Open RAN, Open Risk: Uncovering Threats and Exposing Vulnerabilities in Next-Gen Cellular RAN

Tianchang Yang, Kai Tu,

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

Introduction

Tianchang Yang

Research Assistant, The Pennsylvania State University

Mobile network security, resiliency, and robustness: 5G, Open RAN, baseband

tianchang-yang.github.io

Introduction

Kai Tu

Research Assistant, The Pennsylvania State University

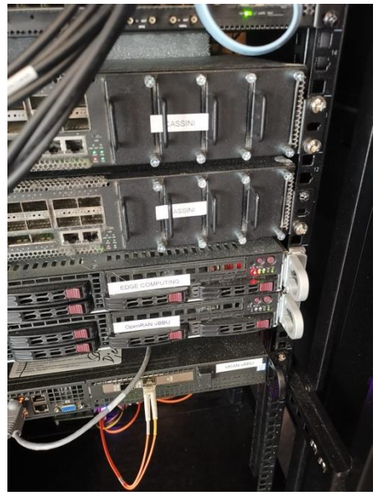
Mobile network and Device Security, baseband security, Automatic Vulnerability Discovery

`hellotkk.github.io`

Radio Access Network (RAN)

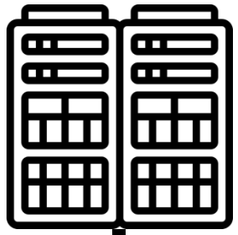
www.anscorporate.com/blog/what-is-a-5g-cell-tower

O-RAN's Virtualization



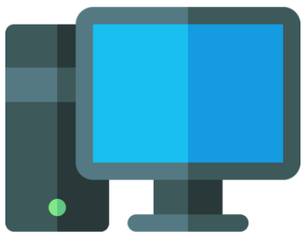
the-mobile-network.com/2019/03/taking-the-open-ran-commercial/
the-mobile-network.com/2019/03/open-ran-at-the-tip-ping-point/

Evolution of Mobile RAN

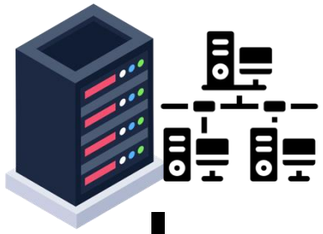


Mainframe of specialized hardware

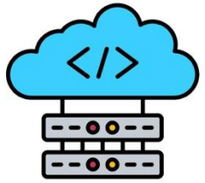
1960-1970



Programs running on modularized hardware

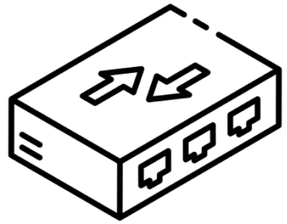


Specialized hardware, servers, switches



SDN

~2010



Simple programmable switches

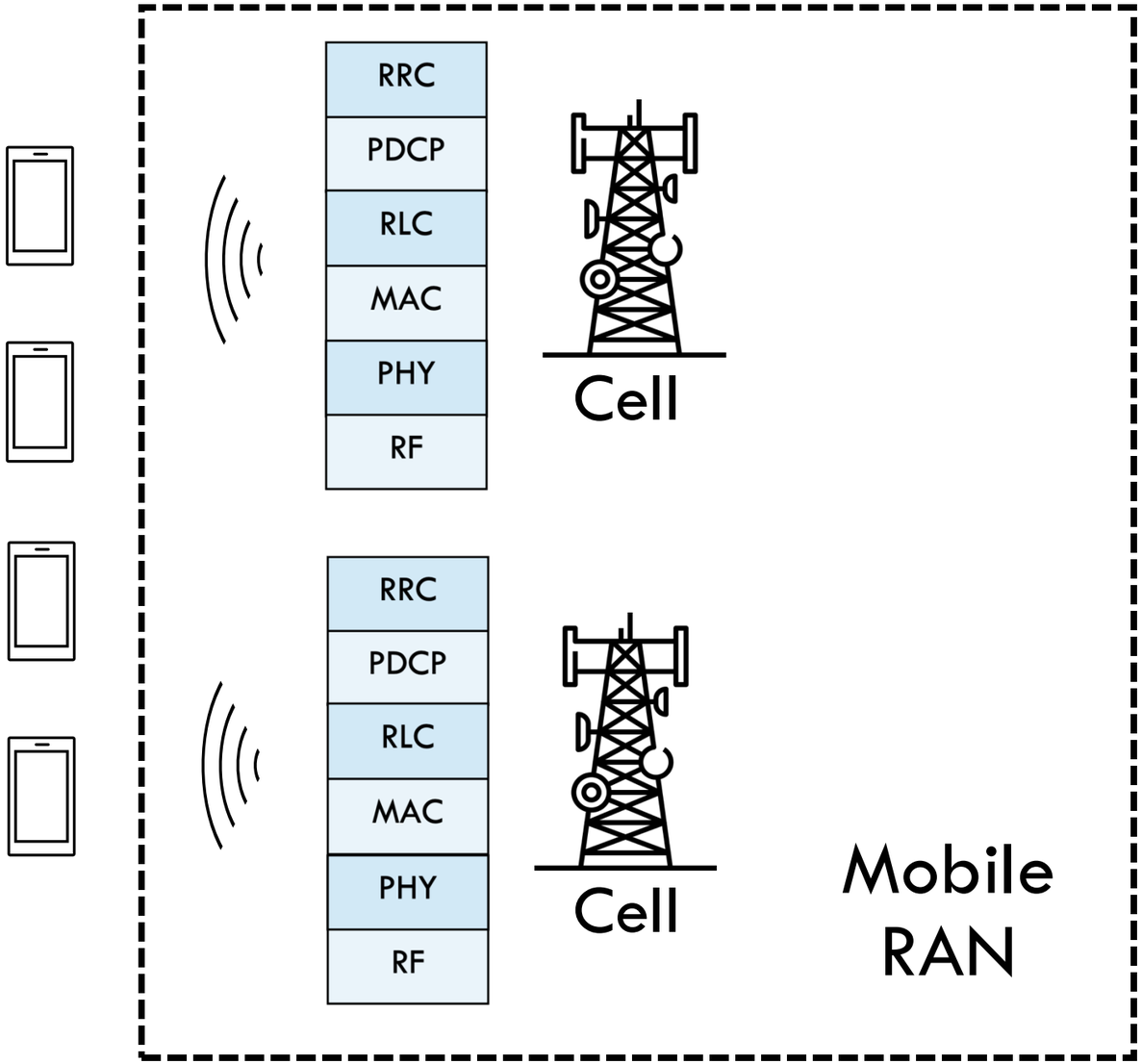
Vendor-provided cell equipment

O-RAN ALLIANCE

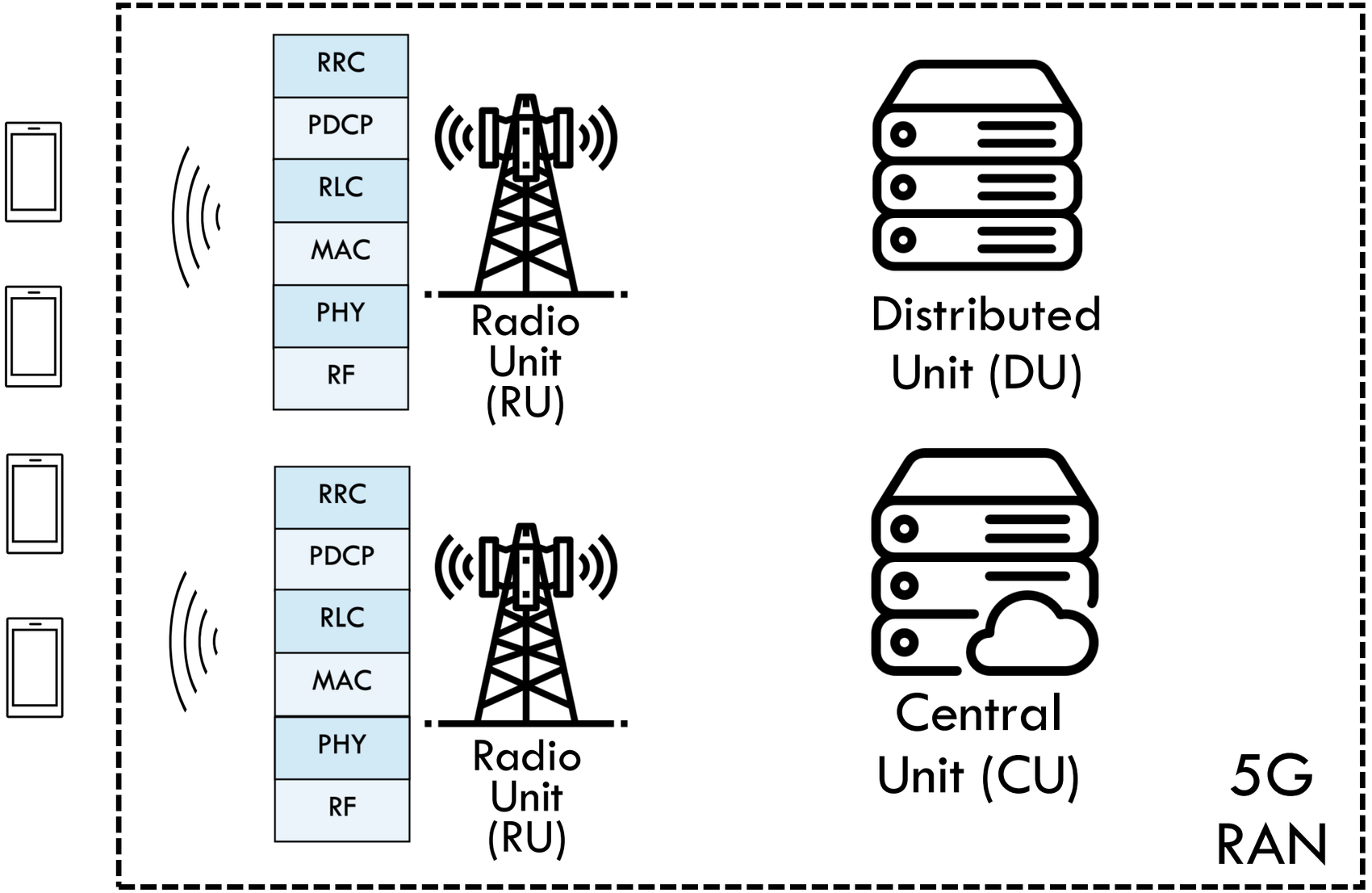
Virtualized, cloud-native RAN on commodity servers

4G and Before RAN

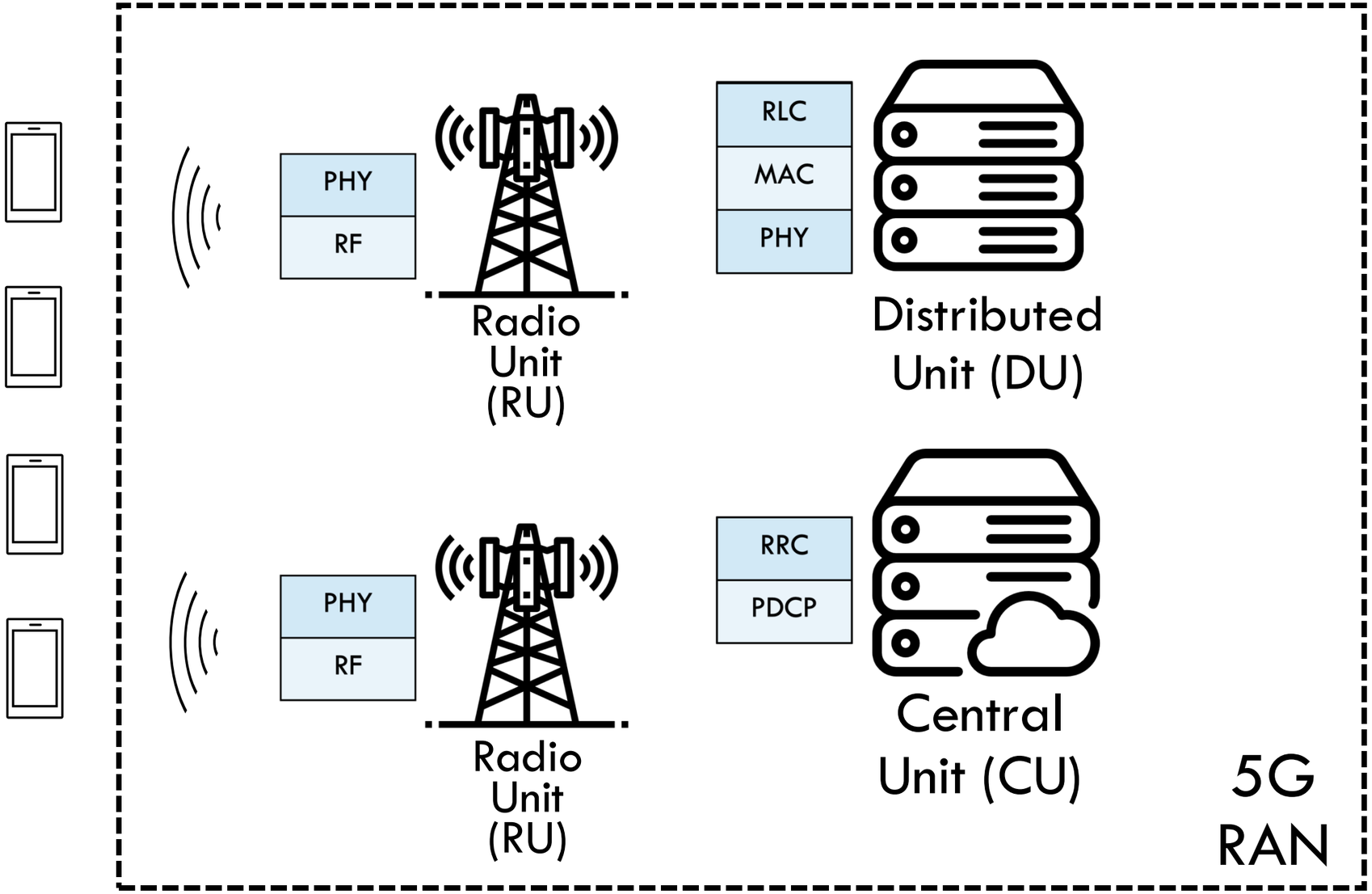
4G and Before RAN



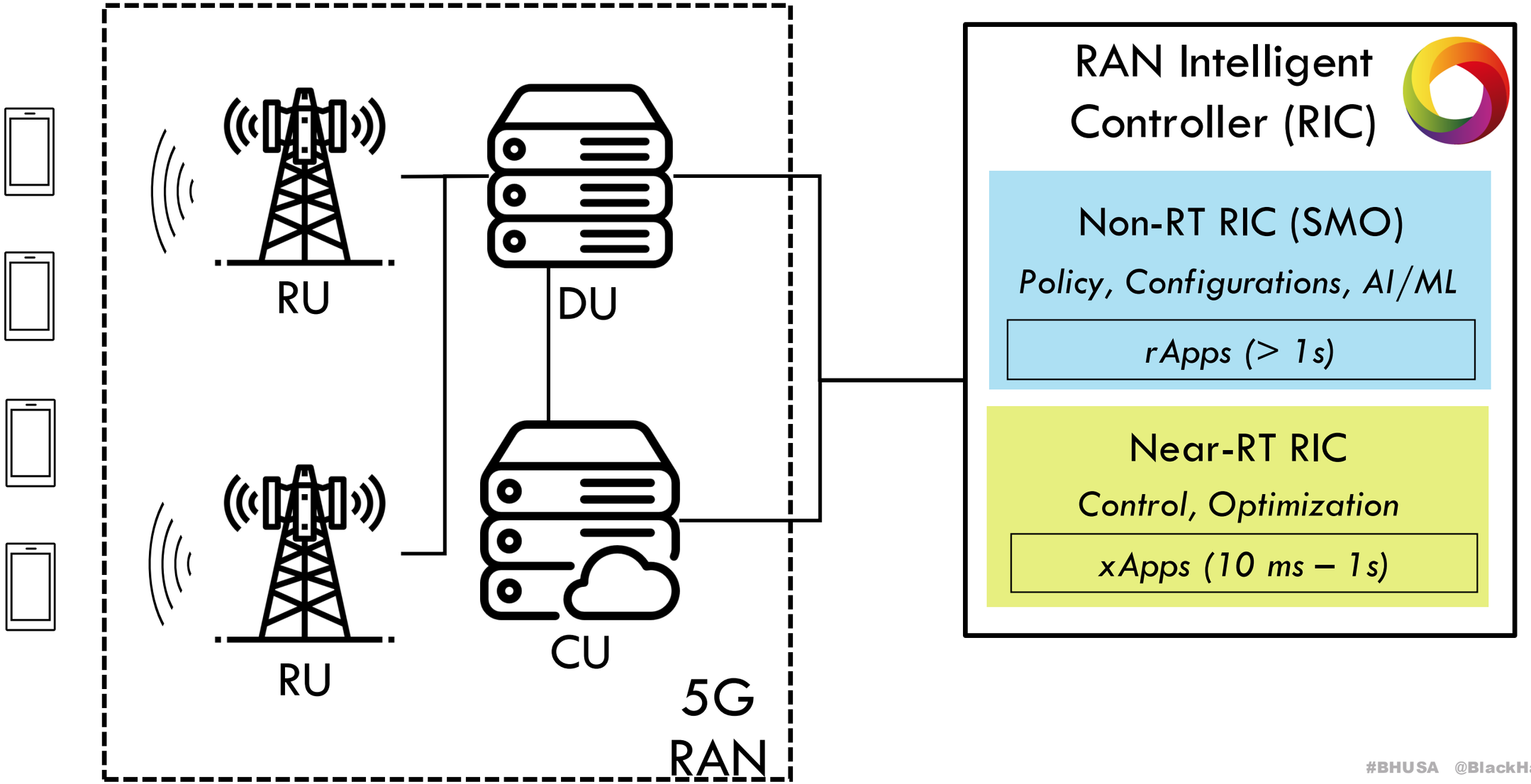
Mobile Network's Transition to 5G



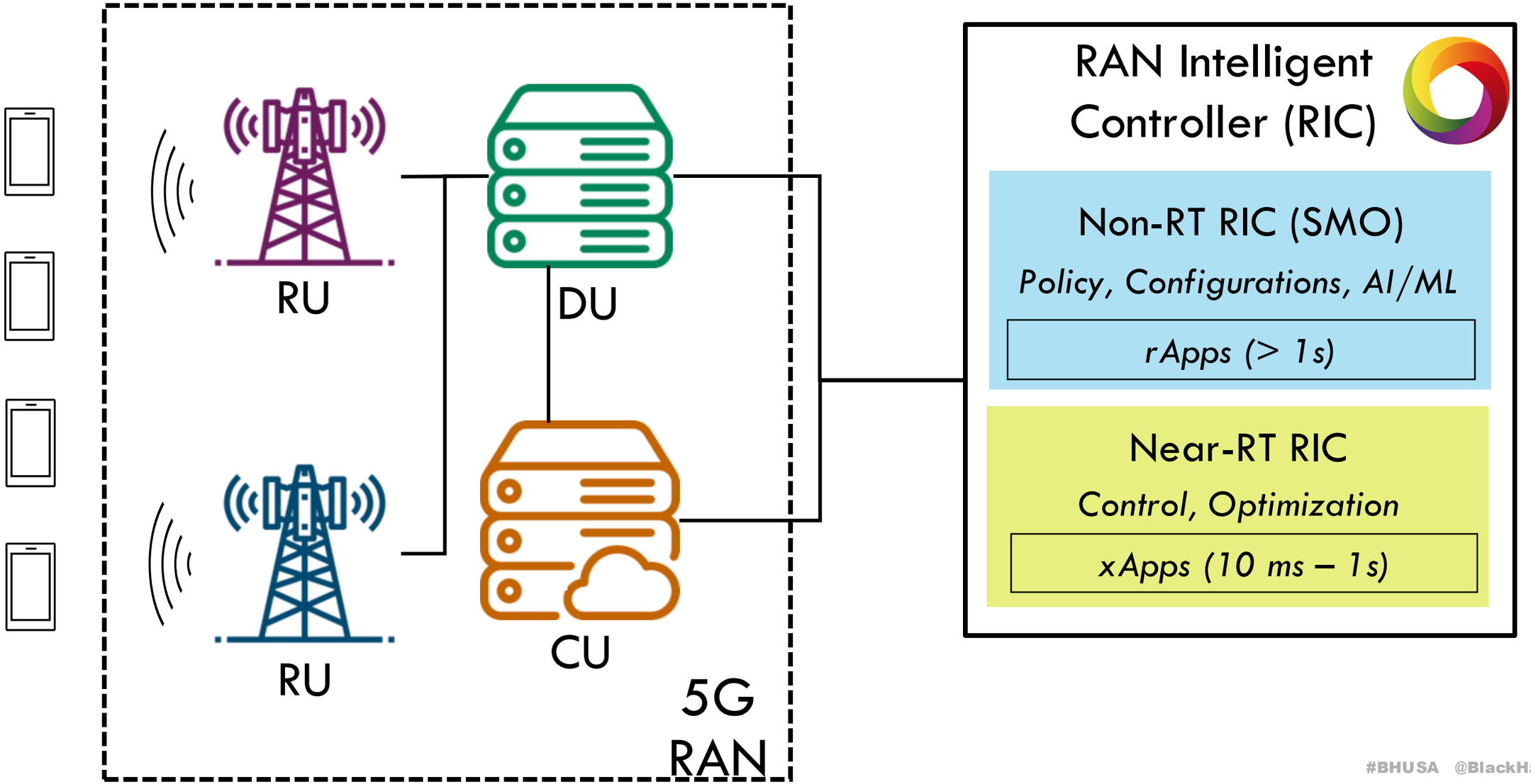
Mobile Network's Transition to 5G



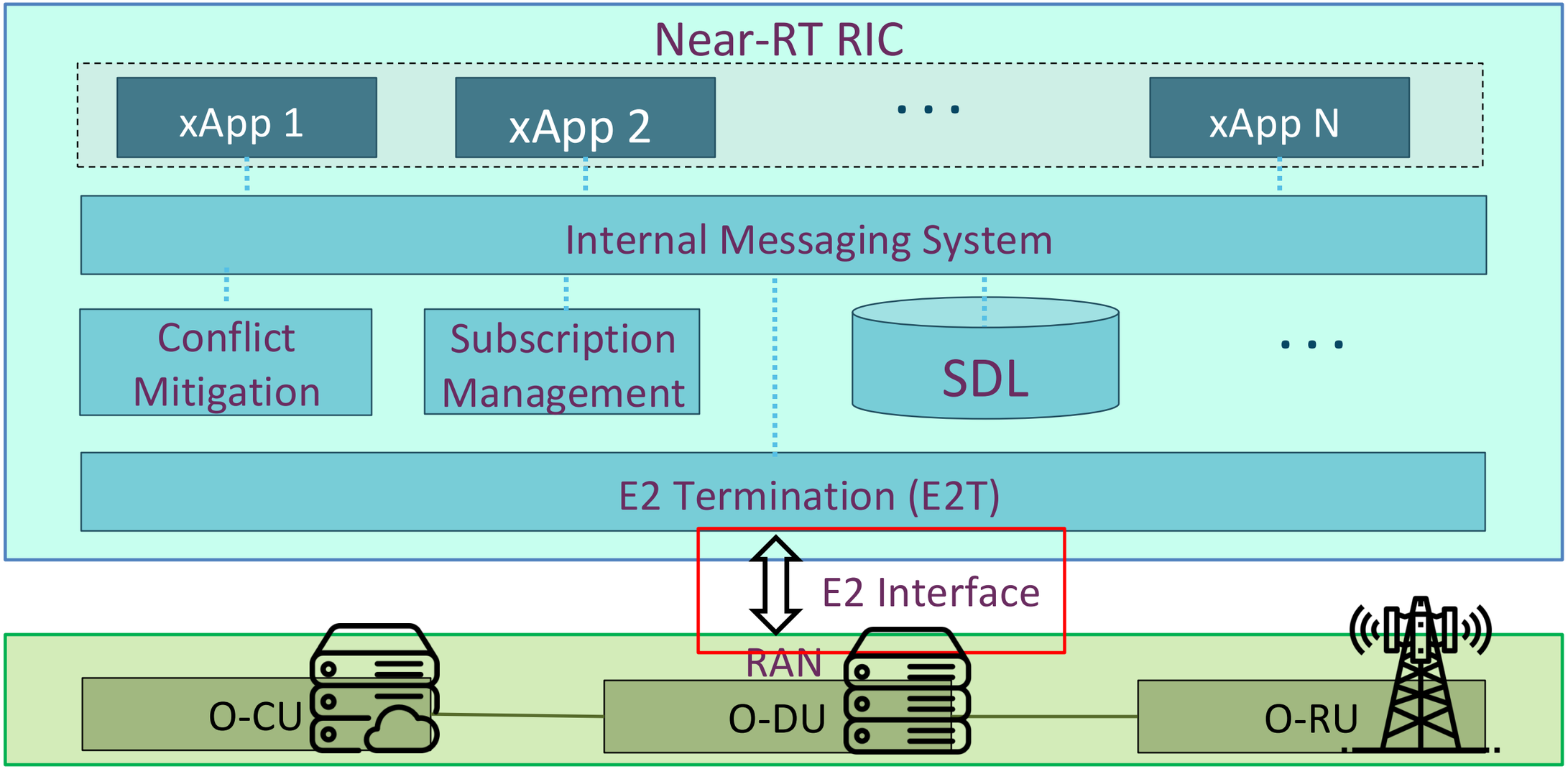
Introduction of O-RAN



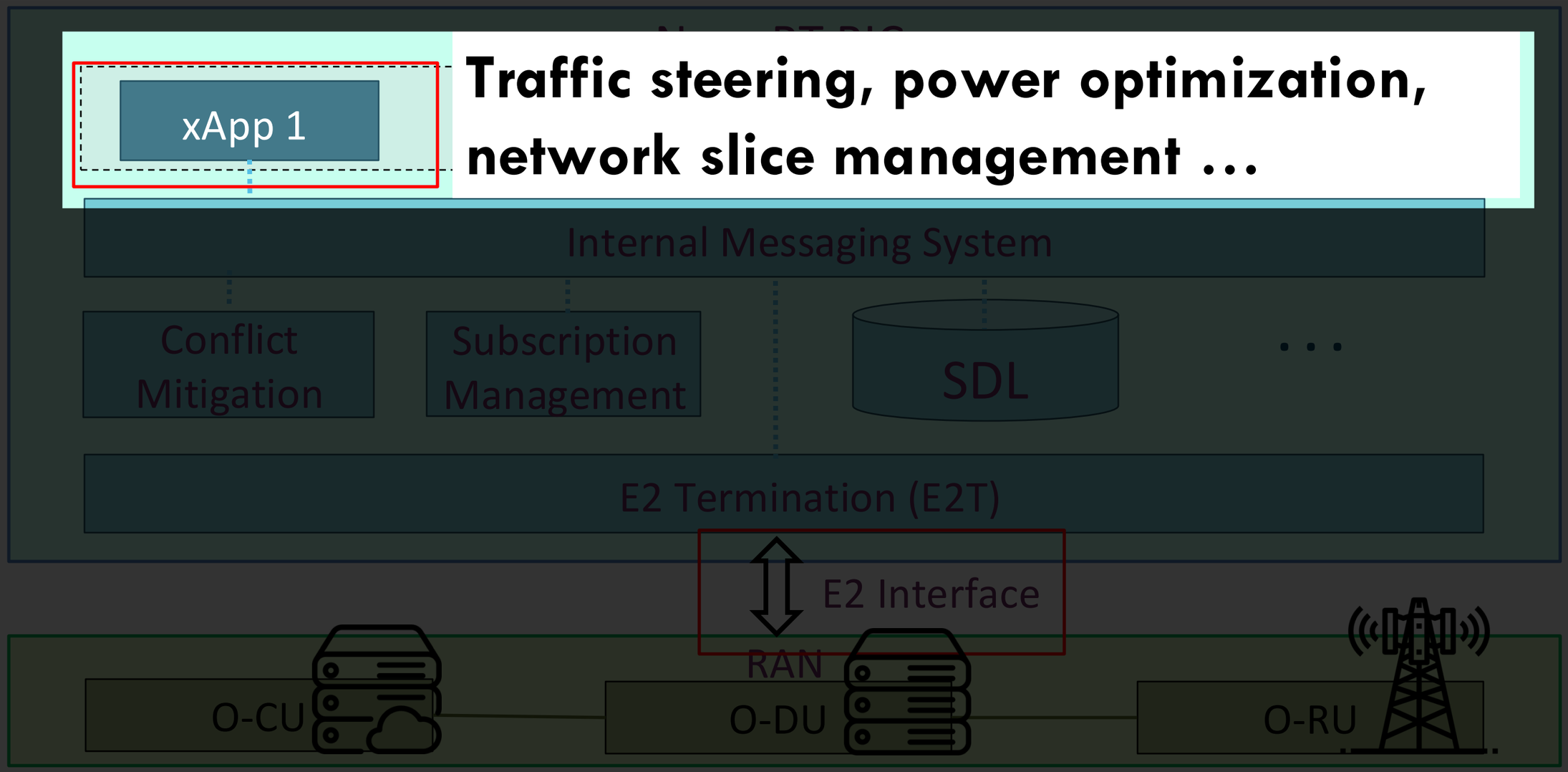
Introduction of O-RAN



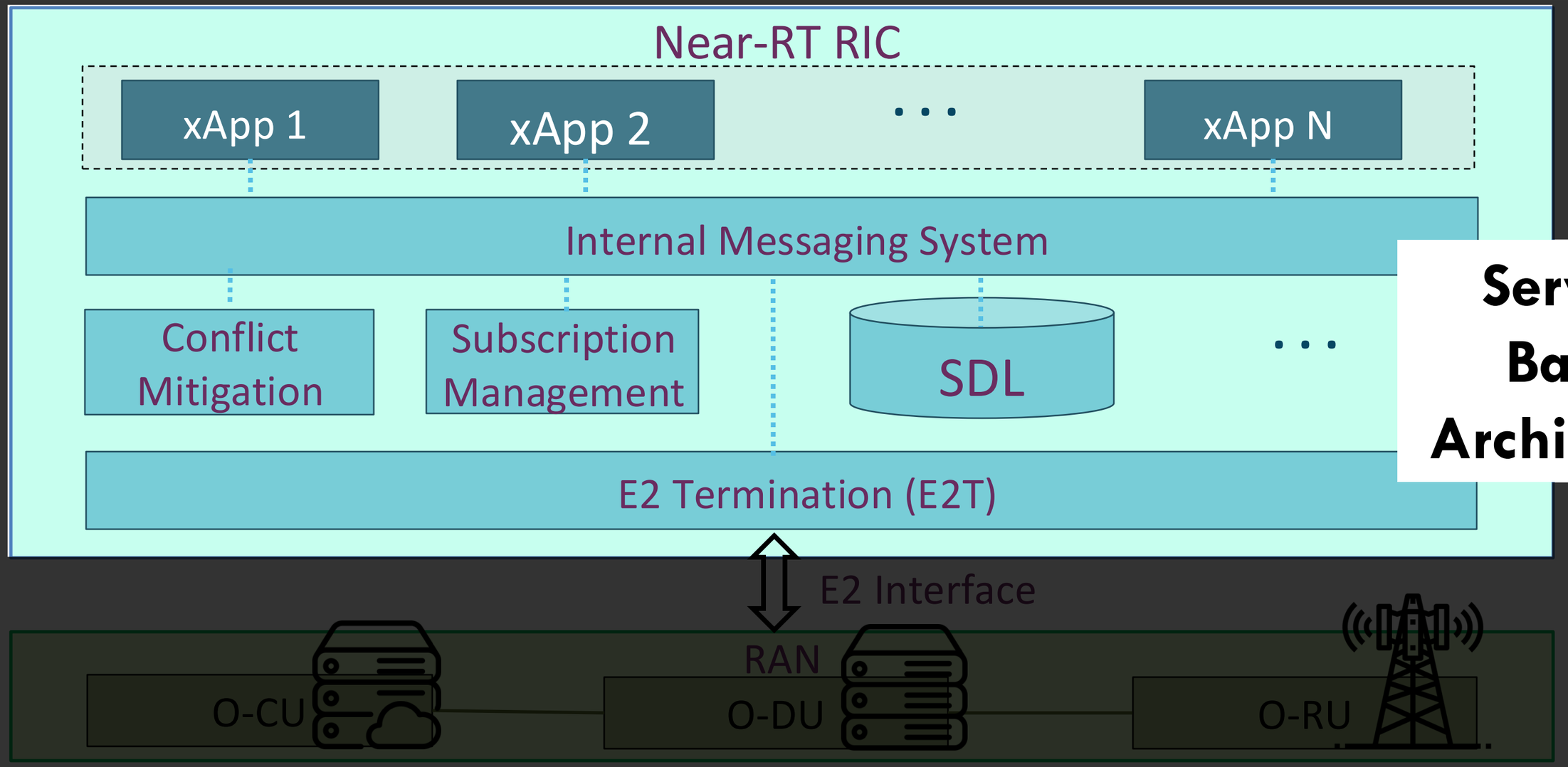
O-RAN RIC Architecture



O-RAN RIC Architecture

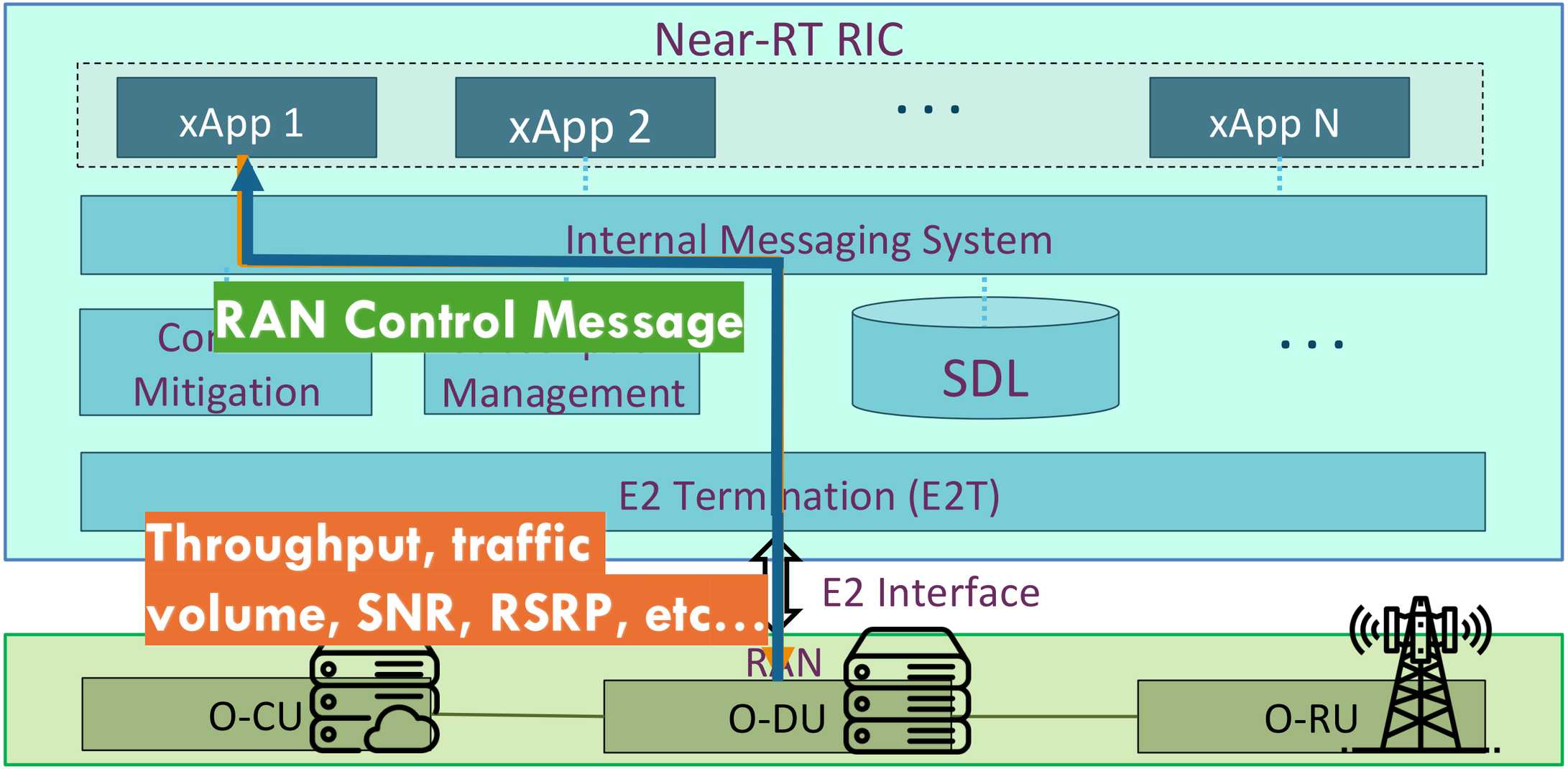


O-RAN RIC Architecture

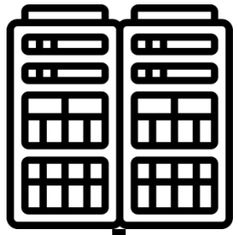


Service-Based Architecture

O-RAN RIC Architecture

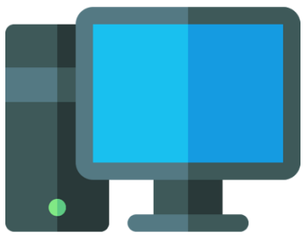


Evolution of Mobile RAN

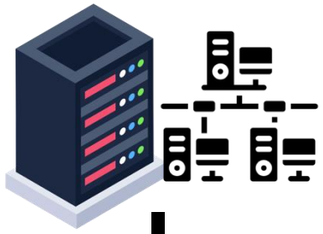


Mainframe of specialized hardware

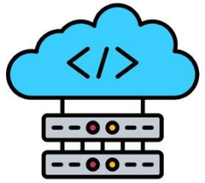
1960-1970



Programs running on modularized hardware

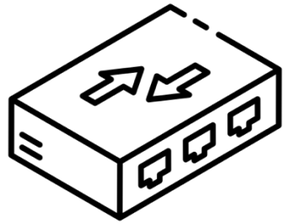


Specialized hardware, servers, switches



SDN

~2010



Simple programmable switches

Vendor-provided cell equipment

O-RAN ALLIANCE

Virtualized, cloud-native RAN on commodity servers

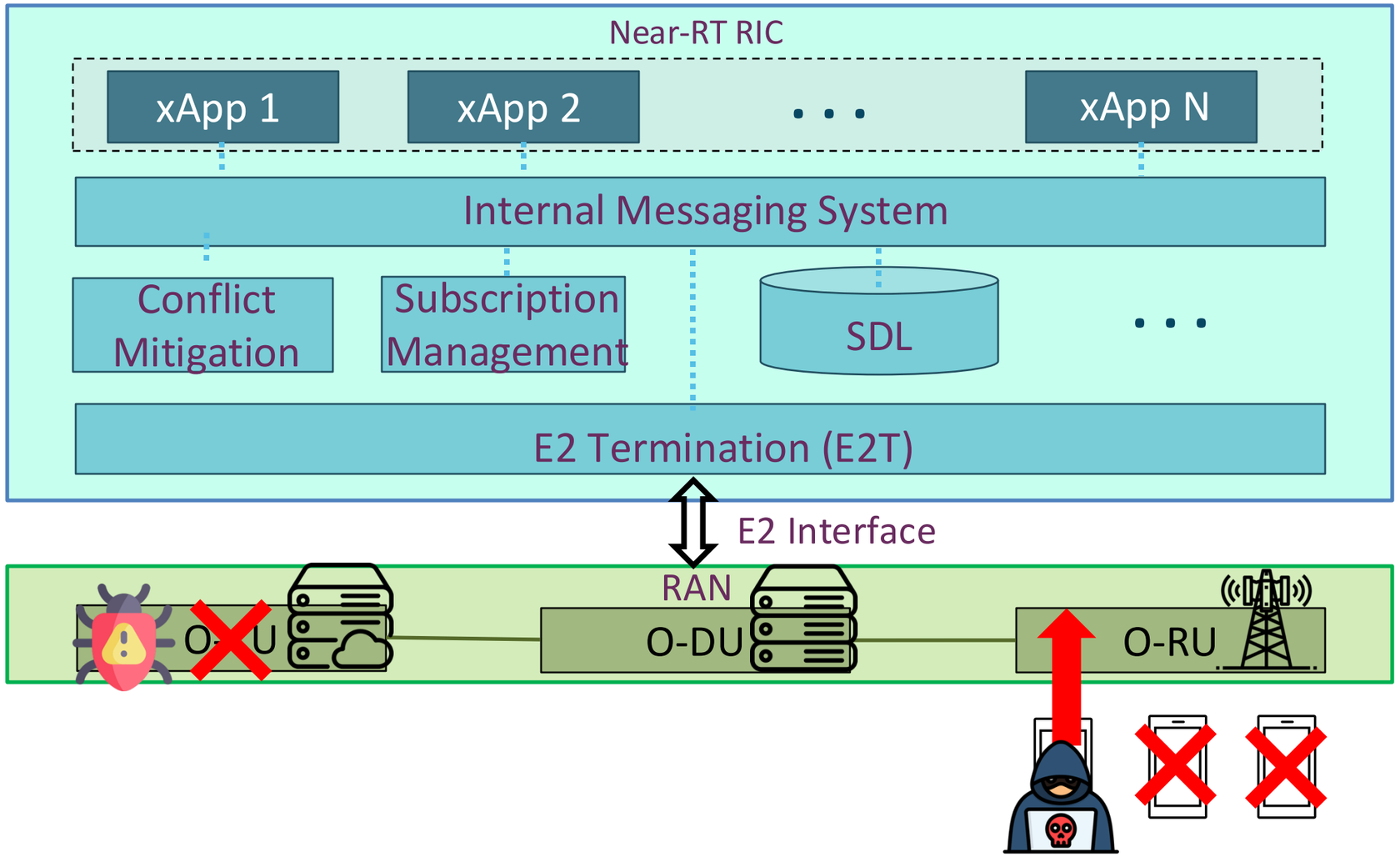
Are O-RAN already in use?

Major operators still opting for single vendor, small or private operators benefiting first

Will *everyone* move to O-RAN eventually?

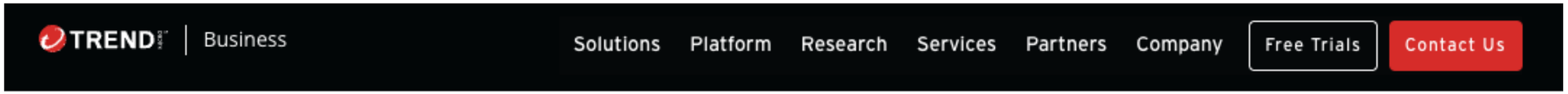
Maybe not every O-RAN promise…
But with AI/ML/LLM booming, cloud-native RAN is inevitable.

Threat Demonstration – Malicious User



O-RAN Vulnerability Sources

Malicious User



Exploits & Vulnerabilities

Attacks on 5G Infrastructure From Users' Devices

Crafted packets from cellular devices such as mobile phones can exploit faulty state machines in the 5G core to attack cellular infrastructure. Smart devices that critical industries such as defense, utilities, and the medical sectors use for their daily operations depend on the speed, efficiency, and productivity brought by 5G. This entry describes CVE-2021-45462 as a potential use case to deploy a denial-of-service (DoS) attack to private 5G networks.

By: Salim S.I.
September 20, 2023
Read time: 8 min (2106 words)

Attacks on 5G Infrastructure from Users' Devices
www.trendmicro.com/en_us/research/23/i/attacks-on-5g-infrastructure-from-users-devices.html

Computer Science > Cryptography and Security

[Submitted on 22 Apr 2025]

RRC Signaling Storm Detection in O-RAN

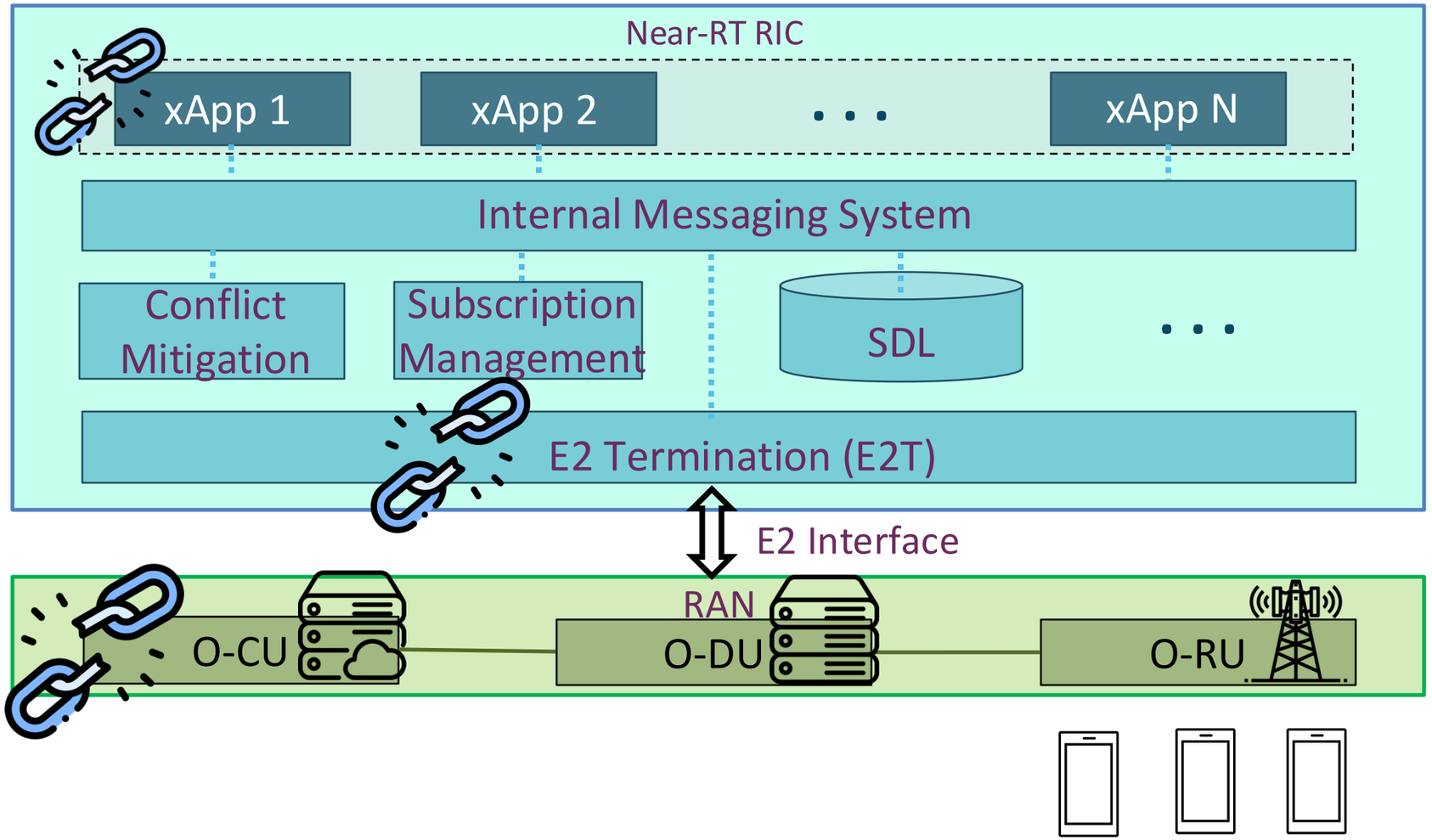
Dang Kien Nguyen, Rim El Malki, Filippo Rebecchi

The Open Radio Access Network (O-RAN) marks a significant shift in the mobile network industry. By transforming a traditionally vertically integrated architecture into an open, data-driven one, O-RAN promises to enhance operational flexibility and drive innovation. In this paper, we harness O-RAN's openness to address one critical threat to 5G availability: signaling storms caused by abuse of the Radio Resource Control (RRC) protocol. Such attacks occur when a flood of RRC messages from one or multiple User Equipments (UEs) deplete resources at a 5G base station (gNB), leading to service degradation. We provide a reference implementation of an RRC signaling storm attack, using the OpenAirInterface (OAI) platform to evaluate its impact on a gNB. We supplement the experimental results with a theoretical model to extend the findings for different load conditions. To mitigate RRC signaling storms, we develop a threshold-based detection technique that relies on RRC layer features to distinguish between malicious activity and legitimate high network load conditions. Leveraging O-RAN capabilities, our detection method is deployed as an external Application (xApp). Performance evaluation shows attacks can be detected within 90ms, providing a mitigation window of 60ms before gNB unavailability, with an overhead of 1.2% and 0% CPU and memory consumption, respectively.

Comments: Accepted to IEEE ISCC 2025
Subjects: **Cryptography and Security (cs.CR)**; Networking and Internet Architecture (cs.NI)
Cite as: arXiv:2504.15738 **[cs.CR]**
(or arXiv:2504.15738v1 **[cs.CR]** for this version)
https://doi.org/10.48550/arXiv.2504.15738

RRC Signaling Storm Detection in O-RAN
arxiv.org/abs/2504.15738

Threat Demonstration – Supply Chain Risks



O-RAN Vulnerability Sources

Malicious User

Supply Chain Risk

P1 SECURITY

Products & Services Usecases About us

The Risks of Unsecured Telecom Supply Chains

Telecom supply chains are increasingly being targeted by cybercriminals, nation-state actors, and other malicious entities seeking to exploit vulnerabilities in the telecom infrastructure. Some of the key risks associated with telecom supply chains include:

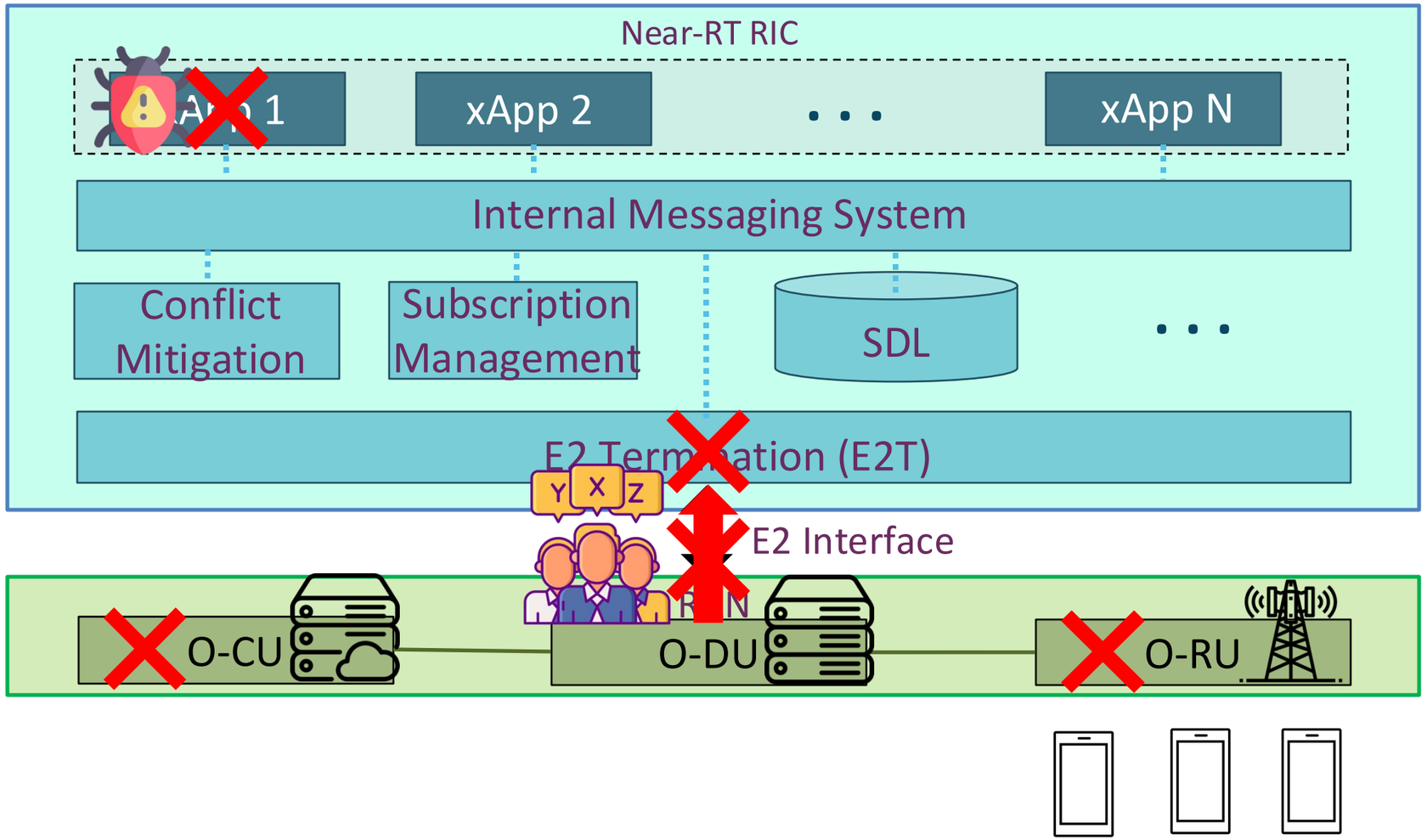
- Third-Party Vendor Risks

One of the biggest challenges telecom operators face in securing their supply chains is the risk posed by third-party vendors. Suppliers who provide telecom hardware, software, and services may not always meet the same stringent security standards as the operators themselves. If a supplier's systems are compromised, it can create vulnerabilities in the entire telecom network.

- Hardware and Software Vulnerabilities

Securing Telecom Supply Chains: Mitigating Risks in the Telecom Ecosystem
www.p1sec.com/blog/securing-telecom-supply-chains-mitigating-risks-in-the-telecom-ecosystem

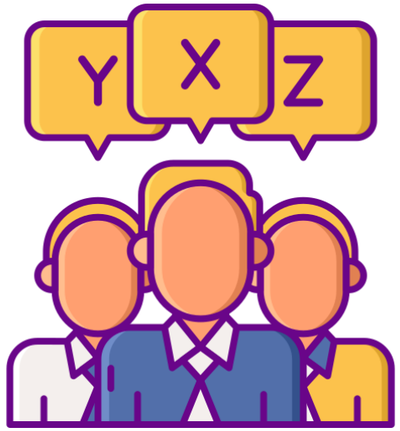
Threat Demonstration – Heterogeneous RAN



O-RAN Vulnerability Sources

Malicious User

Supply Chain Risk



Heterogeneity of RAN Nodes

O-RAN Vulnerability Sources

Malicious User

Supply C

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.

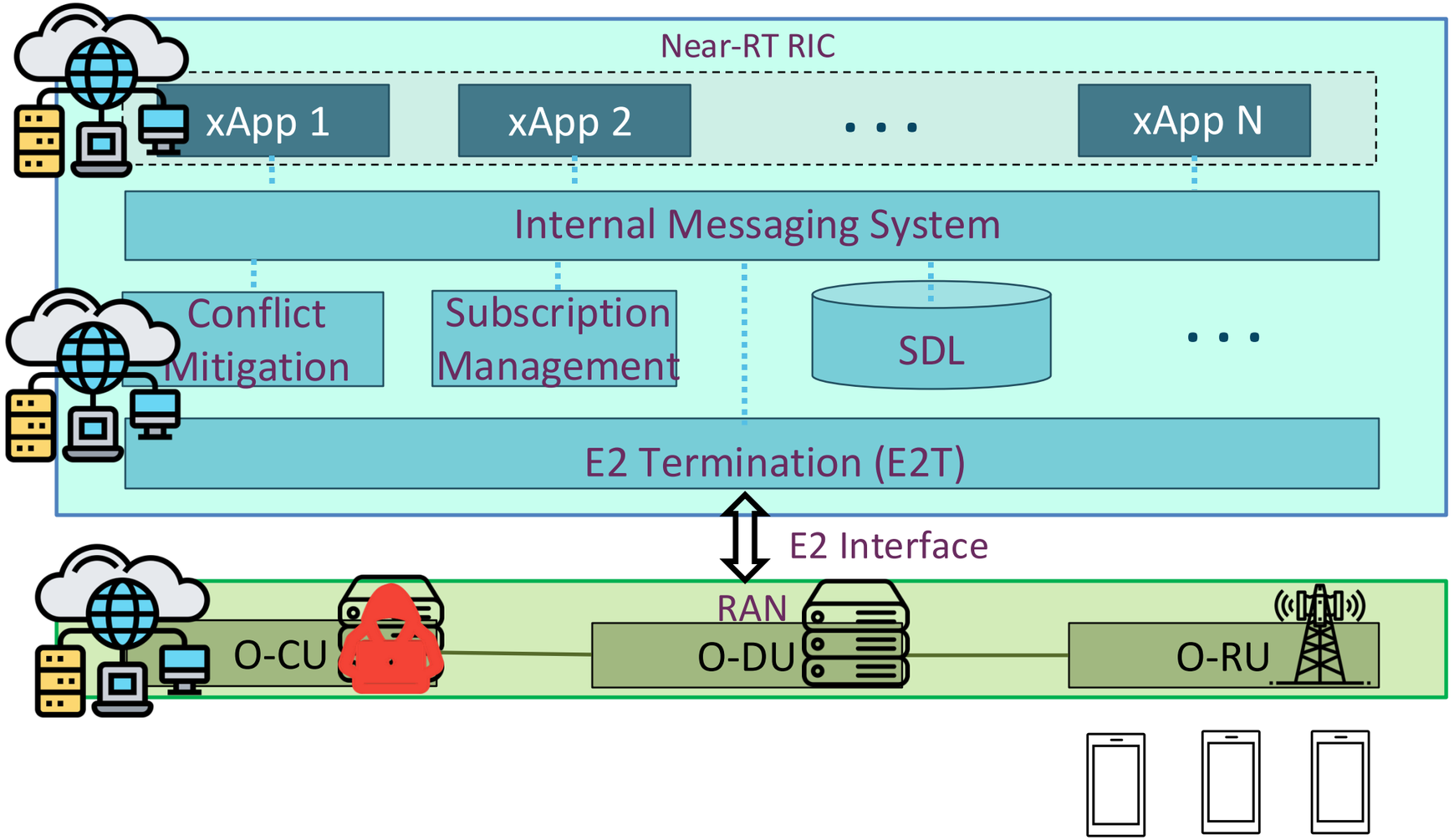
6.17.2 Solution details

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

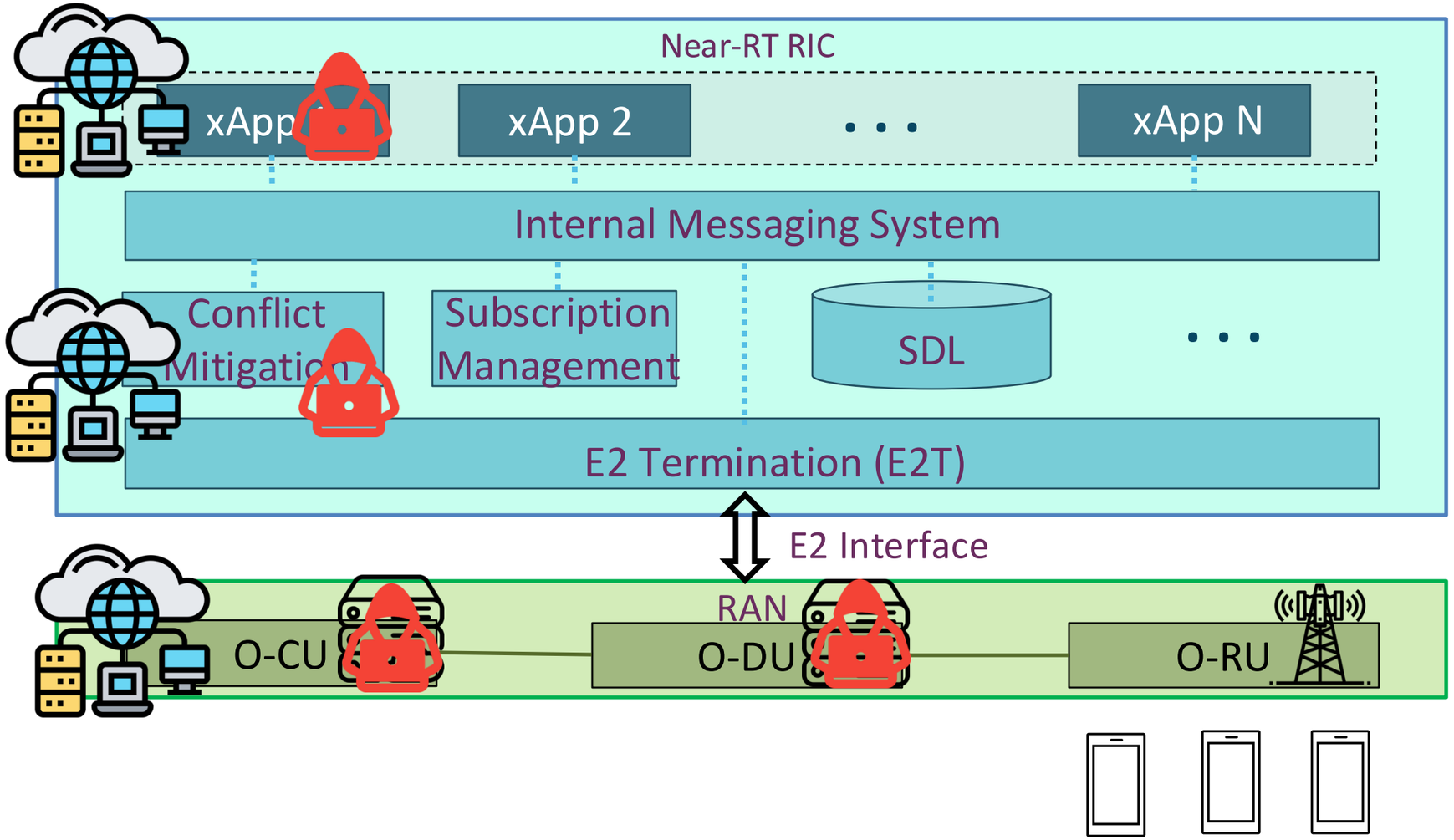
1. Validate received values for validity and range
2. Provide rate limiting on E2 interface to prevent resource exhaustion and DoS
3. Implement security logging for each of the above failure events

O-RAN Study on Security for Near Real Time RIC and xApps
www.o-ran.org/specifications

Threat Demonstration – Cloud Tenants



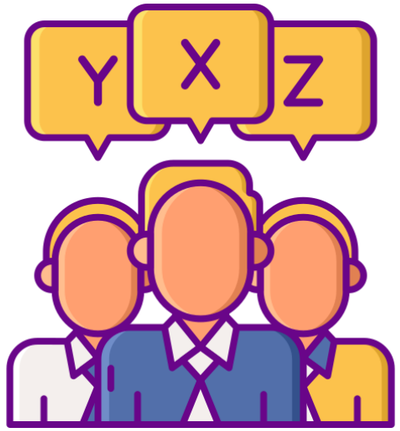
Threat Demonstration – Cloud Tenants



O-RAN Vulnerability Sources

Malicious User

Supply Chain Risk

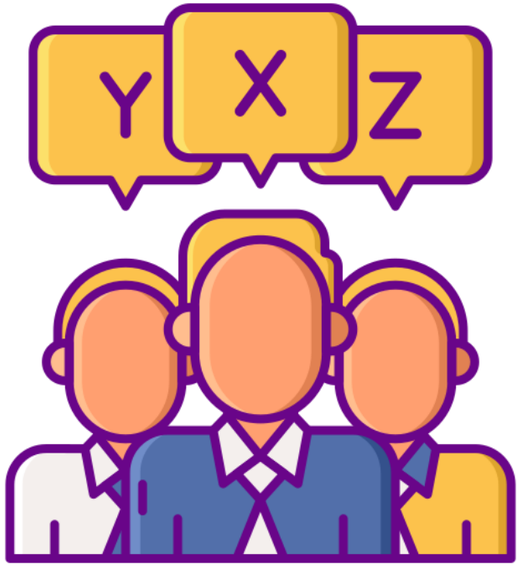


Heterogeneity of RAN Nodes

Cloud Tenants

O-RAN Vulnerability Sources

Malicious User



Heterogeneity of RAN Nodes

O-RAN SOFTWARE COMMUNITY

SD-RAN

OPEN AIR INTERFACE

- Identified **19 flaws** in RIC components and xApps that can lead to DoS of the RIC, and **7 flaws** in CU components that can lead to DoS of the whole cell.
- **22 CVEs** have been assigned to track all 26 issues.
  - CVE-2023-52724, -52725, -52726, -52727, -52728
  - CVE-2024-25377, -29420, -34043, 34044, -34045, -34046, -34047, -34048, -34049, -34050, -57330, -57331, -57332, -57333, 57334
  - CVE-2025-45420, -45421

Assertion Failures	Memory Issue	Runtime Panic	Logical Error
• CVE-2024-57330 (OAI DU) • CVE-2024-57331 (OAI CU) • CVE-2024-57332 (OAI CU) • CVE-2024-57333 (OAI CU) • CVE-2025-45421 (OAI CU)	• CVE-2024-57334 (OAI CU) • CVE-2025-45420 (OAI CU) • CVE-2024-34043 (O-RAN-SC Dependency) • CVE-2024-34044 (O-RAN-SC E2T) • CVE-2024-25377 (O-RAN-SC xApp)	• CVE-2024-34047 (O-RAN-SC E2Manager) • CVE-2024-34048 (O-RAN-SC E2Manager) • CVE-2025-30077 (SD-RAN Dependency) • CVE-2023-52727 (SD-RAN Dependency) • CVE-2023-52728 (SD-RAN Dependency) • CVE-2024-29420 (O-RAN-SC xApp) • CVE-2024-34049 (SD-RAN xApp) • CVE-2024-34050 (SD-RAN xApp) • CVE-2023-52724 (SD-RAN xApp)	• CVE-2023-52726 (SD-RAN xApp) • CVE-2023-52725 (SD-RAN xApp)

- CU/DU
- RIC Platforming/Dependency
- RIC xApp

Assertion Failures	Memory Issue	Runtime Panic	Logical Error
• CVE-2024-57330 (OAI DU) • CVE-2024-57331 (OAI CU) • CVE-2024-57332 (OAI CU) • CVE-2024-57333 (OAI CU) • CVE-2025-45421 (OAI CU)	• CVE-2024-57334 (OAI CU) • CVE-2025-45420 (OAI CU) • CVE-2024-34043 (O-RAN-SC Dependency) • CVE-2024-34044 (O-RAN-SC E2T) • CVE-2024-25377 (O-RAN-SC xApp)	• CVE-2024-34047 (O-RAN-SC E2Manager) • CVE-2024-34048 (O-RAN-SC E2Manager) • CVE-2025-30077 (SD-RAN Dependency) • CVE-2023-52727 (SD-RAN Dependency) • CVE-2023-52728 (SD-RAN Dependency) • CVE-2024-29420 (O-RAN-SC xApp) • CVE-2024-34049 (SD-RAN xApp) • CVE-2024-36650 (SD-RAN xApp) • CVE-2023-52729 (SD-RAN xApp)	• CVE-2023-52726 (SD-RAN xApp) • CVE-2023-52725 (SD-RAN xApp)

code

- CU/DU
- RIC Platforming/Dependency
- RIC xApp

Dependency management:

In-house: buggy, undertested

Third-party: security, backdoor

Assertion Failures	Memory Issue	Runtime Panic	Logical Error
• CVE-2024-57330 (OAI DU) • CVE-2024-57331 (OAI CU) • CVE-2024-57332 (OAI CU) • CVE-2024-57333 (OAI CU) • CVE-2025-45421 (OAI CU)	• CVE-2024-57334 (OAI CU) • CVE-2025-45420 (OAI CU) • CVE-2024-34043 (O-RAN-SC Dependency) • CVE-2024-34044 (O-RAN-SC E2T) • CVE-2024-25377 (O-RAN-SC xApp)	• CVE-2024-34047 (O-RAN-SC E2Manager) • CVE-2024-34048 (O-RAN-SC E2Manager) • CVE-2025-30077 (SD-RAN Dependency) • CVE-2023-52727 (SD-RAN Dependency) • CVE-2023-52728 (SD-RAN Dependency) • CVE-2024-29420 (O-RAN-SC xApp)	• CVE-2023-52726 (SD-RAN xApp) • CVE-2023-52725 (SD-RAN xApp)

```
cString := C.CString(resp.Gnb.RanFunctions[counter].RanFunctionDefinition)
defer C.free(unsafe.Pointer(cString)) // Free the allocated C string when done
// Call the C function
determine := 4 //2 for format1 by name

result := C.encode_action_Definition(cString, C.int(determine))
```

Vulnerable Cgo call

Are Commercial O-RAN Systems Safe?

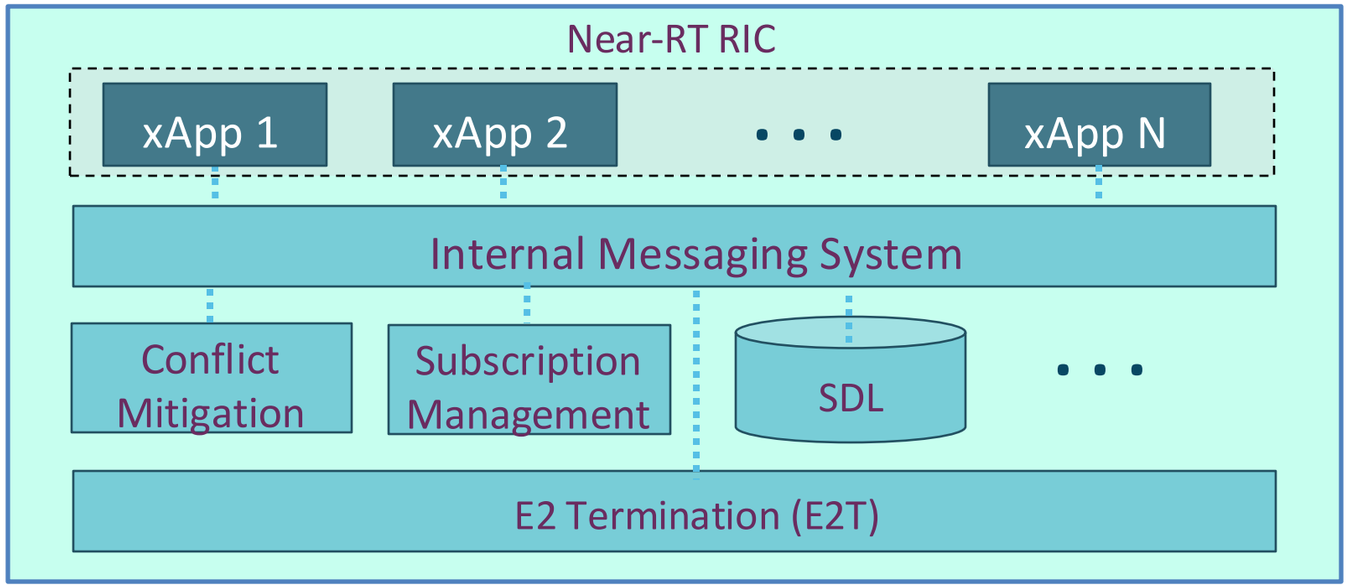
- Closed-source API testing
- Zero-trust design

- 4 implementation-level issue
- Long requests, unexpected formats (json when string expected)

Openness & virtualization creates opportunity, but also vulnerability

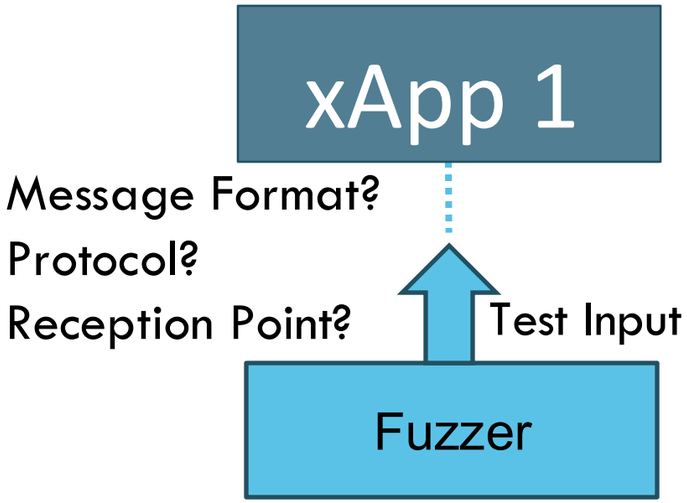
Limitations of Existing Testing Approaches

- Existing tools (AFLNET, BooFuzz, Restler, Frizzer) test **one program** at a time



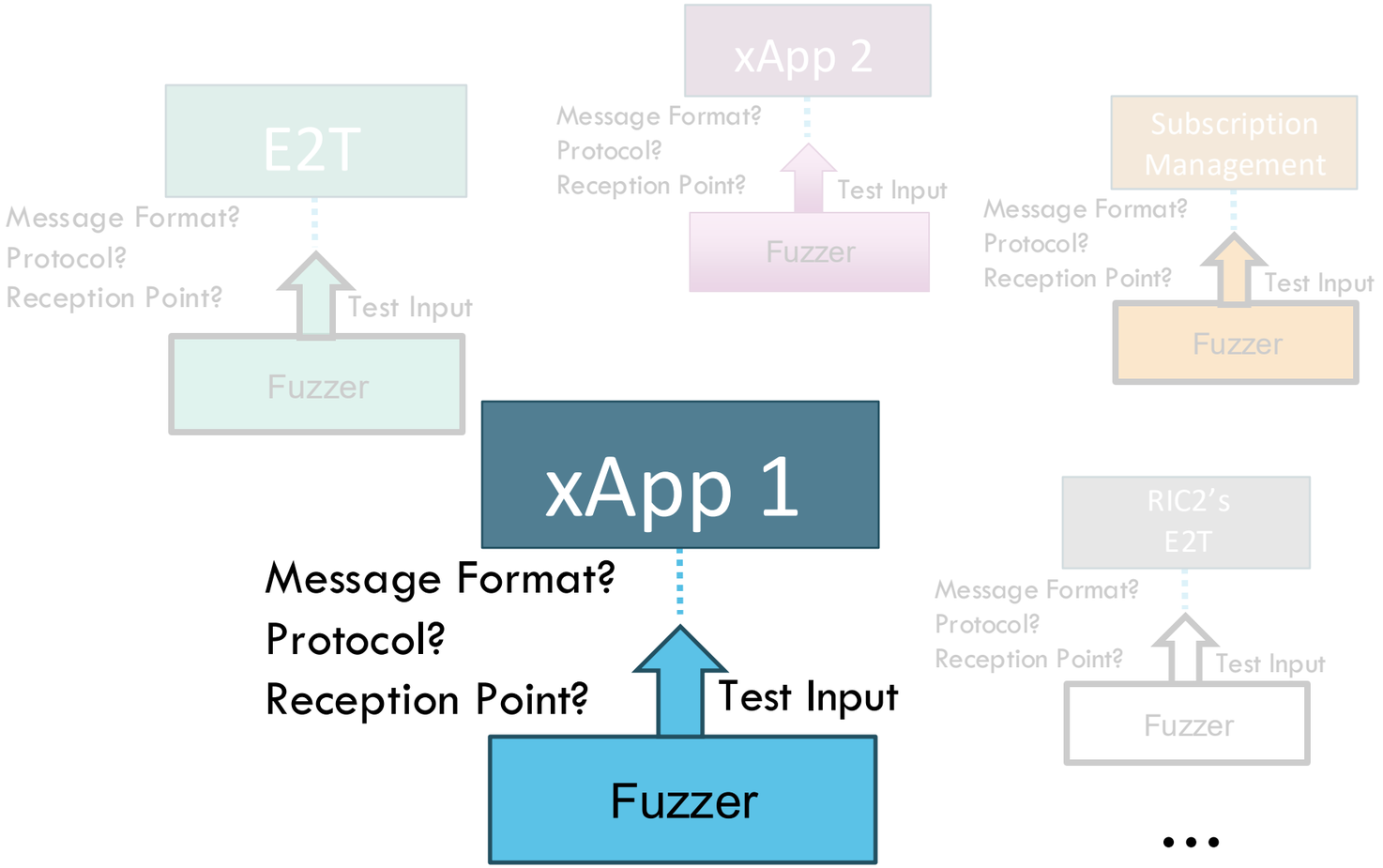
Limitations of Existing Testing Approaches

- Existing tools (AFLNET, BooFuzz, Restler, Frizzer) test **one program** at a time
- Requires details about the **expected message, dependencies, protocols, …**
- **Internal details vary across different implementations**



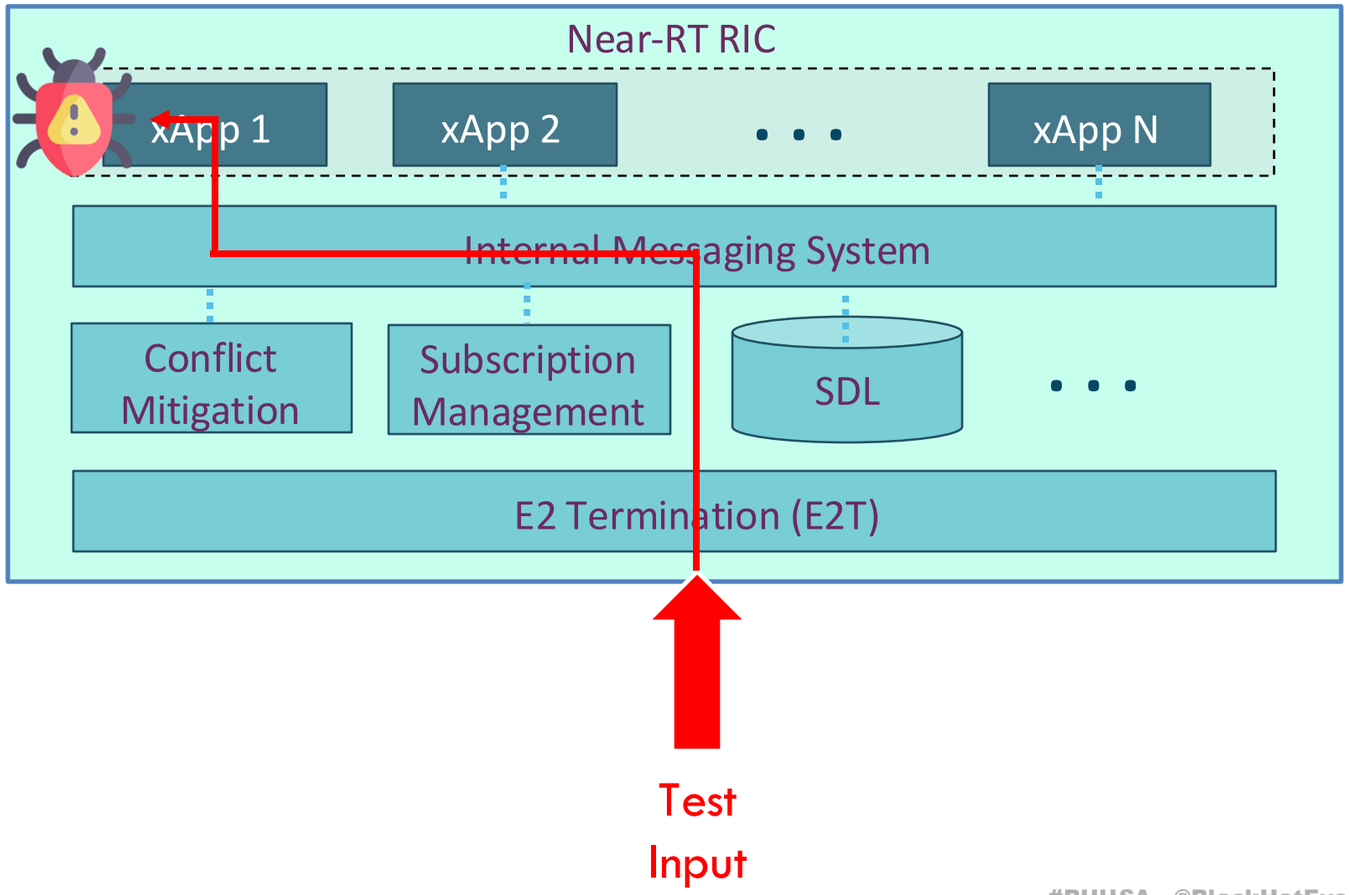
Limitations of Existing Testing Approaches

- Existing tools (AFLNET, BooFuzz, Restler, Frizzer) test **one program** at a time
- Requires details about the **expected message, dependencies, protocols, …**
- **Internal details vary across different implementations**

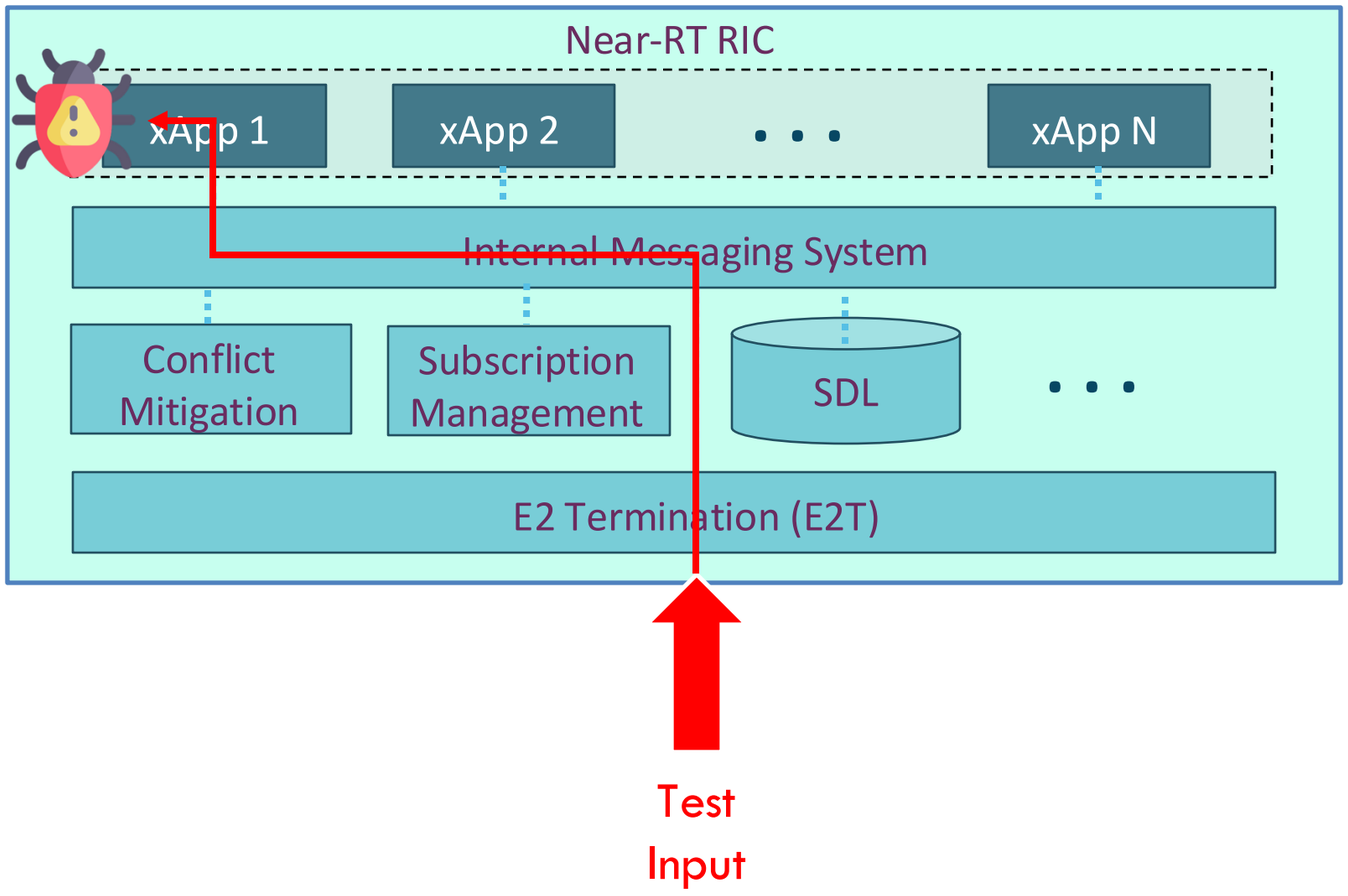


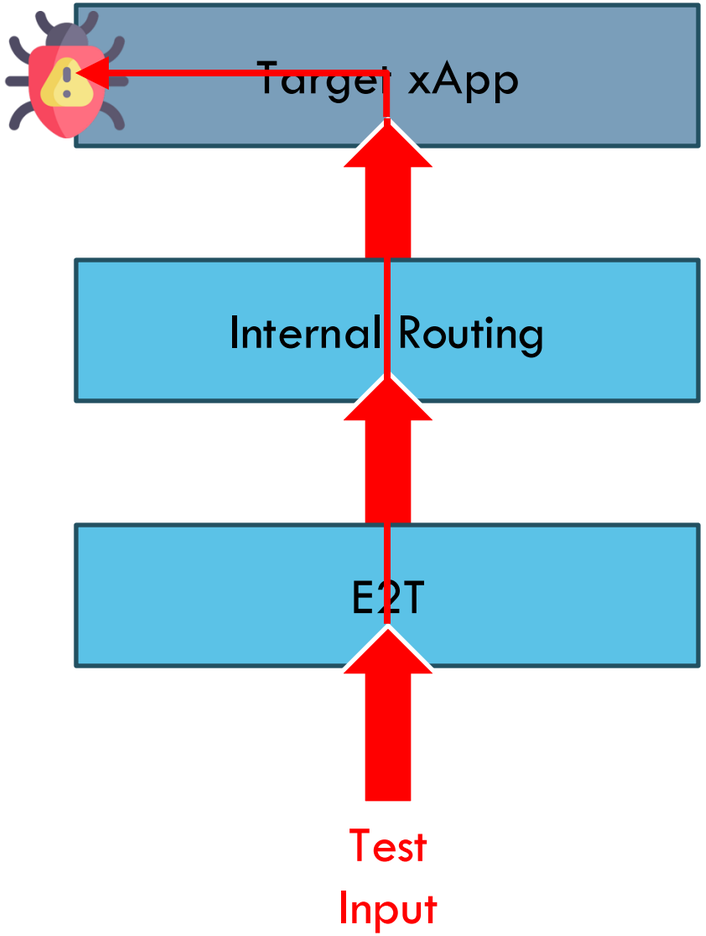
Our Approach: End-to-End Testing

- Send test inputs only through public interface.
- Automatic test generation for the **standardized protocol**
- **Scalable** to all implementations
- All found bugs are exploitable from a misbehaving RAN



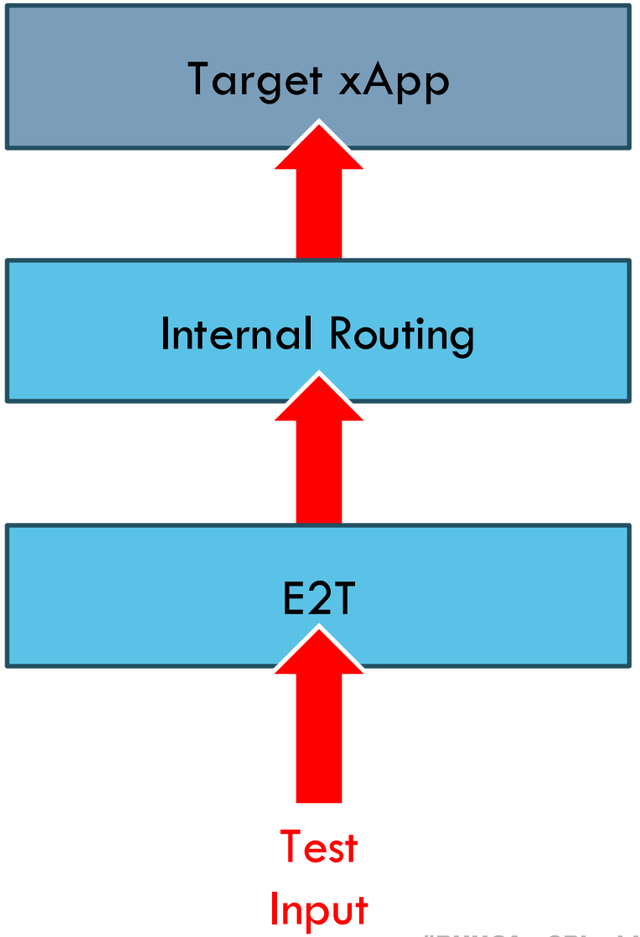
Problem Formulation





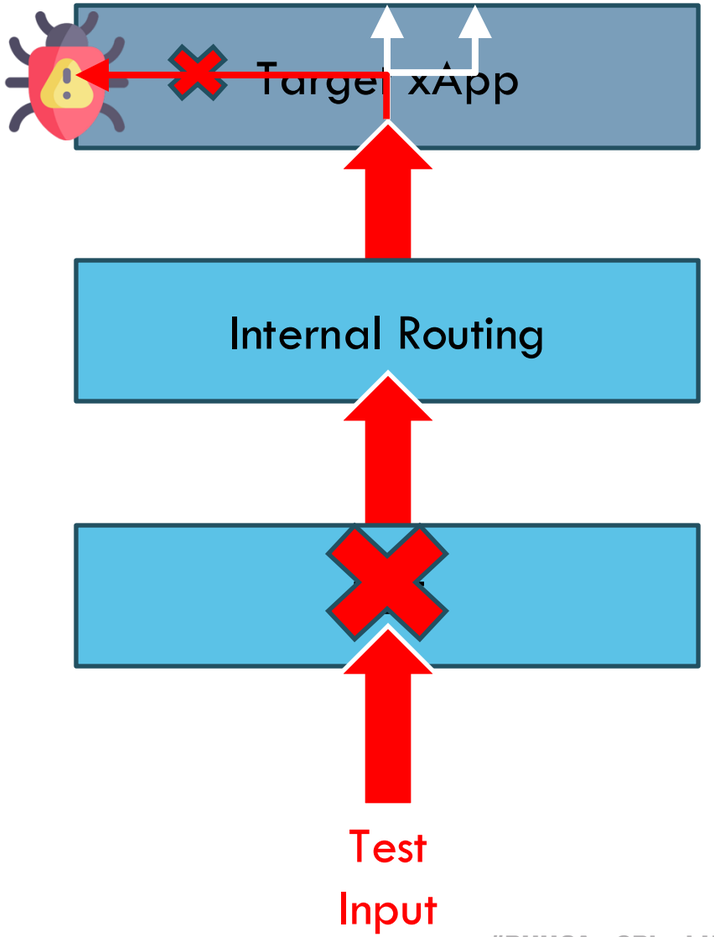
Challenge 1: Generating Targeted and Meaningful Test Inputs

```
var requestID int32
for _, v := range request.GetProtocolIes() {
    if v.Id == int32(v2.ProtocolIeIDRicrequestID) {
        requestID = v.GetValue().GetRicrequestId().GetRicRequestorId()
        break
    }
}
streamID := stream.ID(requestID)
stream, ok := c.streams.Get(streamID)
if !ok {
    return errors.NewNotFound("stream %s not found", streamID)
}
```



Challenge 1: Generating Targeted and Meaningful Test Inputs

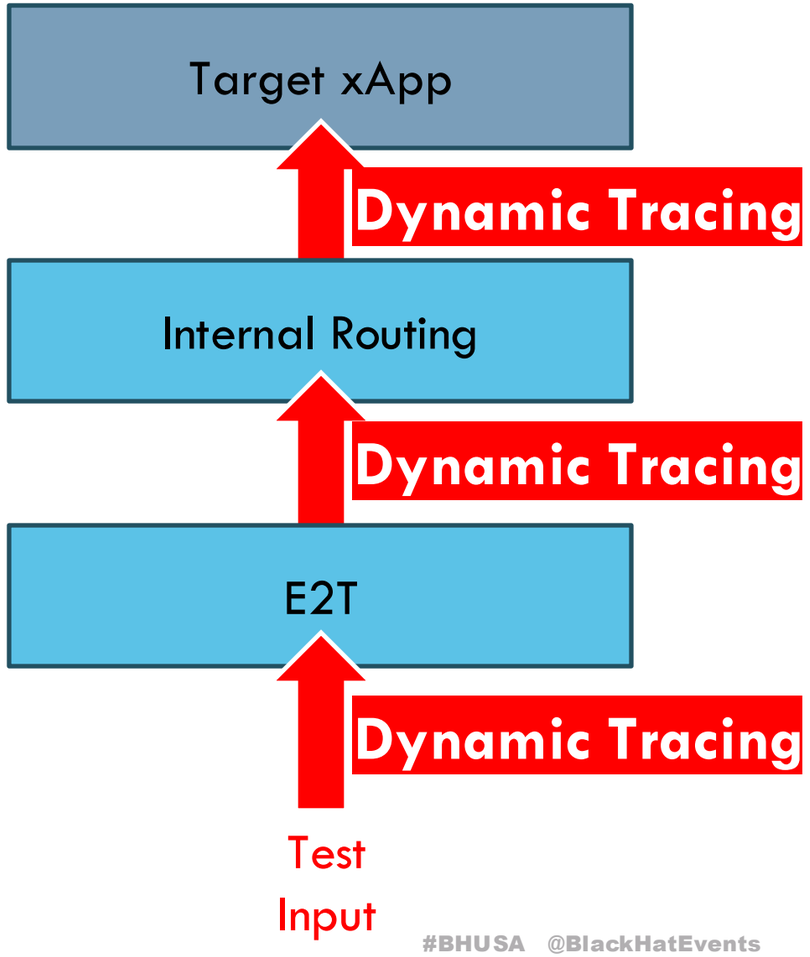
- **Challenge:** generate inputs that can **reach the target** components (avoid under-constraint) while **maintain variability** for effective testing (avoid over-constraint).



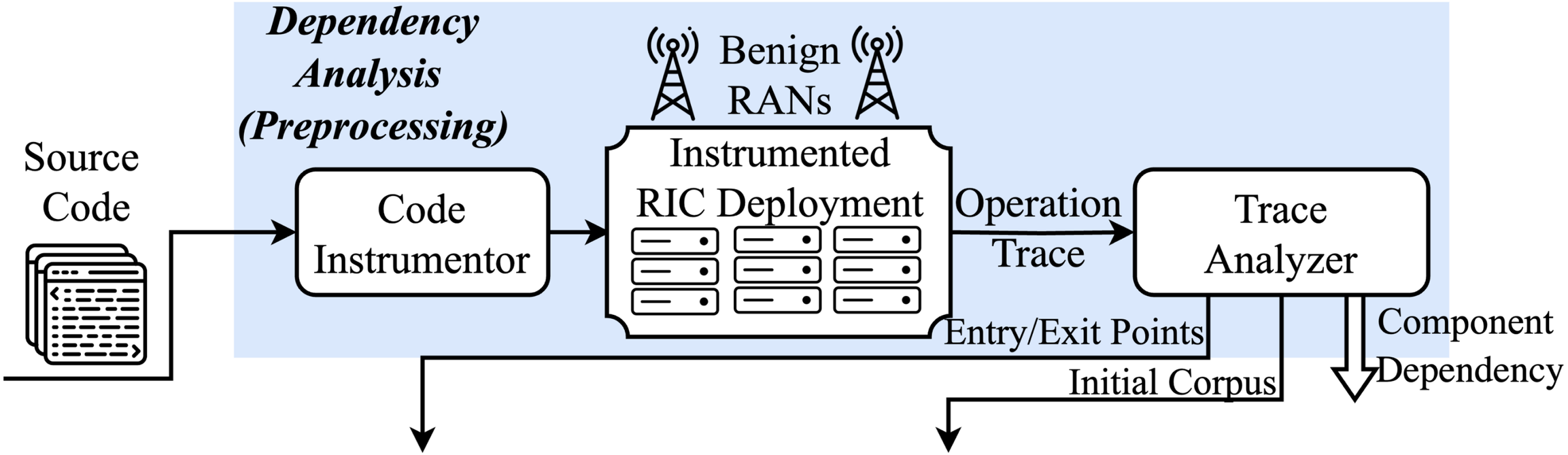
Solution 1: Layered Testing Approach

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

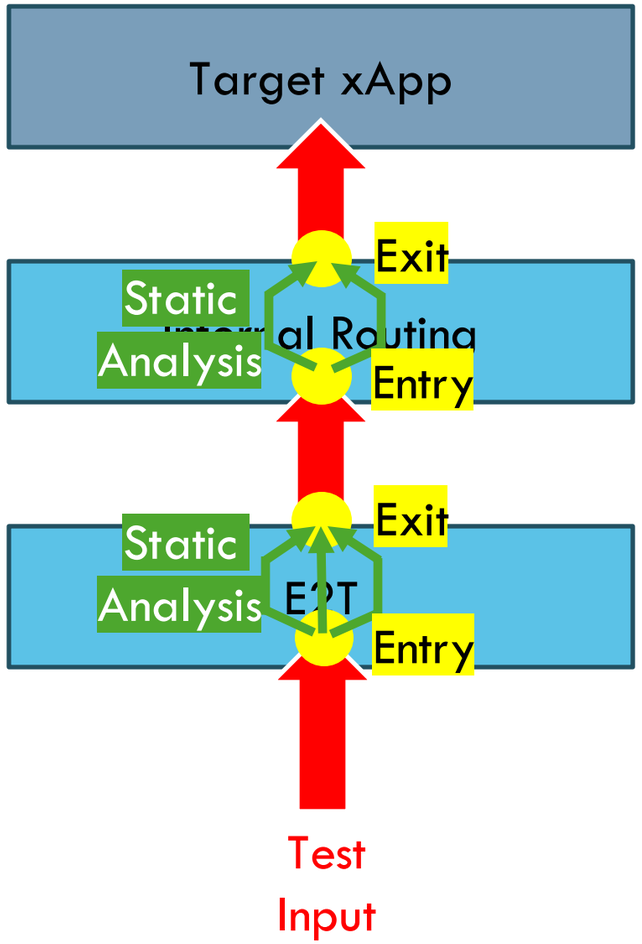
- **Challenge:** How can we find these **layer-dependencies** between components?
- **Solution: Dynamic tracing**



Challenge 2: Enumerate Appropriate Constraints



Scalability Challenges in Static Analysis



GitHub | onosproject/onos-e2t | master | ADD

ignore files/dirs comma separated

Go XML Protocol Buffers Markdown YAML
Shell Dockerfile Makefile Plain Text

Files	Lines	Blanks	Comments	Lines of Code
330	661435	94163	76801	490471

```
onos-e2t % grep -Er --exclude-dir={test,api} \
          '^\s*func\s+(\([^\)]*\)\s*)?[a-zA-Z_][a-zA-Z0-9_]*\s*\(' \
           ./ | wc -l
```

```
onos-e2t % grep -Er --exclude-dir={test,api} \
           '^\s*func\s+(\([^\)]*\)\s*)?[a-zA-Z_][a-
            ./ | wc -l
      6152
```

Challenge & Solution 3: Efficient Static Analysis

Solution:

- Program Dependency Graph (PDG)-based view of control dependencies to find critical conditions
- Backward dataflow analysis to generate constraints on the input message
- **Selectively analyze** functions validating inputs, ignoring generic functions (e.g., network operations, data retrieval)

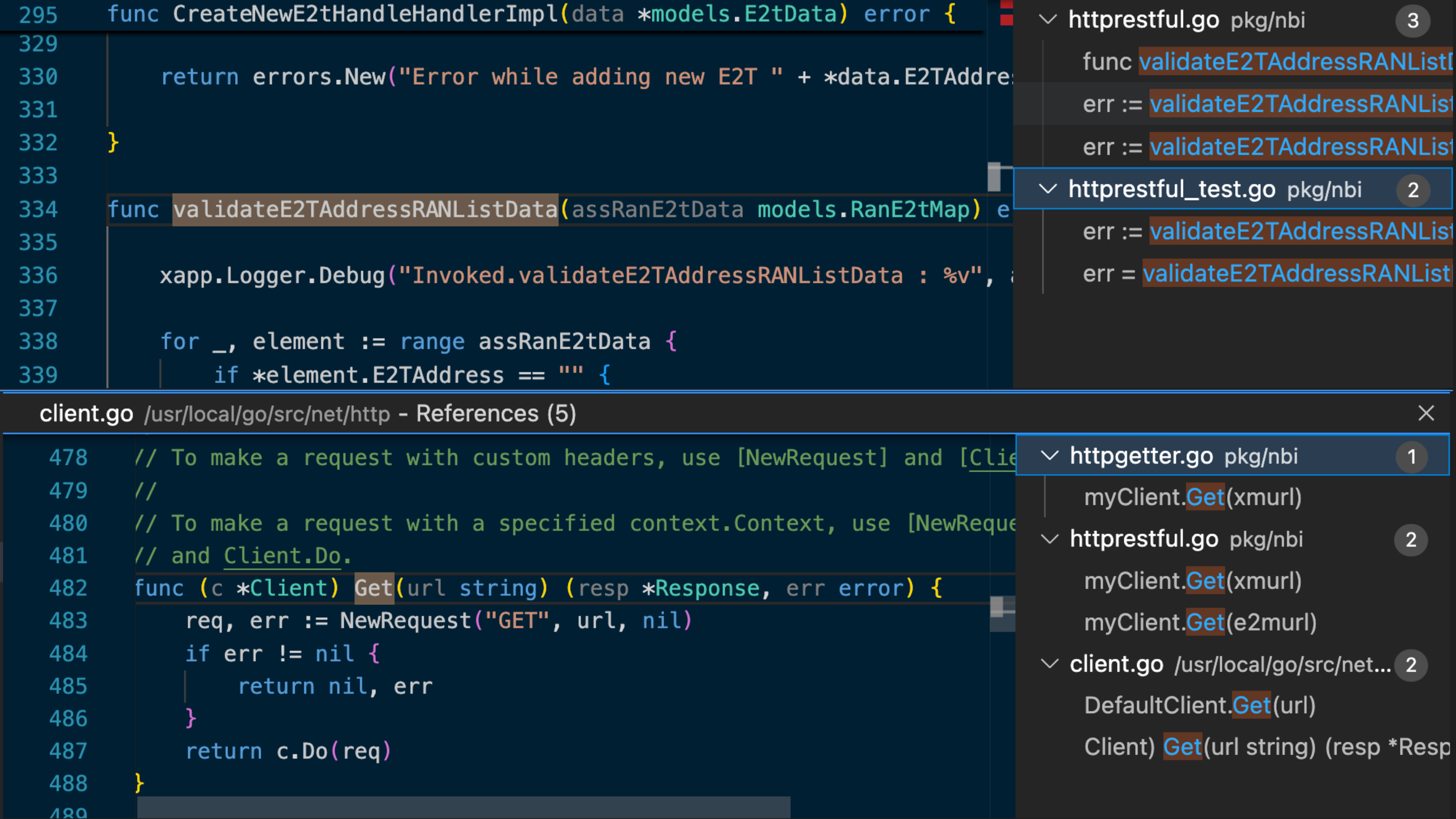
```
err := validateE2TAddressRANListData(data)
```

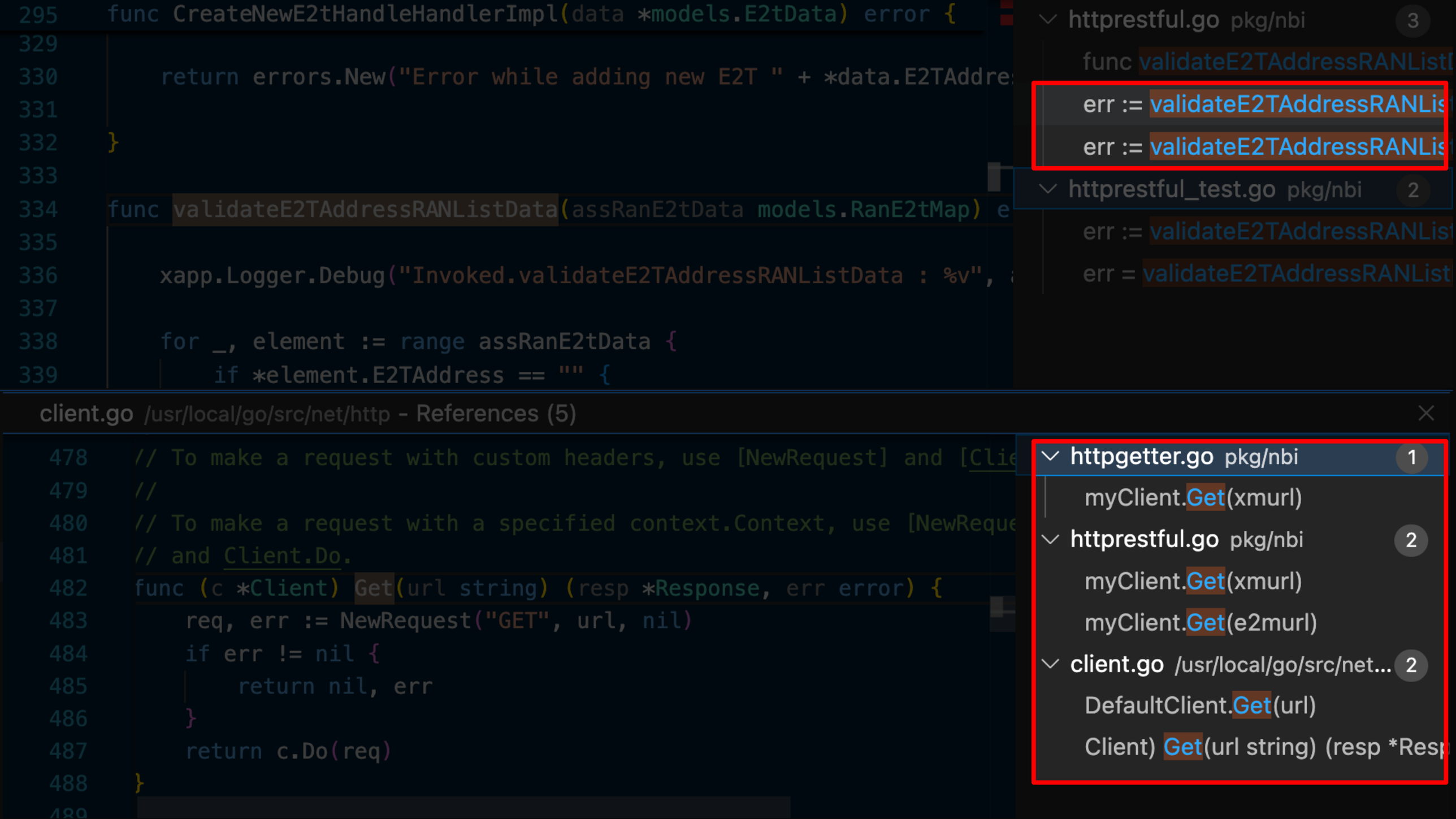
```
r, err := myClient.Get(xmurl)
```

How to Discern Generic/Validating Functions?

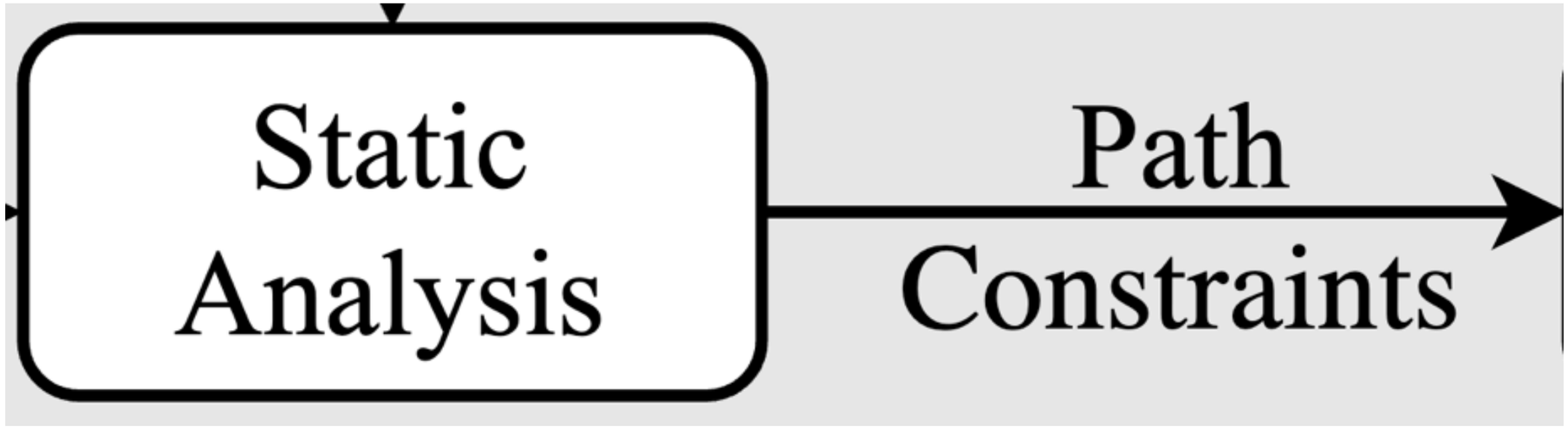
- Name?
- LLM?
- Control Flow?

```
func AssociateRanToE2THandlerImpl(data models.RanE2tMap) error {
func DisassociateRanToE2THandlerImpl(data models.RanE2tMap) error {
func DeleteE2tHandleHandlerImpl(data *models.E2tDeleteData) error {
func DumpDebugData() (models.Debuginfo, error) {
func httpGetXApps(xmurl string) (*[]rtmgr.XApp, error) {
func httpGetE2TList(e2murl string) (*[]rtmgr.E2tIdentity, error) {
func PopulateE2TMap(e2tDataList *[]rtmgr.E2tIdentity, e2ts map[string]rtmgr.E2TInstance,
func retrieveStartupData(xmurl string, nbiif string, fileName string, configfile string,
func (r *HttpRestful) Initialize(xmurl string, nbiif string, fileName string, configfile
func (r *HttpRestful) Terminate() error {
func addSubscription(subs *rtmgr.SubscriptionList, xappSubData *models.XappSubscriptionD
func delSubscription(subs *rtmgr.SubscriptionList, xappSubData *models.XappSubscriptionD
func updateSubscription(data *rtmgr.XappList) {
func PopulateSubscription(sub_list xfmodel.SubscriptionList) {
func Adddelrmrroute(routelist models.Routelist, rtflag bool) error {
func checkrepeatedroute(data string) bool {
func NewHttpGetter() *HttpGetter {
func fetchAllXApps(xmurl string) (*[]rtmgr.XApp, error) {
func (g *HttpGetter) Initialize(xmurl string, nbiif string, fileName string, configfile
func (g *HttpGetter) Terminate() error {
func LaunchRest(nbiif string) {
func NewFile() *File {
func (f *File) ReadAll(file string) (*rtmgr.RicComponents, error) {
func (f *File) WriteAll(file string, rcs *rtmgr.RicComponents) error {
func (f *File) WriteXApps(file string, xApps *[]rtmgr.XApp) error {
func (f *File) WriteNewE2TInstance(file string, E2TInst *rtmgr.E2TInstance, meiddata str
func (f *File) WriteAssRANToE2TInstance(file string, rane2tmap models.RanE2tMap) error {
func (f *File) WriteDisAssRANFromE2TInstance(file string, disassranmap models.RanE2tMap)
func (f *File) WriteDeleteE2TInstance(file string, E2TInst *models.E2tDeleteData) error
func GetSdl(sdlName string) (Engine, error) {
func (params *RMRParams) String() string {
func NewRmrPush() *RmrPush {
func (c *RmrPush) Initialize(ip string) error {
func (c *RmrPush) Terminate() error {
func (c *RmrPush) AddEndpoint(ep *rtmgr.Endpoint) error {
func (c *RmrPush) DeleteEndpoint(ep *rtmgr.Endpoint) error {
func (c *RmrPush) UpdateEndpoints(rcs *rtmgr.RicComponents) {
func (c *RmrPush) DistributeAll(policies *[]string) error {
func (c *RmrPush) send_sync(ep *rtmgr.Endpoint, policies *[]string, call_id int) {
func (c *RmrPush) send_data(ep *rtmgr.Endpoint, policies *[]string, call_id int) bool {
func (c *RmrPush) CheckEndpoint(payload string) (ep *rtmgr.Endpoint) {
func (c *RmrPush) CreateEndpoint(rmrsrc string) (ep *string, whid int) {
func (c *RmrPush) DistributeToEp(policies *[]string, ep string, whid int) error {
```

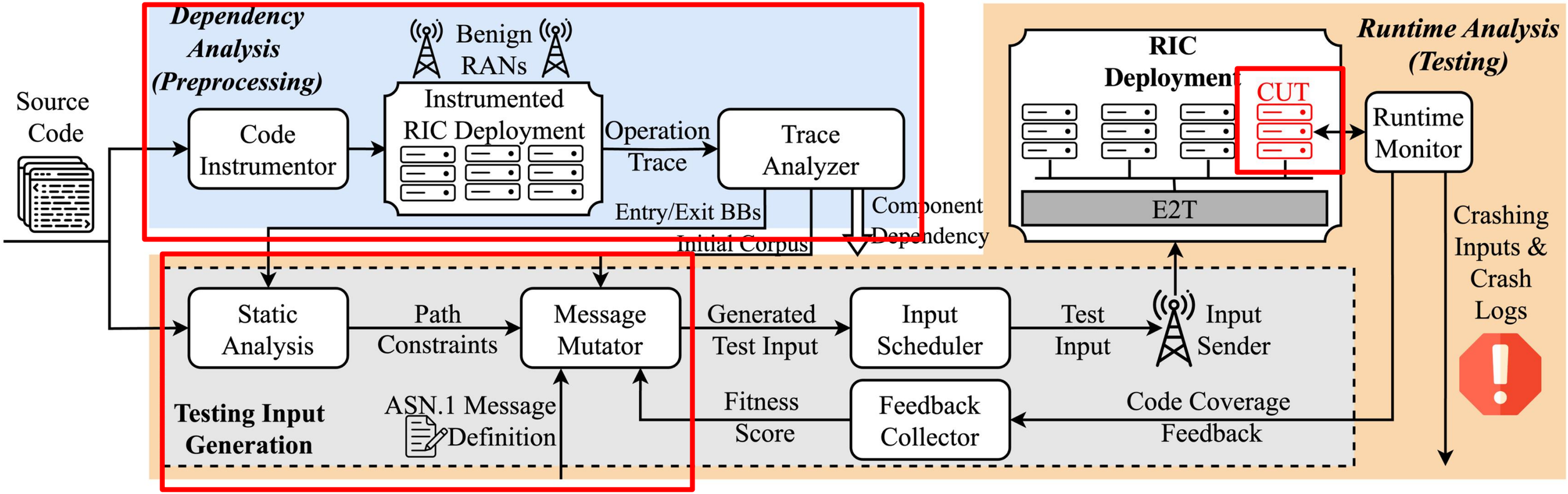

295 func CreateNewE2tHandleHandlerImpl(data *models.E2tData) error {
329
330 return errors.New("Error while adding new E2T " + *data.E2TAddre
331
332 }
333
334 func validateE2TAddressRANListData(assRanE2tData models.RanE2tMap) e
335
336 xapp.Logger.Debug("Invoked.validateE2TAddressRANListData : %v",
337
338 for _, element := range assRanE2tData {
339 if *element.E2TAddress == "" {
httprestful.go pkg/nbi 3
func validateE2TAddressRANListI
err := validateE2TAddressRANLis
err := validateE2TAddressRANLis
httprestful_test.go pkg/nbi 2
err := validateE2TAddressRANLis
err = validateE2TAddressRANList
client.go /usr/local/go/src/net/http - References (5)
478 // To make a request with custom headers, use [NewRequest] and [Clie
479 //
480 // To make a request with a specified context.Context, use [NewReque
481 // and Client.Do.
482 func (c *Client) Get(url string) (resp *Response, err error) {
483 req, err := NewRequest("GET", url, nil)
484 if err != nil {
485 return nil, err
486 }
487 return c.Do(req)
488 }
httpgetter.go pkg/nbi 1
myClient.Get(xmurl)
httprestful.go pkg/nbi 2
myClient.Get(xmurl)
myClient.Get(e2murl)
client.go /usr/local/go/src/net... 2
DefaultClient.Get(url)
Client) Get(url string) (resp *Resp


func CreateNewE2tHandleHandlerImpl(data *models.E2tData) error {
return errors.New("Error while adding new E2T " + *data.E2TAddre
}
func validateE2TAddressRANListData(assRanE2tData models.RanE2tMap) e
xapp.Logger.Debug("Invoked.validateE2TAddressRANListData : %v",
for _, element := range assRanE2tData {
if *element.E2TAddress == "" {
httprestful.go pkg/nbi 3
func validateE2TAddressRANListI
err := validateE2TAddressRANLis
err := validateE2TAddressRANLis
httprestful_test.go pkg/nbi 2
err := validateE2TAddressRANLis
err = validateE2TAddressRANList
client.go /usr/local/go/src/net/http - References (5)
// To make a request with custom headers, use [NewRequest] and [Clie
//
// To make a request with a specified context.Context, use [NewReque
// and Client.Do.
func (c *Client) Get(url string) (resp *Response, err error) {
req, err := NewRequest("GET", url, nil)
if err != nil {
return nil, err
}
return c.Do(req)
}
httpgetter.go pkg/nbi 1
myClient.Get(xmurl)
httprestful.go pkg/nbi 2
myClient.Get(xmurl)
myClient.Get(e2murl)
client.go /usr/local/go/src/net... 2
DefaultClient.Get(url)
Client) Get(url string) (resp *Resp

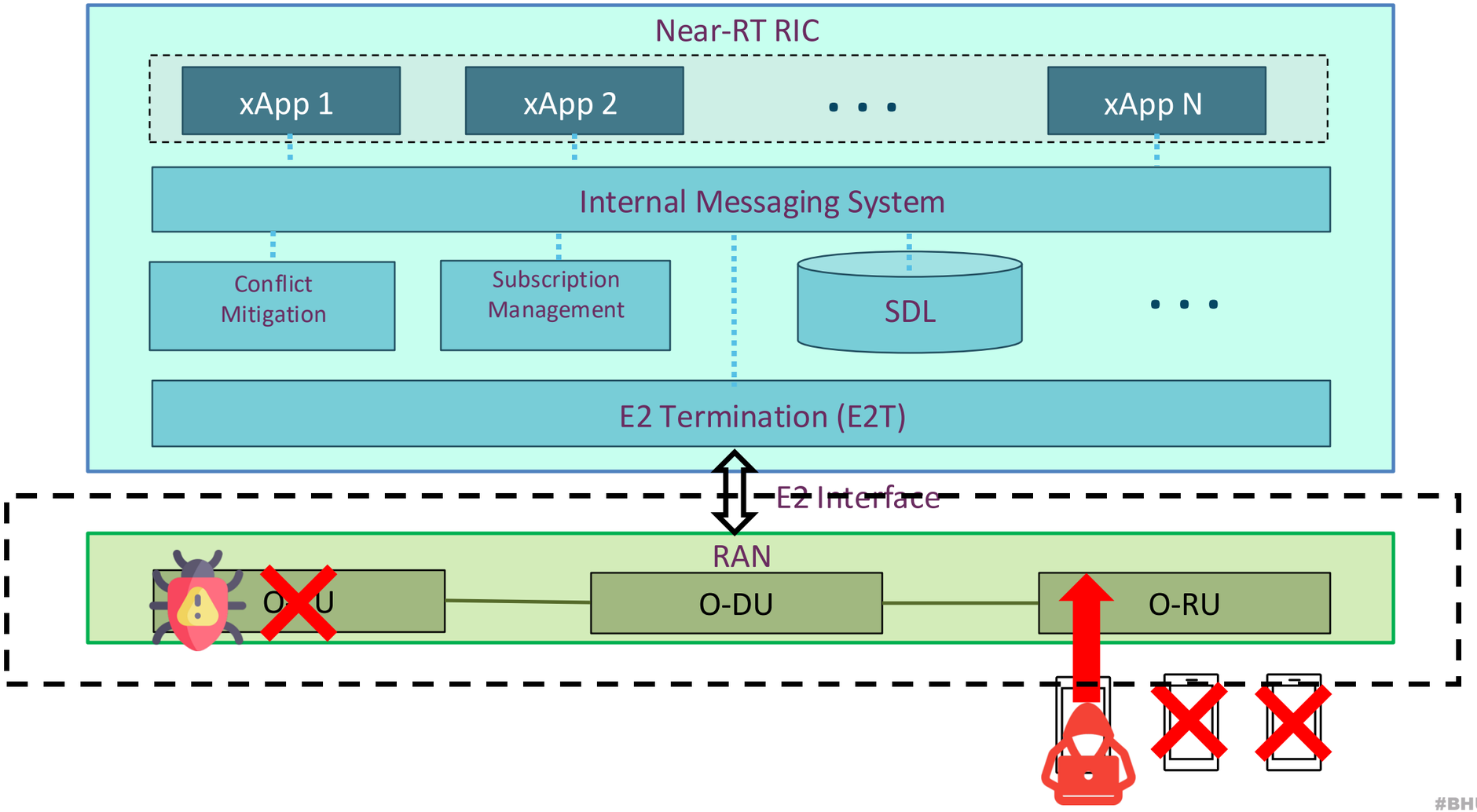
Architecture



Architecture

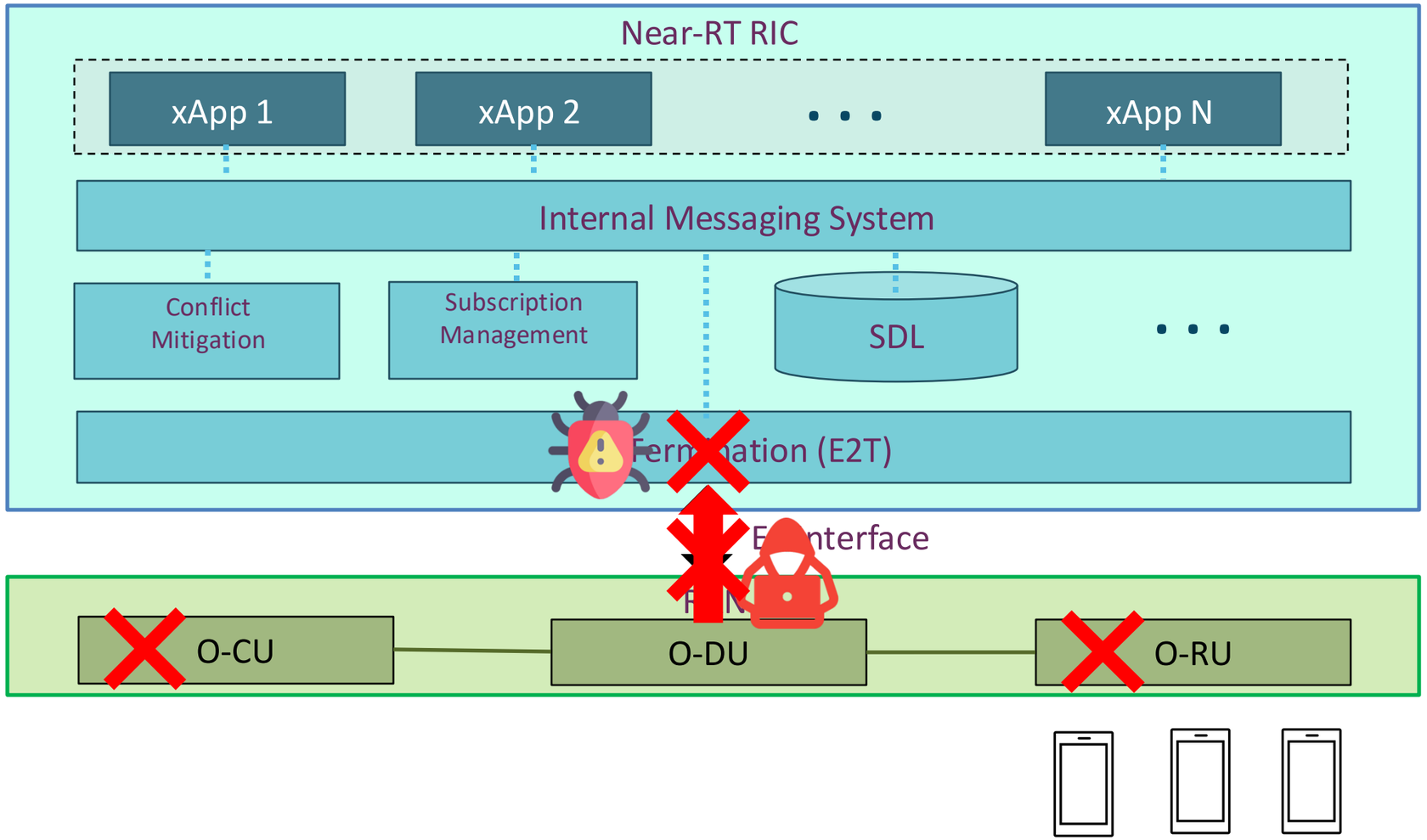


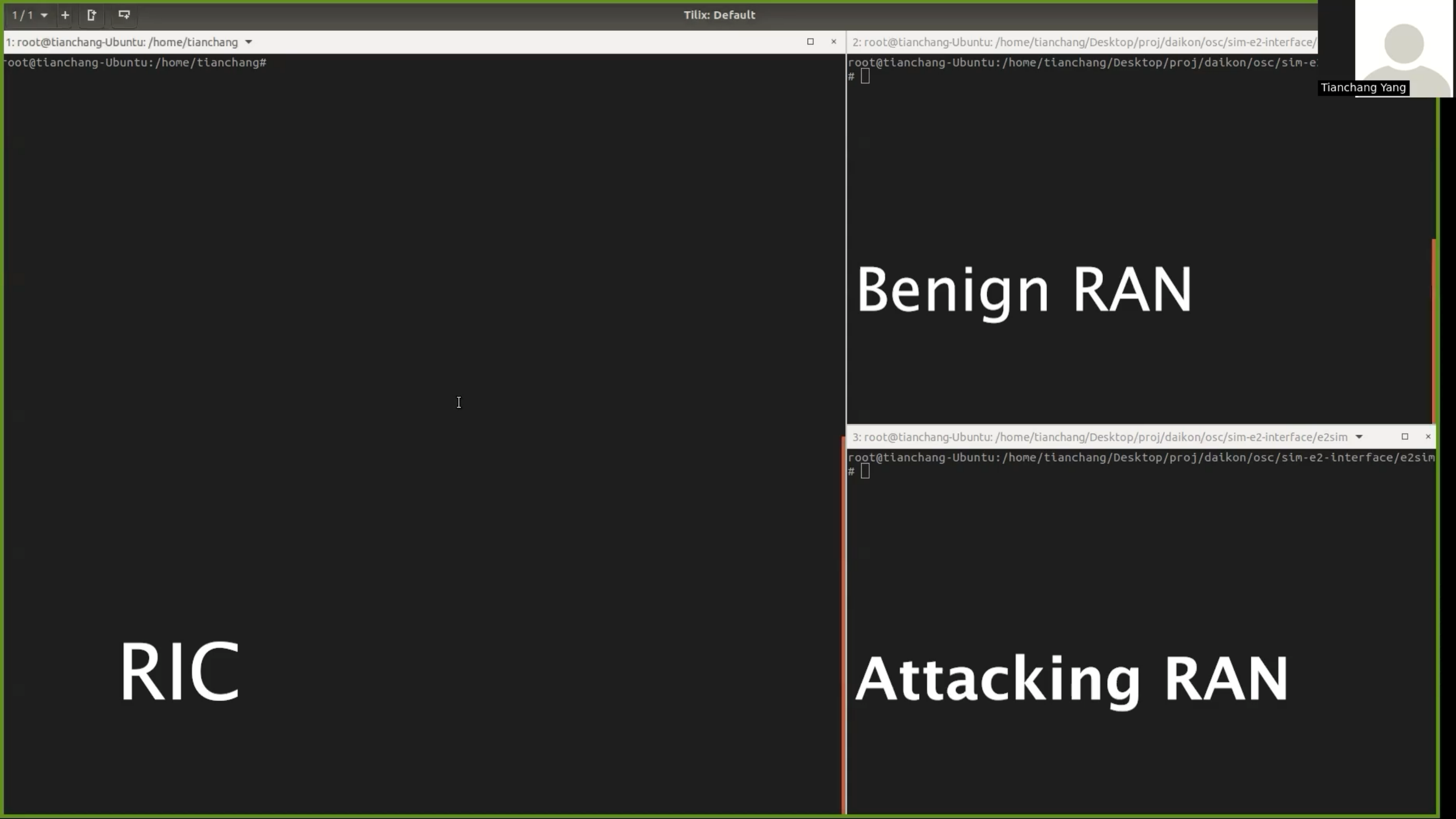
RAN Vulnerability Demo




Benign RAN
(CU & DU)
Attacker
UE
Benign RAN
(RU)
RAN
RU

RIC Vulnerability Demo




1 / 1
Tilix: Default
1: root@tianchang-Ubuntu: /home/tianchang
root@tianchang-Ubuntu:/home/tianchang#
2: root@tianchang-Ubuntu: /home/tianchang/Desktop/proj/daikon/osc/sim-e2-interface/
root@tianchang-Ubuntu:/home/tianchang/Desktop/proj/daikon/osc/sim-e
#
Tianchang Yang
Benign RAN
3: root@tianchang-Ubuntu: /home/tianchang/Desktop/proj/daikon/osc/sim-e2-interface/e2sim
root@tianchang-Ubuntu:/home/tianchang/Desktop/proj/daikon/osc/sim-e2-interface/e2sim
#
RIC
Attacking RAN

Final Thoughts

- O-RAN introduces expanded **attack surfaces** and more likely for a bug to cascade into **system-wide disruption**
- Read the specs. Dive into the code. Contribute.

Final Thoughts

- O-RAN intr more likely for a uption
- Read the sp

O-RAN Software Community Releases “L”: Boosting Integration, AI/ML, and Open Source Collaboration
o-ran-sc.org/blog/2025/07/24/o-ran-software-community-releases-l-boosting-integration-ai-ml-and-open-source-collaboration/

Final Thoughts

- O-RAN introduces expanded **attack surfaces** and more likely for a bug to cascade into **system-wide disruption**
- Read the specs. Dive into the code. Contribute.
- Think broader: side channel, privacy leaks, flooding, …

Thank You!

Tianchang Yang
tzy5088@psu.edu
tianchang-yang.github.io

Kai Tu
kjt5562@psu.edu
hellotkk.github.io

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

Paper

Code