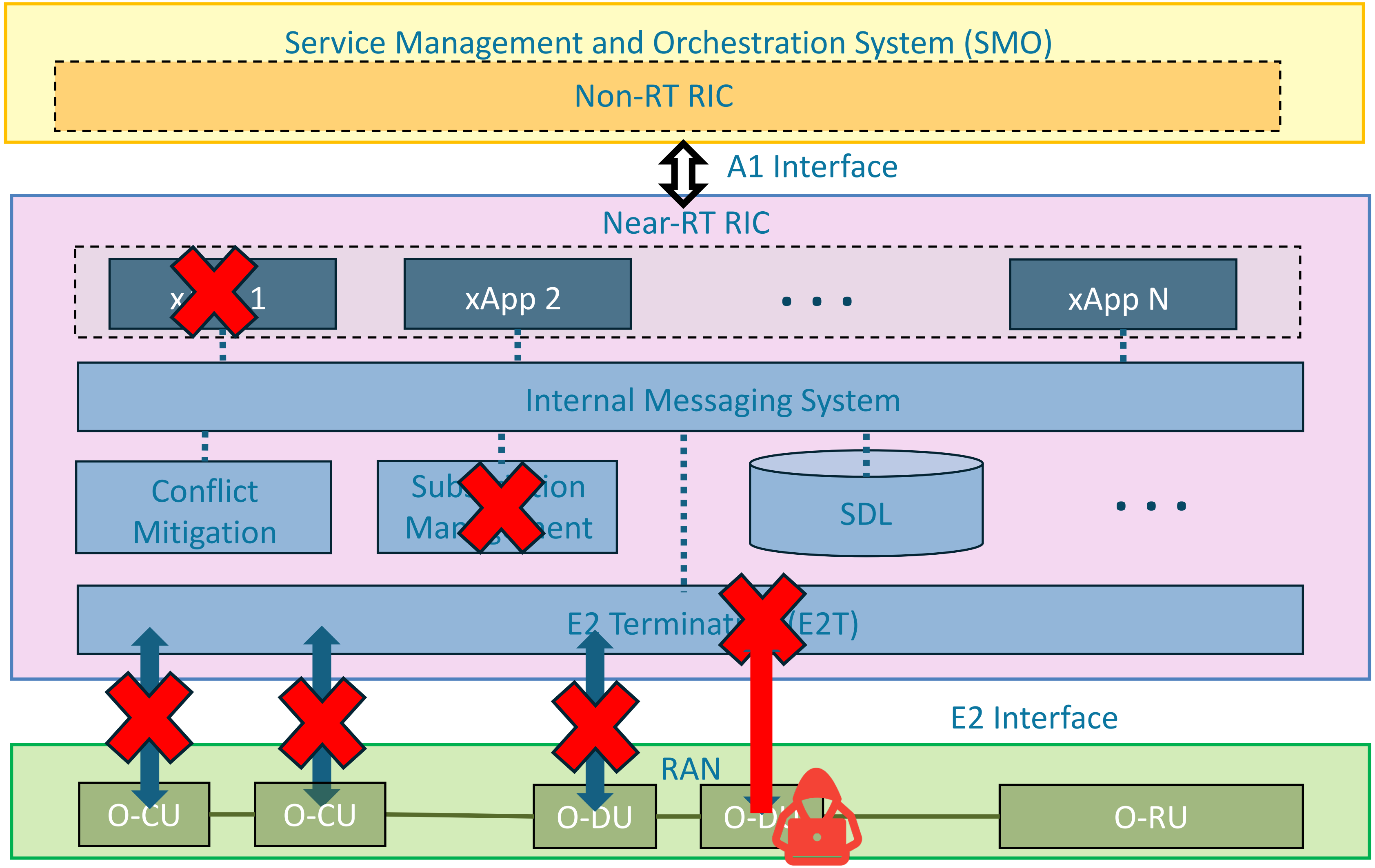
Potential Vulnerabilities in RIC



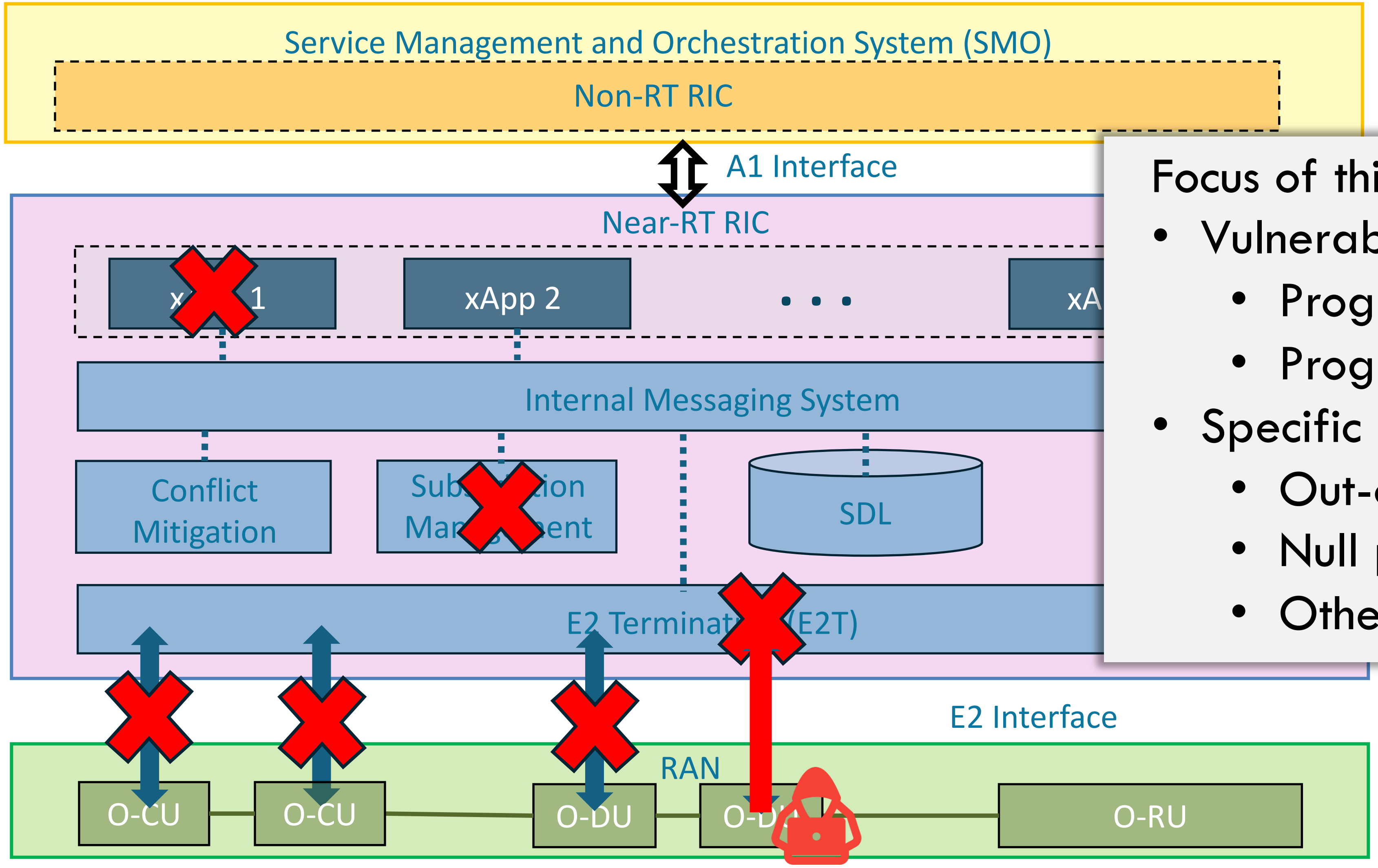
Potential Vulnerabilities in RIC



Potential Vulnerabilities in RIC



Potential Vulnerabilities in RIC



Focus of this work:

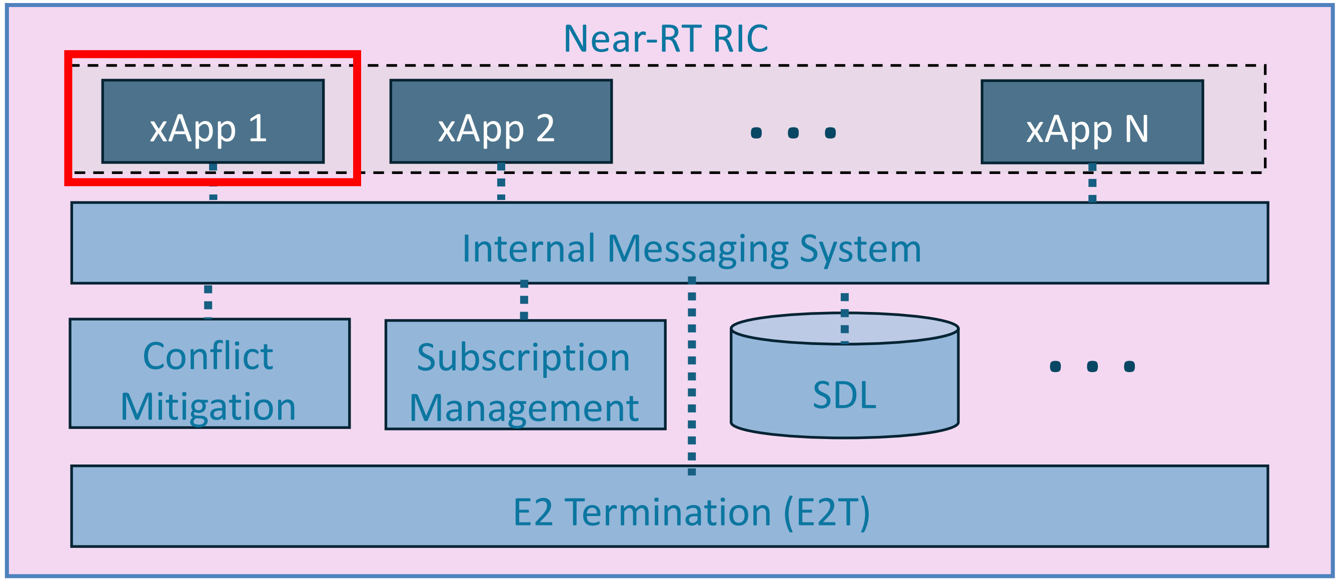
- Vulnerabilities triggering:
  - Program crashes
  - Program hangs
- Specific issues:
  - Out-of-bound access
  - Null pointer dereference
  - Other memory issues

Can we develop an automated reasoning framework to analyze the ***robustness and operational integrity*** *of O-RAN implementations, providing high-security assurances prior to their commercial deployments?*

Design

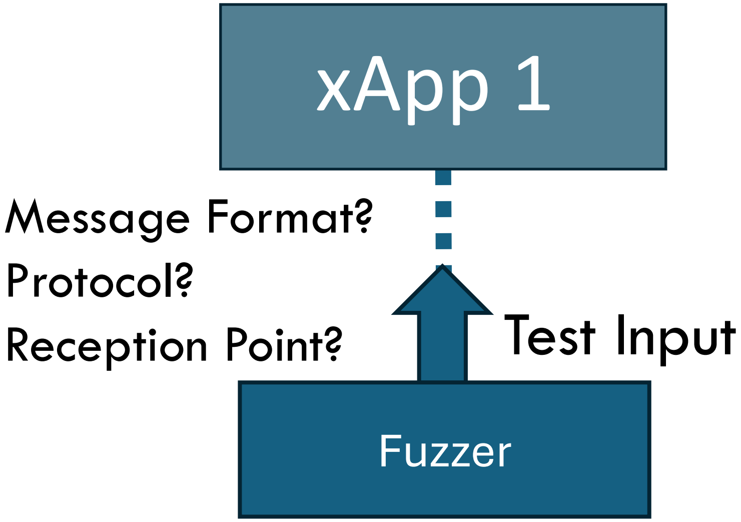
Limitations of Off-the-Shelf Methods

- Existing protocol testers (AFLNET, BooFuzz, Restler, Frizzer) test **one program** at a time
- Requires details about the **expected message**
- **Vary across different implementations**
- High number of **false-positives** (unexploitable vulnerabilities)



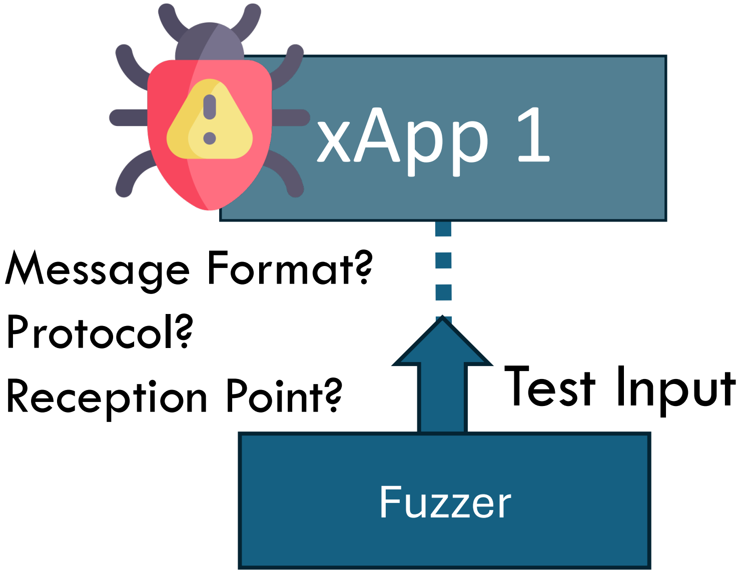
Limitations of Off-the-Shelf Methods

- Existing protocol testers (AFLNET, BooFuzz, Restler, Frizzer) test **one program** at a time
- Requires details about the **expected message**
- **Vary across different implementations**
- High number of **false-positives** (unexploitable vulnerabilities)



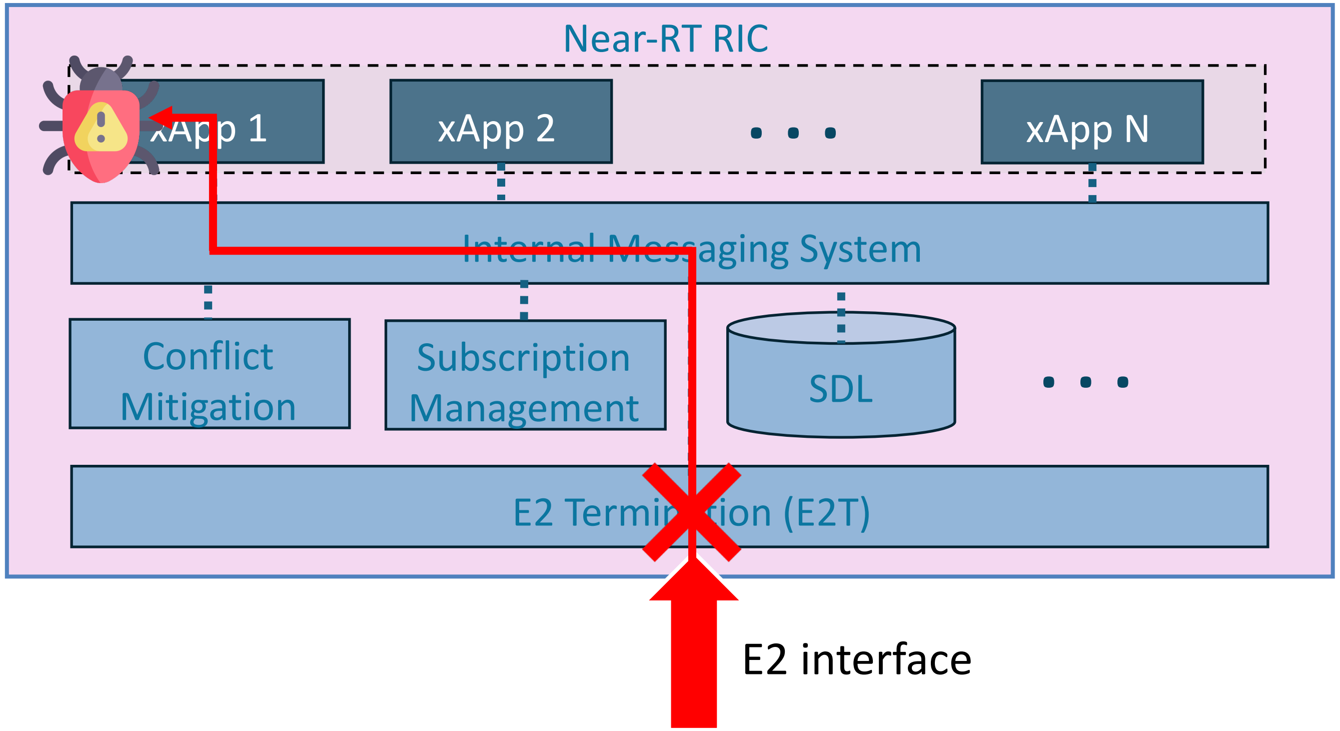
Limitations of Off-the-Shelf Methods

- Existing protocol testers (AFLNET, BooFuzz, Restler, Frizzer) test **one program** at a time
- Requires details about the **expected message**
- **Vary across different implementations**
- High number of **false-positives** (unexploitable vulnerabilities)



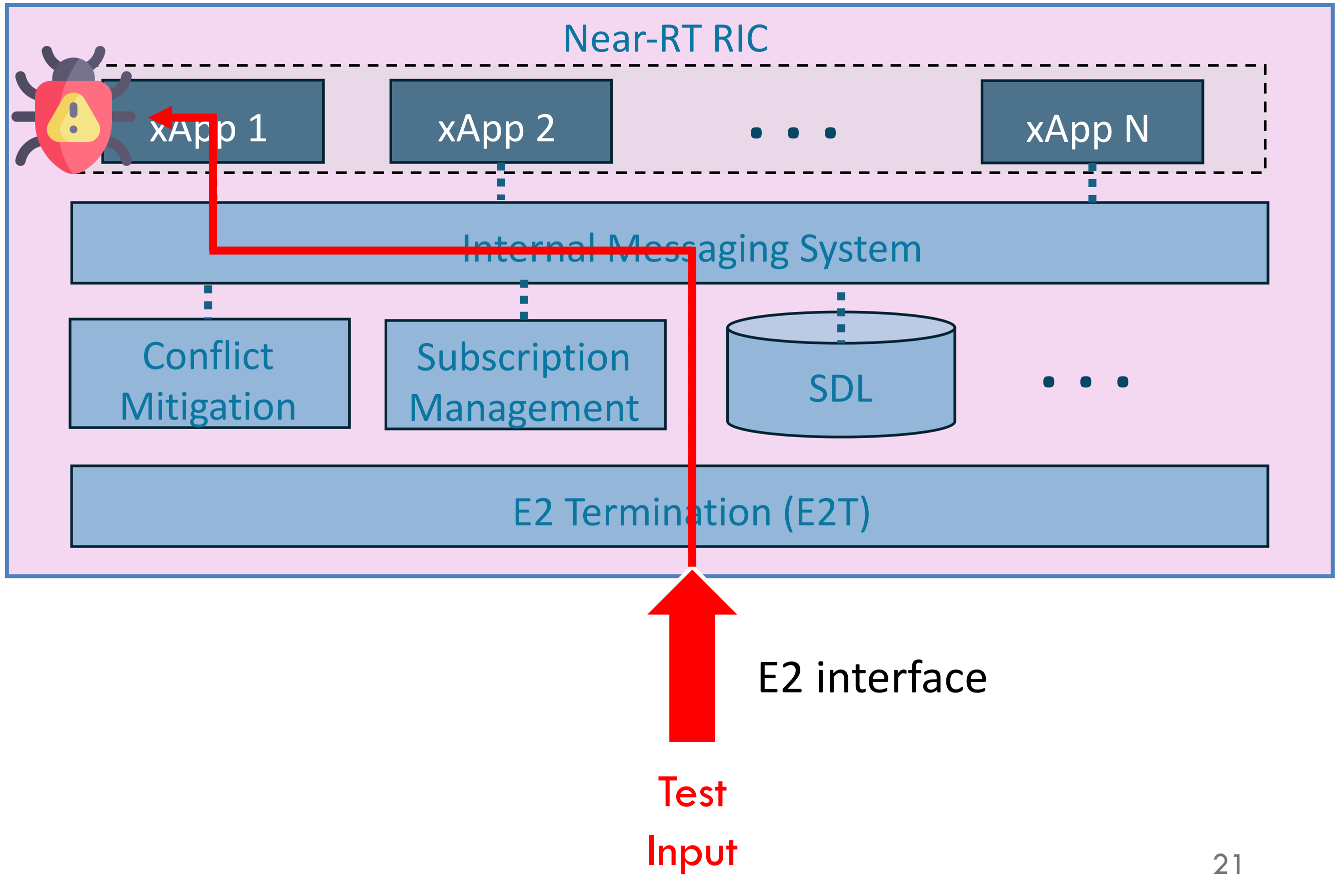
Limitations of Off-the-Shelf Methods

- Existing protocol testers (AFLNET, BooFuzz, Restler, Frizzer) test **one program** at a time
- Requires details about the **expected message**
- **Vary across different implementations**
- High number of **false-positives** (unexploitable vulnerabilities)



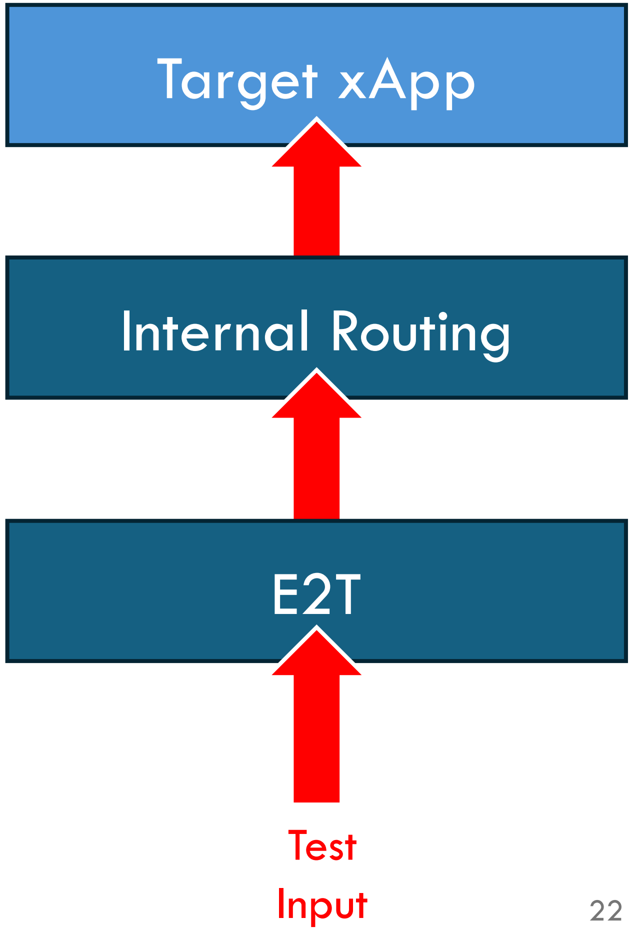
ORANalyst's Approach: End-to-End Testing

- *Send* test inputs only through E2 interface.
- Automatic test input generation for the **standardized E2 protocol**
- All found bugs are exploitable from a misbehaving RAN



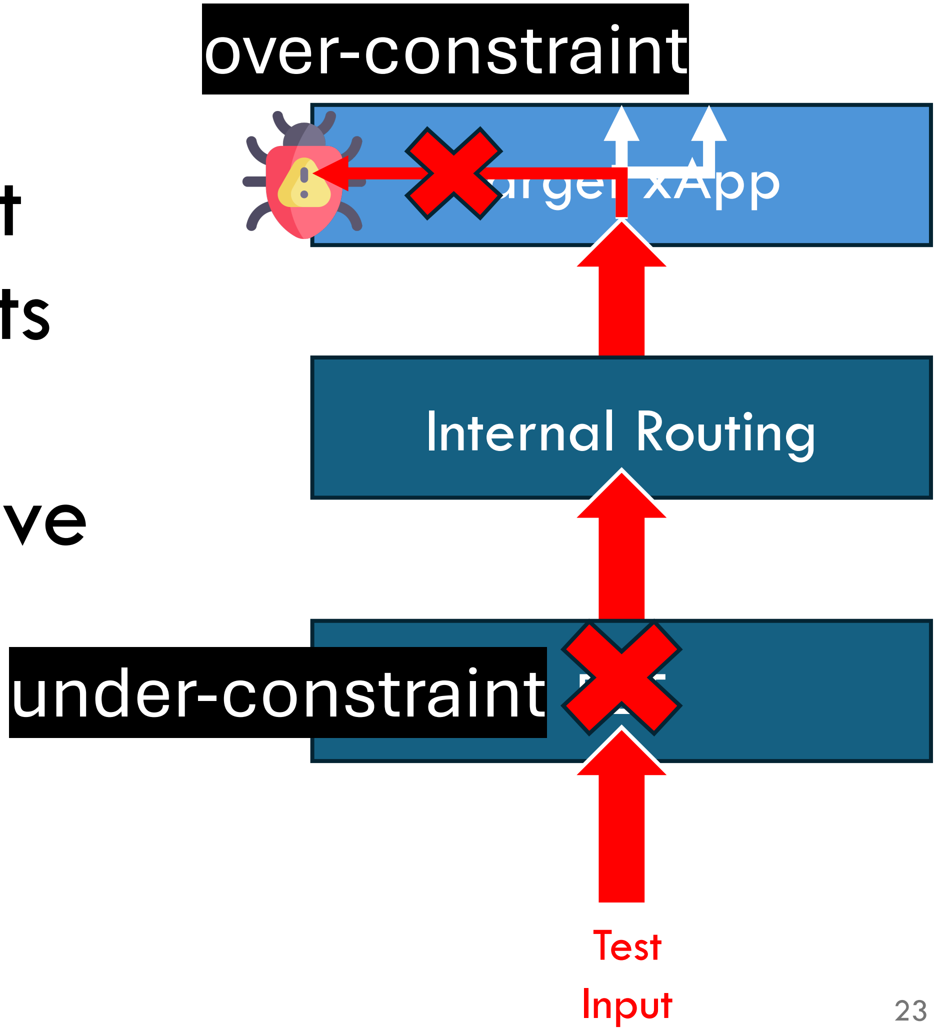
Challenge 1: Generating Targeted and Meaningful Test Inputs

- **Challenge:** generate inputs that can **reach the target** components (avoid under-constraint) while **maintain variability** for effective testing (avoid over-constraint).



Challenge 1: Generating Targeted and Meaningful Test Inputs

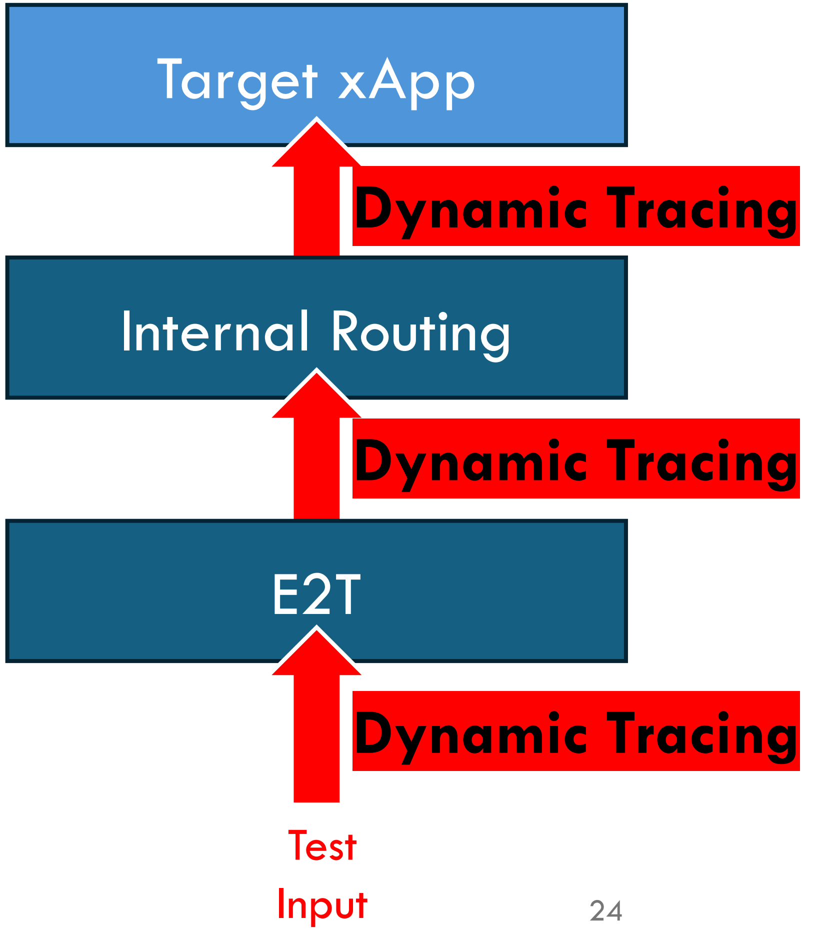
- **Challenge:** generate inputs that can **reach the target** components (avoid under-constraint) while **maintain variability** for effective testing (avoid over-constraint).



Solution 1: Layered Testing Approach

- **Layered approach:**
  - First test the component directly connected with E2 – E2T
  - Gradually move to deeper components
  - At each component, find appropriate constraints so the test inputs can reach the next component.

- **Challenge:** How can we find these **layer-dependencies** between components?
- **Solution: Dynamic tracing**

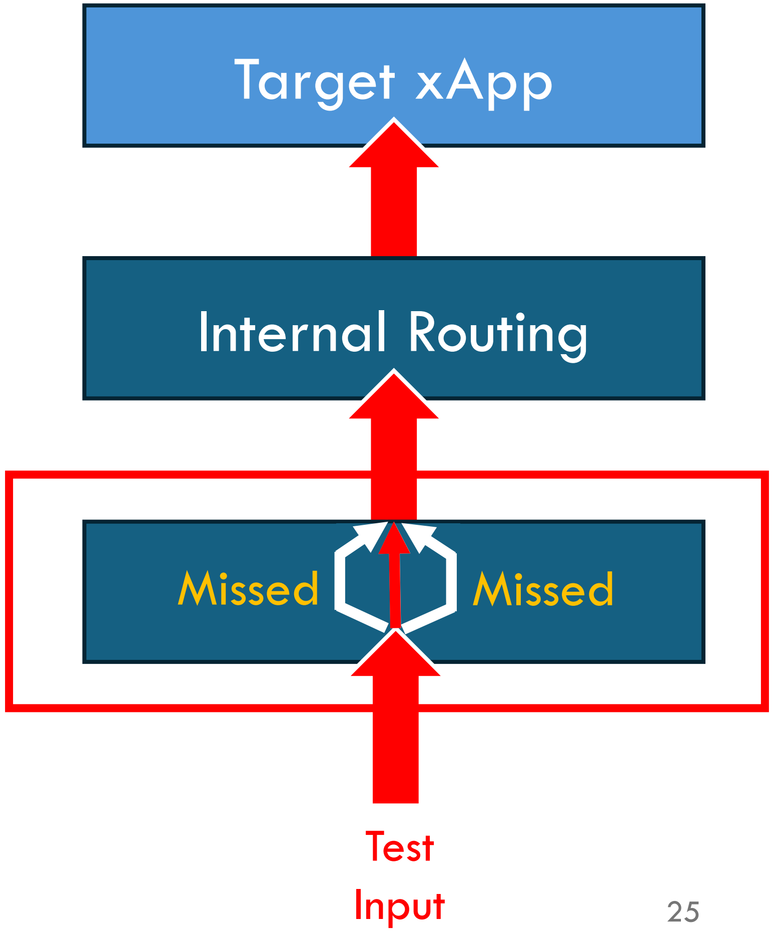


Challenge 2: Enumerate Appropriate Constraints

- Dynamic tracing may miss execution paths in each components.

Solution:

- Collects entry & exit basic blocks in each component during **dynamic tracing**
- Applying **static analysis** to reliably find all execution paths & associated conditions

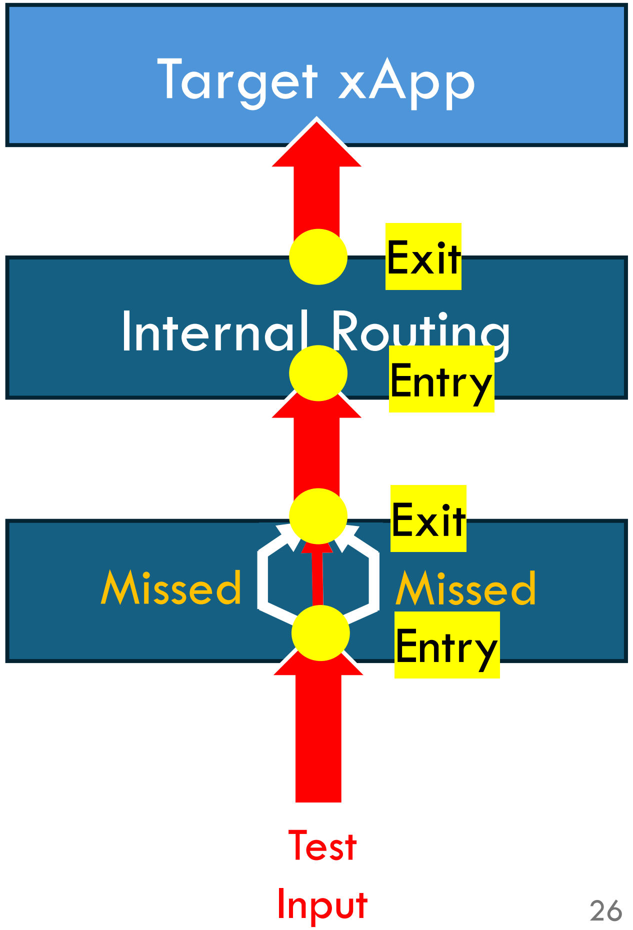


Challenge 2: Enumerate Appropriate Constraints

- Dynamic tracing may miss execution paths in each components.

Solution:

- Collects entry & exit basic blocks in each component during **dynamic tracing**
- Applying **static analysis** to reliably find all execution paths & associated conditions

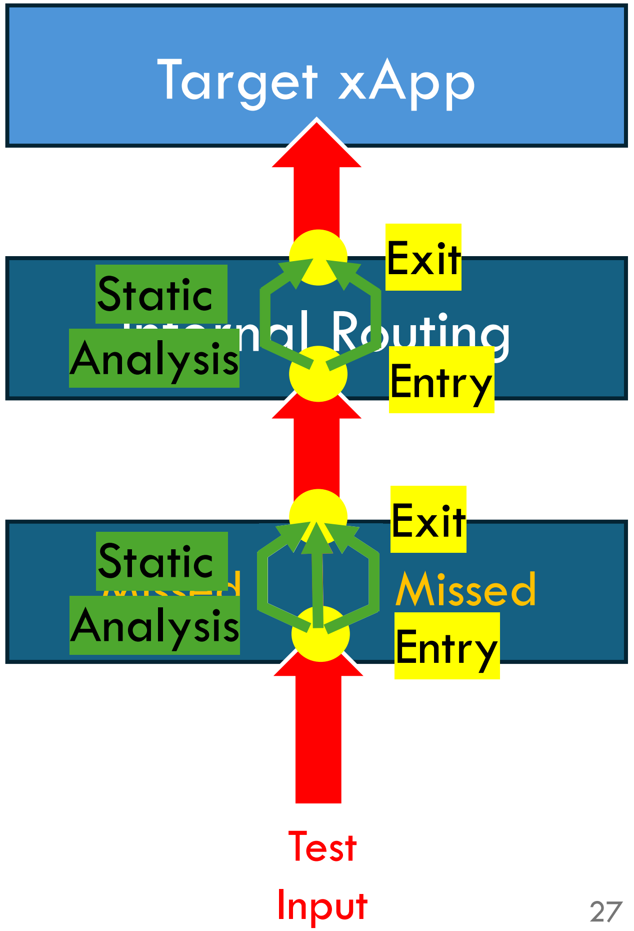


Challenge 2: Enumerate Appropriate Constraints

- Dynamic tracing may miss execution paths in each components.

Solution:

- Collects entry & exit basic blocks in each component during **dynamic tracing**
- Applying **static analysis** to reliably find all execution paths & associated conditions



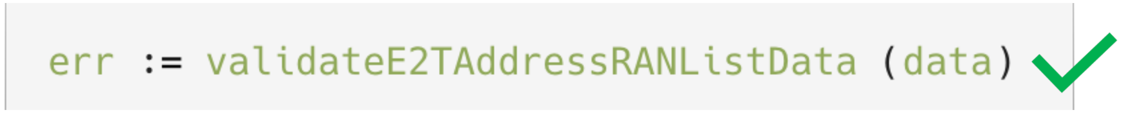
Challenge & Solution 3: Efficient Static Analysis

- **Challenge:** due to complex checks and validation logics performed in RIC components, static analysis runs into **path explosion problem**
  - One component may contain over 6,000 functions and over 100,000 LoC

Solution:

- PDG-based view of control dependencies to find critical conditions
- **Selectively analyze** functions validating inputs, ignoring generic functions (e.g., network operations, data retrieval)

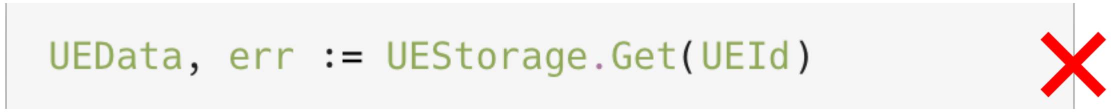
Validating Function:



```
err := validateE2TAddressRANListData (data)
```

✓

Generic Function:

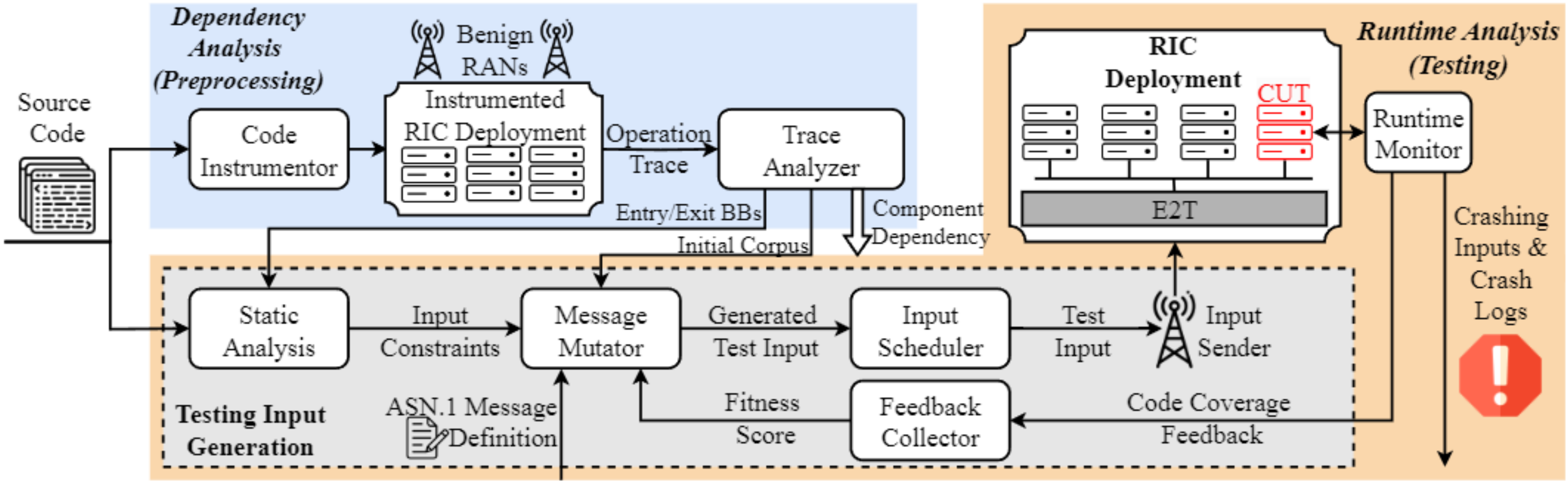


```
UEData, err := UEStorage.Get(UEId)
```

✗

ORANalyst Architecture

- Preprocessing **Dependency Analysis** and Testing **Runtime Analysis**
- Evolutionary feedback-driven fuzz testing

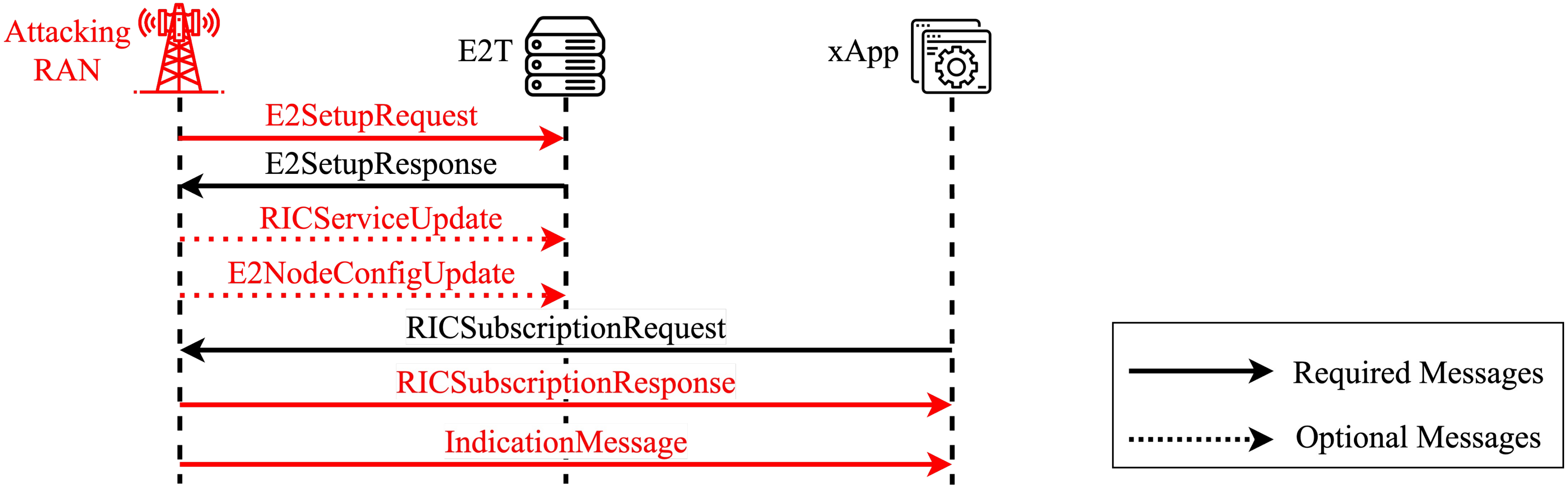


Evaluation

Evaluation Setup & Result

- Evaluated on both available commercially adopted, open-source O-RAN compliant implementations on their latest releases:
  - O-RAN-SC (I release)
  - SD-RAN (1.4 release)
- Evaluated on 10 components across the two implementations, each for 24 hours.
  - O-RAN-SC: E2T, subscription manager, E2 manager, routing manager, Kpimon xApp
  - SD-RAN: E2T, topology management, Rimedo-ts xApp, Kpimon xApp, PCI xApp
- Found **19 critical flaws** in RIC components and xApps that can lead to DoS of the RIC
  - Memory Corruptions
  - Incorrect Error Handlings
  - Thread Issue
- **15 CVEs** have been assigned to track all 19 issues
  - CVE-2024-25377, -29420, -34043, 34044, -34045, -34046, -34047, -34048, -52724, -52725, -52726, -52727, -52728, -34049, -34050

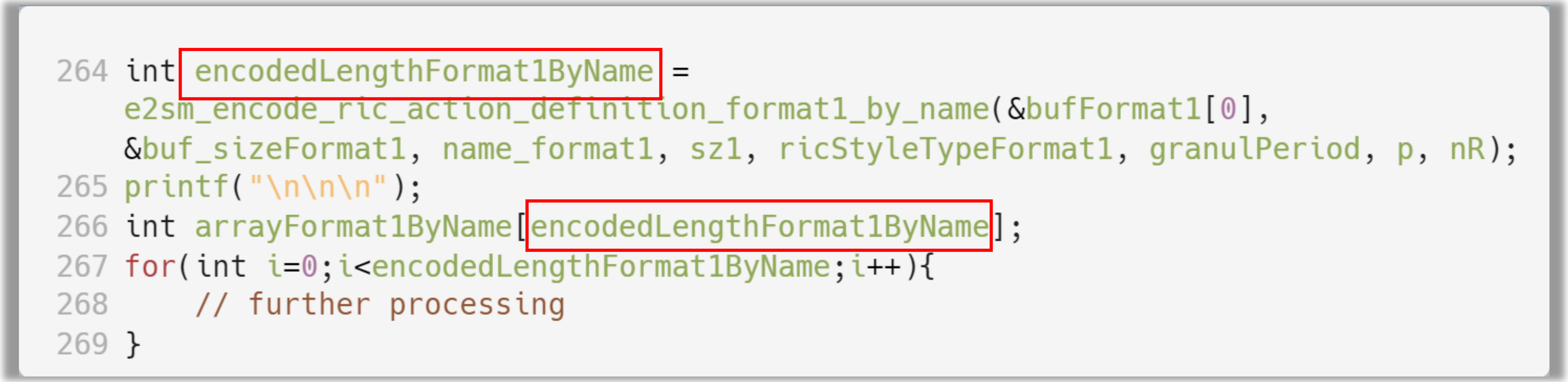
Vulnerable Message Flows



Vulnerability Impact

- Crashed and irresponsive component and applications
- Potential unauthorized memory access
- Communication channel blockage with no error message

Sample Identified Issues: Insufficient Checks



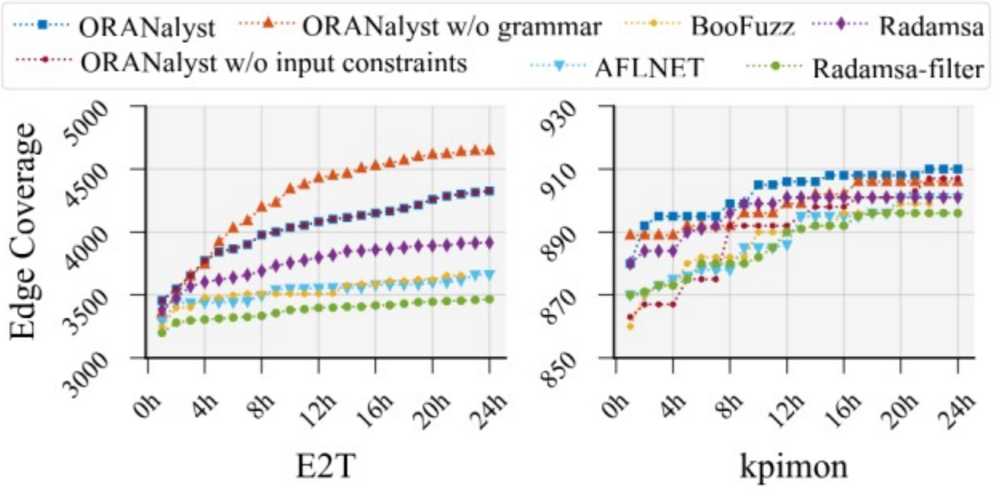
```
int encodedLengthFormat1ByName =
e2sm_encode_ric_action_definition_format1_by_name(&bufFormat1[0],
&buf_sizeFormat1, name_format1, sz1, ricStyleTypeFormat1, granulPeriod, p, nR);
printf("\n\n\n");
int arrayFormat1ByName[encodedLengthFormat1ByName];
for(int i=0;i<encodedLengthFormat1ByName;i++){
    // further processing
}
```

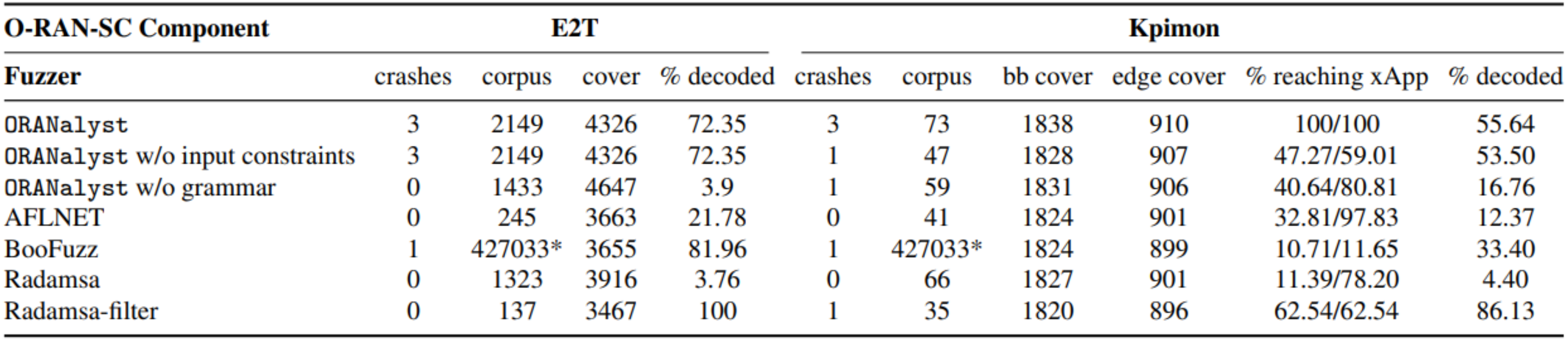
O-RAN-SC's KPIMon xApp
ric-app-kpimon-go/e2sm/wrapper.c

Memory violations due to negative-sized array initialization

Comparative Analysis & Ablation Studies

- Compared against state-of-the-art protocol testers and fuzzers
- 24-hour test time and same initial corpus
- Metrics: code coverage, issues found, % decoded test inputs, % reaching deep components





O-RAN-SC Component	E2T				Kpimon					
Fuzzer	crashes	corpus	cover	% decoded	crashes	corpus	bb cover	edge cover	% reaching xApp	% decoded
ORANalyst	3	2149	4326	72.35	3	73	1838	910	100/100	55.64
ORANalyst w/o input constraints	3	2149	4326	72.35	1	47	1828	907	47.27/59.01	53.50
ORANalyst w/o grammar	0	1433	4647	3.9	1	59	1831	906	40.64/80.81	16.76
AFLNET	0	245	3663	21.78	0	41	1824	901	32.81/97.83	12.37
BooFuzz	1	427033*	3655	81.96	1	427033*	1824	899	10.71/11.65	33.40
Radamsa	0	1323	3916	3.76	0	66	1827	901	11.39/78.20	4.40
Radamsa-filter	0	137	3467	100	1	35	1820	896	62.54/62.54	86.13

Conclusion

- *ORANalyst*: first testing framework to test the operational robustness of O-RAN's **service-based** RIC implementations.
- Combines **dynamic tracing with effective static analysis**
- Evaluation of ORANalyst on two open-source commercially-adopted RIC implementations reveals 19 previously undiscovered vulnerabilities, with 15 CVEs assigned.
- *ORANalyst* outperforms state-of-the-art protocol testers in code coverage, issues found, and effectiveness of generated inputs.
- ORANalyst is available at github.com/SyNSec-den/ORANalyst

ORANalyst: Systematic Testing Framework for Open RAN Implementations

https://github.com/SyNSec-den/ORANalyst

Tianchang Yang

Contact: tzy5088@psu.edu

tianchang-yang.github.io