Detection of Denial of Service Attacks on the Open Radio Access Network Intelligent Controller through the E2 Interface

Vikas Krishnan Radhakrishnan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Luiz Pereira da Silva, Chair
Aloizio Pereira da Silva, Co-chair
Ivan Seskar

April 28, 2023
Arlington, Virginia

Keywords: O-RAN, xApp, RIC, E2 Interface, Security, DoS, Near-RT RIC

Detection of Denial of Service Attacks on the Open Radio Access Network Intelligent Controller through the E2 Interface

Vikas Krishnan Radhakrishnan

(ABSTRACT)

Open Radio Access Networks (Open RANs) enable flexible cellular network deployments by adopting open-source software and white-box hardware to build reference architectures customizable to innovative target use cases. The Open Radio Access Network (O-RAN) Alliance defines specifications introducing new Radio Access Network (RAN) Intelligent Controller (RIC) functions that leverage open interfaces between disaggregated RAN elements to provide precise RAN control and monitoring capabilities using applications called xApps and rApps. Multiple xApps targeting novel use cases have been developed by the O-RAN Software Community (OSC) and incubated on the Near-Real-Time RIC (Near-RT RIC) platform. However, the Near-RT RIC has, so far, been demonstrated to support only a single xApp capable of controlling the RAN elements. This work studies the scalability of the OSC Near-RT RIC to support simultaneous control signaling by multiple xApps targeting the RAN element. We particularly analyze its internal message routing mechanism and experimentally expose the design limitations of the OSC Near-RT RIC in supporting simultaneous xApp control. To this end, we extend an existing open-source RAN slicing xApp and prototype a slice-aware User Equipment (UE) admission control xApp implementing the RAN Control E2 Service Model (E2SM) to demonstrate a multi-xApp control signaling use case and assess the control routing capability of the Near-RT RIC through an end-to-end O-RAN experiment using the OSC Near-RT RIC platform and an open-source Software Defined Radio (SDR) stack. We also propose and implement a tag-based message

routing strategy for disambiguating multiple xApps to enable simultaneous xApp control. Our experimental results prove that our routing strategy ensures 100% delivery of control messages between multiple xApps and E2 Nodes while guaranteeing control scalability and xApp non-repudiation. Using the improved Near-RT RIC platform, we assess the security posture and resiliency of the OSC Near-RT RIC in the event of volumetric application layer Denial of Service (DoS) attacks exploiting the E2 interface and the E2 Application Protocol (E2AP). We design a DoS attack agent capable of orchestrating a signaling storm attack and a high-intensity resource exhaustion DoS attack on the Near-RT RIC platform components. Additionally, we develop a latency monitoring xApp solution to detect application layer signaling storm attacks. The experimental results indicate that signaling storm attacks targeting the E2 Terminator on the Near-RT RIC cause control loop violations over the E2 interface affecting service delivery and optimization for benign E2 Nodes. We also observe that a high-intensity E2 Setup DoS attack results in unbridled memory resource consumption leading to service interruption and application crash. Our results also show that the E2 interface at the Near-RT RIC is vulnerable to volumetric application layer DoS attacks, and robust monitoring, load-balancing, and DoS mitigation strategies must be incorporated to guarantee resiliency and high reliability of the Near-RT RIC.

Detection of Denial of Service Attacks on the Open Radio Access Network Intelligent Controller through the E2 Interface

Vikas Krishnan Radhakrishnan

(GENERAL AUDIENCE ABSTRACT)

Telecommunication networks need sophisticated controllers to support novel use cases and applications. Cellular base stations can be managed and optimized for better user experience through an intelligent radio controller called the Near-Real-Time Radio Access Network (RAN) Intelligent Controller (RIC) (Near-RT RIC), defined by the Open Radio Access Network (O-RAN) Alliance. This controller supports simultaneous connections to multiple base stations through the E2 interface and allows simple radio applications called xApps to control the behavior of those base stations. In this research work, we study the performance and behavior of the Near-RT RIC when a malicious or compromised base station tries to overwhelm the controller through a Denial of Service (DoS) attack. We develop a solution to determine the application layer communication delay between the controller and the base station to detect potential attacks trying to compromise the functionality and availability of the controller. To implement this solution, we also upgrade the controller to support multiple radio applications to interact and control one or more base stations simultaneously. Through the developed solution, we prove that the O-RAN Software Community (OSC) Near-RT RIC is highly vulnerable to DoS attacks from malicious base stations targeting the controller over the E2 interface.

Dedication

To dad and mom, and my brother Vishwa Krishnan.

Acknowledgments

I would first like to express my heartfelt thanks to Dr. Alexandre Huff, professor at the Federal University of Technology – Paraná, for his invaluable technical guidance in comprehending the open-source implementation of various O-RAN software pieces without which this work would not have seen the light of day. I am also extremely thankful for his constant support and motivation during our collaborative research work.

I extend my sincere thanks to my advisor Dr. Aloizio Pereira da Silva, and my committee chair, Dr. Luiz DaSilva, for their constant guidance and support with fine-tuning my research methodologies and articulating my findings with clarity and conviction. I thank my advisor for his kindness and understanding during tough times in my research journey. I also extend my gratitude to my external committee member, Professor Ivan Seskar, Chief Technologist & Director, WINLAB, for graciously accepting to review my work and for sharing his invaluable insights on improving this research further.

I thank Dr. Jacek Kibiłda and post-doctoral researchers Dr. João Santos and Dr. Mayukh Roy Chowdhury for their constructive feedback on my research. Thanks also to my research peers Oren Collaço, Prateek Sethi, Jaswanth Mallu, Tapan Bhatnagar, Aditya Sathish, Tarun Rao Keshabhoina, among other friends for being there for me when I needed help the most and for those memorable experiences, be it with coursework or during outings. Special thanks go out to my research colleague André Gomes for giving me timely advice and for patiently listening to my concerns and providing pragmatic suggestions during my research.

Finally, I thank my family and relatives, both near and far, from the past and in the present, for believing in me and always having my back at testing times. I would be remiss if I didn't acknowledge the efforts my brother, Vishwa Krishnan, put behind the scenes to get me enrolled in graduate school and see me defend this work successfully.

Last, but not the least, I gratefully acknowledge the infrastructure support I received from the Commonwealth Cyber Initiative (CCI) for conducting experiments using radio hardware and cloud computing resources. This work was supported by the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber R&D, innovation, and workforce development. Visit CCI at: www.cyberinitiative.org

Contents

List of Figures

List of Tables

Acronyms

4G fourth generation

5G fifth generation

AI Artificial Intelligence

API Application Programming Interface

ASN.1 Abstract Syntax Notation One

CCI Commonwealth Cyber Initiative

DBaaS Database as a Service

DMS Deployment Management Services

DoS Denial of Service

E2AP E2 Application Protocol

E2SM E2 Service Model

eNB evolved NodeB

EPC Evolved Packet Core

gNB Next Generation NodeB

gRPC Google Remote Procedure Call

IE Information Element

IMSI International Mobile Subscriber Identity

IP Internet Protocol

IPSec Internet Protocol (IP) Security

KPM Key Performance Measurement

LTE Long Term Evolution

MAC Medium Access Control

ML Machine Learning

Near-RT RIC Near-Real-Time RIC

NIB Network Information Base

Non-RT RIC Non-Real-Time RIC

NSA Non-Stand-Alone

O-RAN Open Radio Access Network

Open RAN Open Radio Access Network

OSC O-RAN Software Community

PF Proportional Fair

PHY Physical

PLMN Public Land Mobile Network

PRB Physical Resource Block

QoS Quality of Service

R-NIB Radio Network Information Base (NIB)

RAN Radio Access Network

RC RAN Control

RIC RAN Intelligent Controller

RMR RIC Message Router

SA Stand-Alone

SAC Slice-aware Admission Control

SCTP Stream Control Transmission Protocol

SDL Shared Data Layer

SDR Software Defined Radio

SINR Signal to Interference & Noise Ratio

SMO Service Management and Orchestration

SUCI SUbscriber Concealed Identity

UE User Equipment

UE-NIB UE Network Information Base (NIB)

USRP Universal Software Radio Peripheral

VPN Virtual Private Network

WG Working Group

Chapter 1

Introduction

1.1 Research Motivation

Open Radio Access Network (Open RAN) is an industry initiative for ensuring interoperability between disaggregated elements of the Radio Access Network (RAN) using open interfaces. The Open Radio Access Network (O-RAN) Alliance defines specifications to standardize these open interfaces and enable intelligent RAN control [1]. The O-RAN Software Community (OSC) develops open-source O-RAN software for the RAN Intelligent Controllers (RICs) following the reference archtitecture specified by the O-RAN Alliance [2]. The RIC platform allows custom applications to control the RAN. Given the increasing industry interest in rolling out O-RAN-compliant fifth generation (5G) mobile networks for civilian and military use cases, it is crucial to assess and guarantee the security posture and controllability of the RIC. It is also important to ensure that the RIC platform is scalable enough to support multiple applications targeting wide-ranging use cases.

1.2 Research Objectives

This thesis seeks to investigate the OSC Near-Real-Time RIC (Near-RT RIC) for compliance with the O-RAN specifications. It also aims to profile the scalability of the Near-RT RIC platform to support simultaneous RAN control applications. This thesis also proposes a

method to measure the control-loop latency of messages between the Near-RT RIC and the RAN. Finally, it evaluates the control loop signaling performance of the Near-RT RIC during a volumetric application layer Denial of Service (DoS) attack from a malicious RAN element. This thesis emphasizes real-time research experiments conducted on an end-to-end O-RAN deployment implemented on the Commonwealth Cyber Initiative (CCI) xG testbed [3], a Software Defined Radio (SDR)-based O-RAN-compliant experimental research testbed.

1.3 Research Contributions

We make the following contributions as part of this thesis:

- **Contribution 1:** We demonstrate, for the first time, the shortcomings in the design of the current reference implementation of the open-source Near-RT RIC from the OSC that restrict it from supporting simultaneous RAN control workflows through multiple xApps.

- **Contribution 2:** We semantically describe an Information Element (IE) named *RIC Requestor ID* defined in the O-RAN E2 Application Protocol (E2AP) specification [4] and extend it to propose a robust message-routing mechanism that enables simultaneous control messaging support for multiple xApps.

- **Contribution 3:** We enhance an existing open-source RAN slicing xApp, and prototype a new Slice-aware Admission Control (SAC) xApp to study an O-RAN use case with multiple xApps to enable control scalability for the Near-RT RIC platform.

- **Contribution 4:** We develop a Latency Monitoring xApp to monitor the control-loop signaling latency between the Near-RT RIC and the RAN.

- **Contribution 5:** We design a multi-threaded E2 Node simulator capable of orchestrating a resource exhaustion DoS attack and launching a signaling storm using RIC Indications over the E2 interface.

- **Contribution 6:** We assess the resiliency of the Near-RT RIC platform in the event of a volumetric application layer DoS attack via the E2 interface.

In addition to the above contributions, this thesis also reports on the following related extensions of research work:

- **Contribution 7:** Development of an Artificial Intelligence (AI)/Machine Learning (ML)-based end-to-end O-RAN workflow for policy-based RAN slicing that was demonstrated at the Mobile World Congress 2022.

- **Contribution 8:** Design of a RIC orchestration solution to optimize the placement of disaggregated RIC platform components for minimal control-loop execution latency.

1.4 Related Work

Research on enabling software-defined control of the RAN is reported in [5, 6, 7]. Within an O-RAN-compliant network architecture, most research work revolve around demonstrating RAN control using simulated RAN [8, 9, 10] without leveraging open-source software radio stacks to realize RAN control through the Near-RT RIC as demonstrated in [11]. Resiliency in O-RAN functions is studied in [12], however, it mainly targets fault tolerance for xApps deployed on the Near-RT RIC and not the Near-RT RIC platform itself. Furthermore, O-RAN specifications concerning the security of the Near-RT RIC have been mostly focused on mitigating threats from malicious xApps and compromised ML data pipelines from the

Non-Real-Time RIC (Non-RT RIC) or the Service Management and Orchestration (SMO) entity [13, 14, 15, 16, 17]. Also, security agencies stress the necessity to secure all O-RAN components and functions [18, 19]. However, E2 interface resiliency and security during interactions between multiple xApps and external E2 Nodes are still largely open questions.

1.5 Thesis Outline

This thesis is organized into nine chapters. Chapter 2 describes the Near-RT RIC architecture, and details the components and messaging sequences involved in the E2 interface. Chapter 3 illustrates an O-RAN use case requiring RAN control through multiple xApps and experimentally exposes the inability of the OSC Near-RT RIC to simultaneously support multiple xApps for RAN control. Chapter 4 analyzes the internal message routing mechanism for low-latency communications within the Near-RT RIC and expands upon end-to-end subscription and control message routing to ascertain the message routing limitation within the Near-RT RIC. Chapter 5 explains the implementation of the improved routing mechanism with experimental results validating multi-xApp RAN control scalability for the Near-RT RIC. Chapter 6 discusses the security aspects and the resiliency of the OSC Near-RT RIC platform and outlines the types of DoS attacks that can target the Near-RT RIC along with a potential latency monitoring solution leveraging the messaging routing mechanism demonstrated in Chapter 5. Chapter 7 elucidates the end-to-end experimental setup for orchestrating volumetric DoS attacks and explains each component involved in the attack. Chapter 8 reports the results concerning the performance of the Near-RT RIC and the E2 interface during normal operation, signaling storm attack conditions, and a high-intensity volumetric application layer DoS attack. Chapter 9 concludes the thesis with a summary of the research contributions and results, and outlines the scope for future work.

Chapter 2

Near-RT RIC and E2 Interface Communication

This chapter lays the foundation for the **contributions 1, 2, 3, 4, 5, and 6** and serves as a guide for the work realized in Chapters 3 through 8. In this chapter, we describe the platform components within the Near-RT RIC per the O-RAN specifications and the open-source reference software implementation from OSC. We also explore the lifecycle management of an xApp and understand the relevant procedures and messages involved in establishing a successful end-to-end connection between the xApp and the RAN. In O-RAN parlance, the RAN element is usually denoted as an E2 Node; in this document, these terms are interchangeable.

2.1 Near-RT RIC Platform

The Near-RT RIC is a conglomeration of multiple platform components providing key services to the infrastructure as a whole. Figure 2.1 shows the internal architecture of the OSC Near-RT RIC platform. The platform is managed as a Kubernetes cluster hosting several Near-RT RIC platform components deployed as microservices in a particular namespace within the cluster. These components coordinate with one another to facilitate service delivery through external interfaces (e.g., O1, A1, and E2) [2]. The E2 interface is a logical

network interface that allows southbound E2 Nodes to connect to the Near-RT RIC and expose externally-controllable *RAN functions* within the platform. These RAN functions can be utilized by one or more authorized xApps to modify specific RAN behavior [20]. The main Near-RT RIC platform components involved in enabling end-to-end network communication between an xApp and an E2 Node include:

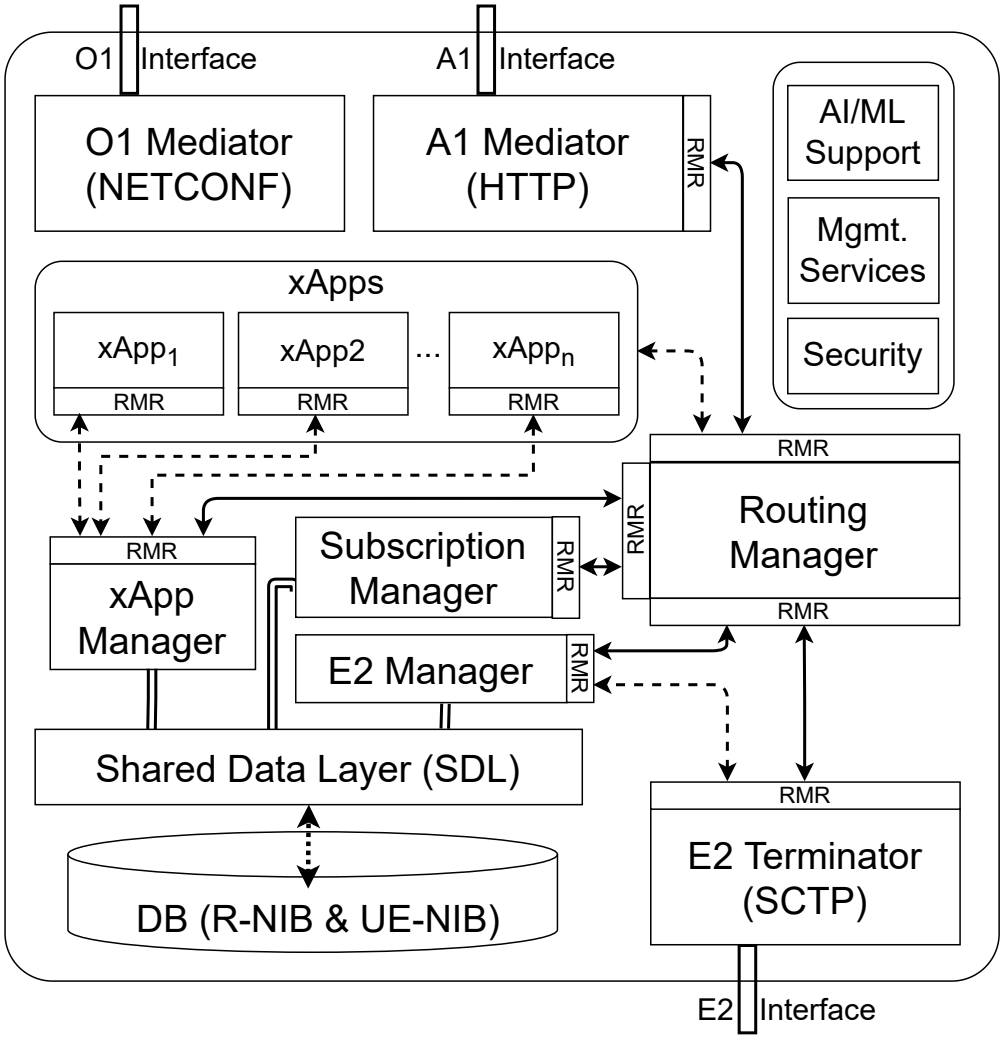


Figure 2.1: OSC Near-RT RIC architecture.

2.1.1 Routing Manager

The Routing Manager facilitates inter-platform messaging within the Near-RT RIC. It generates and distributes routing policies to all other platform components and any deployed xApps. These routing policies are used by the internal messaging infrastructure, called the RIC Message Router (RMR), to determine the correct messaging endpoint.

2.1.2 RMR

The RMR is not a monolithic platform component; instead, it is a messaging library incorporated by all platform components on the OSC Near-RT RIC, and functions as a message bus within the platform. This lightweight library is implemented atop Socket Interface-95 (SI95), which is reportedly more performant than the commonly-used Google Remote Procedure Call (gRPC) library [21]. The RMR sets up the sender and receiver message endpoints at predefined ports on the platform components and xApps that are deployed in the OSC Near-RT RIC. We explore the RMR and the Routing Manager in more detail in Chapter 4.

2.1.3 E2 Terminator

The E2 Terminator acts as the gateway for all E2 Nodes to access Near-RT RIC functions and services over the E2 interface. It is configured to listen for incoming E2 setup connections from E2 Nodes over a Stream Control Transmission Protocol (SCTP) connection. During E2 interface setup, the E2 Terminator records identifying information of the E2 Node in the platform database so that the Routing Manager can set up routing policies for RMR messages to reach this E2 Node. The E2 Terminator maintains E2 interface connectivity between the Near-RT RIC and one or more E2 Nodes and routes messages between them. It uses Abstract

Syntax Notation One (ASN.1) messaging for decoding and encoding E2AP messages from and to the E2 Nodes, respectively, over the E2 interface. For internal messaging within the Near-RT RIC, the RMR library is used to route ASN.1-decoded messages to the receiver endpoint based on the routes advertised by the Routing Manager.

2.1.4 E2 Manager

The E2 Manager functions as a means to a scalable architecture for managing multiple E2 Terminator instances within the same Near-RT RIC platform. It monitors the connection status of E2 Nodes bound to multiple E2 Terminator instances. Upon any state change, such as a successful connection or disconnection/loss of either the E2 Terminator instance or of an E2 Node to a managed E2 Terminator instance, it also triggers the Routing Manager to distribute updated routes, including that of the newly connected/disconnected E2 Node endpoint, to all platform components and deployed xApps.

2.1.5 Subscription Manager

The Subscription Manager enables authorized xApps to subscribe to RAN functions exposed by any connected E2 Nodes. It also merges identical subscriptions from multiple xApps targeting the same E2 Node, avoiding redundant data traffic on the E2 interface. A publish-subscribe model is, thus, provided for xApps to control and optimize the RAN. Upon receiving a valid subscription request from an xApp, the Subscription Manager generates a unique subscription ID and assigns it to the requesting xApp. Upon receiving a successful subscription response from the E2 Node, it notifies the xApp and triggers the Routing Manager to generate an updated routing table with the assigned subscription ID to enable further communications between the xApp and the E2 Node.

2.1.6 xApp Manager

The xApp Manager is in charge of the lifecycle management of xApps. The functions performed by this component map to the management Application Programming Interfaces (APIs) for the Near-RT RIC outlined in the Working Group (WG) 3 specifications [20]. These specifications envision the deployment and management of xApps to be controlled by the northbound SMO entity. The OSC Near-RT RIC currently requires xApp deployments to be triggered manually by first "onboarding" them first onto a Deployment Management Services (DMS) agent and then launching the onboarded xApp as a container-based microservice. Upon successful registration, the xApp Manager triggers the Routing Manager to update the routing table and inform all platform components of the induction of the registered xApp into the Near-RT RIC platform. Although the message flow for deregistering a deployed xApp has not yet been defined by O-RAN WG3 [20], the OSC Near-RT RIC software still implements this functionality in the xApp Manager as part of the manual lifecycle management process for xApps.

2.1.7 Database and Shared Data Layer (SDL)

The Near-RT RIC platform stores RAN data in two different databases, namely the Radio Network Information Base (NIB) (R-NIB) and the User Equipment (UE) Network Information Base (NIB) (UE-NIB). These databases persist RAN and UE data consumed by xApps to optimize service delivery. For example, the R-NIB could include details such as the E2 Node ID, its corresponding Public Land Mobile Network (PLMN) ID, and its connection status, while the UE-NIB could contain identifying information about the list of UEs, e.g., International Mobile Subscriber Identity (IMSI) for pre-5G networks, and SUbscriber Concealed Identity (SUCI) for 5G networks, and other tracking data associated with the

connected E2 Nodes. The databases are abstracted using the SDL API to enforce database integrity and monitorability, and to ensure high availability, scaling, and load balancing. This API can be accessed by all Near-RT RIC components and authorized xApps; however, in Figure 2.1, we only capture those API connections necessary for the end-to-end communication between an xApp and the E2 Node.

2.2 E2 Control-plane Signaling

As previously discussed, the E2 interface is a logical link between the Near-RT RIC and one or more E2 Nodes. Signaling over the E2 interface is achieved using ASN.1-encoded messages transported over a time-critical SCTP connection. Each E2 message consists of multiple IEs, and a sequence of E2 messages constitutes an E2 procedure [4]. The E2AP is an application layer signaling protocol that defines two types of procedures for operationalizing service delivery to the E2 Nodes.

2.2.1 Near-RT RIC Support Functions

These functional procedures allow the Near-RT RIC to support the establishment and management of an E2 interface connection atop the transport network layer. The E2 Terminator utilizes these functions to realize multiple E2 procedures such as E2 Setup, E2 Reset, E2 Removal, Near-RT RIC Service Update, and E2 Node Configuration Update. For example, the E2 Setup procedure, depicted in Figure 2.2, shows the messaging sequence between an E2 Node and the Near-RT RIC to set up a successful E2 connection.

The E2AP specifications [22] detail the mechanics of the other aforementioned Near-RT RIC support functions. For a functional E2 interface link between the E2 Node and the Near-RT RIC,

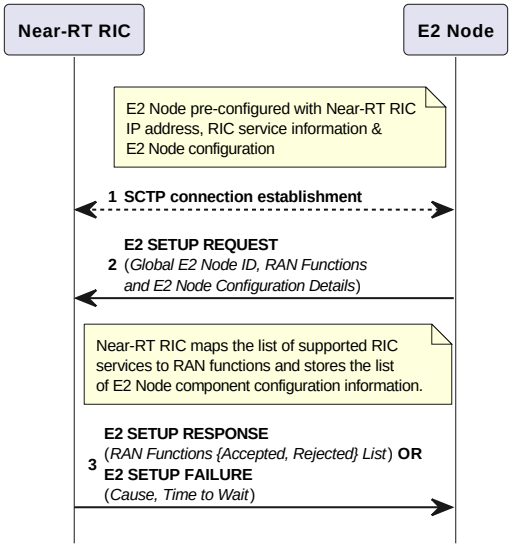


Figure 2.2: E2 Setup procedure between the E2 Node and Near-RT RIC.

the completion of the E2 Setup procedure is mandatory, while other support procedures ensure scalable and resilient operation of the interface over extended time periods.

2.2.2 Near-RT RIC Services

Near-RT RIC services include global procedures that allow for configurable service logic execution by an xApp on an E2 Node through the Near-RT RIC. The E2AP specifications define four *RIC Services* that xApps can provide to the E2 Node.

- REPORT Service: Allows the E2 Node to transfer information from the E2 Node as reports to the Near-RT RIC. This service involves asynchronous messaging in that no response is elicited from the Near-RT RIC.
- INSERT Service: Allows the E2 Node to temporarily suspend call processing and request control guidance from the Near-RT RIC. This service involves synchronous messaging, and a predefined subsequent action is automatically executed upon the

expiration of a *Time to Wait* timer.

- CONTROL Service: Allows the Near-RT RIC to initiate or resume an associated procedure in the E2 Node. This service involves synchronous messaging, provided the Near-RT RIC requests acknowledgment for the control request from the E2 Node.
- POLICY Service: Allows the E2 Node to automatically execute a specific policy upon the occurrence of the trigger event. This service involves asynchronous messaging.

For an xApp to consume a RIC Service, one or more E2AP procedures are required to be executed by the Near-RT RIC on the E2 Node. Figure 2.3 shows the sequence diagram detailing the fundamental E2AP procedures carried out by the E2 Terminator in the Near-RT RIC to install a RIC Service on the E2 Node [22]. The following E2AP procedures are defined by O-RAN WG3 [4]:

- *RIC Subscription*: Sets up a subscription between an xApp and an E2 Node;
- *RIC Subscription Delete*: Removes an existing subscription on the E2 Node;
- *RIC Subscription Delete Required*: Allows an E2 Node to request deletion of an existing subscription;
- *RIC Indication*: Transfers report data from the E2 Node to the Near-RT RIC based on the subscription;
- *RIC Control*: Initiates or resumes a specific functionality in the E2 Node.

A RIC Procedure requires a sequence of individual E2AP messages to be exchanged between the Near-RT RIC and the E2 Node. For example, to carry out the RIC Subscription Procedure, a RIC SUBSCRIPTION REQUEST message (sequence ① in Figure 2.3) is sent to the

E2 Node over the E2 interface. The message contains a contract-like payload that informs the E2 Node of the RIC service requested by the xApp in the Near-RT RIC. It carries IEs identifying the entity within the Near-RT RIC e.g., an xApp requesting a RIC Service and the corresponding RAN Function ID on the E2 Node that needs to be subscribed to. In response, the E2 Node attempts to provision its internal resources to meet the requirements in the subscription request and, if successful, responds with a RIC SUBSCRIPTION RESPONSE message. In case the subscription cannot be completed, a RIC SUBSCRIPTION FAILURE message (sequence ②) is sent back to the Near-RT RIC. Similar to the RIC Subscription Procedure, sequence ③ in Figure 2.3 implements the RIC Indication Procedure, sequences ④, ⑤, ⑥ implement the RIC Control Procedure, while ⑧, and ⑨ together implement the Subscription Delete Procedure.

Every RIC Service, except for the RIC CONTROL Service, always begins with a Subscription Procedure, which includes message sequences ① and ② in Figure 2.3. The REPORT Service involves a subscription to the E2 Node for a periodic or event-triggered condition, satisfying which, the E2 Node activates the RIC Indication Procedure with an Indication Type of REPORT. This Service involves message sequences ①, ②, and ③. The INSERT service differs from the REPORT service in that the E2 Node halts processing the associated call upon detecting the event trigger. It activates the RIC Indication Procedure with an Indication Type of INSERT and awaits further guidance from the Near-RT RIC. In such cases, the Near-RT RIC references the Call Processing ID IE from the RIC Indication Procedure and issues a RIC CONTROL REQUEST (④) to the E2 Node with subsequent actions. If the E2 Node is able to service the control request, and the Near-RT RIC has requested an acknowledgment for the same, it responds with a RIC CONTROL ACKNOWLEDGE message (⑤); otherwise, a RIC CONTROL FAILURE message (⑥) is sent back. The Near-RT RIC can also choose to activate the RIC Control Procedure asynchronously (⑥ and ⑦) based on

its internal event trigger conditions as well.

One or more RIC Services can be combined to create an E2 Service Model (E2SM) that can steer and optimize RAN behavior towards a target condition or state depending on the use case for the O-RAN deployment [23]. xApps leverage E2SMs to address RAN-specific use cases, as we shall explain in Chapter 4.

2.3 xApp Lifecycle Management

xApps are plug-and-play-style containerized microservices dedicated to the Near-RT RIC platform, and are deployed and managed by the SMO. The complete lifecycle of an xApp in the OSC Near-RT RIC is shown in Figure 2.4 and involves three distinct phases:

1. **Registration**: Once deployed within the Near-RT RIC, an xApp needs to first register itself with the xApp Manager. The registration process involves sharing a configuration file with data such as the xApp's name, version, the types of messages it intends to send and receive within the Near-RT RIC platform, among other details. The xApp Manager validates the registration request and triggers a routing table update to the Routing Manager to enable the xApp to communicate with the rest of the Near-RT RIC platform components.

2. **Operation and Reconfiguration**: After successful registration, the xApp performs service discovery, a process by which different Near-RT RIC services the xApp is authorized to interact with are discovered. Next, the xApp establishes communications using the RMR messaging infrastructure. Subsequently, it queries the SDL to check for connected E2 Nodes. Upon finding a matching target, the xApp sends a subscription request containing a RIC Service installation request to the Subscription

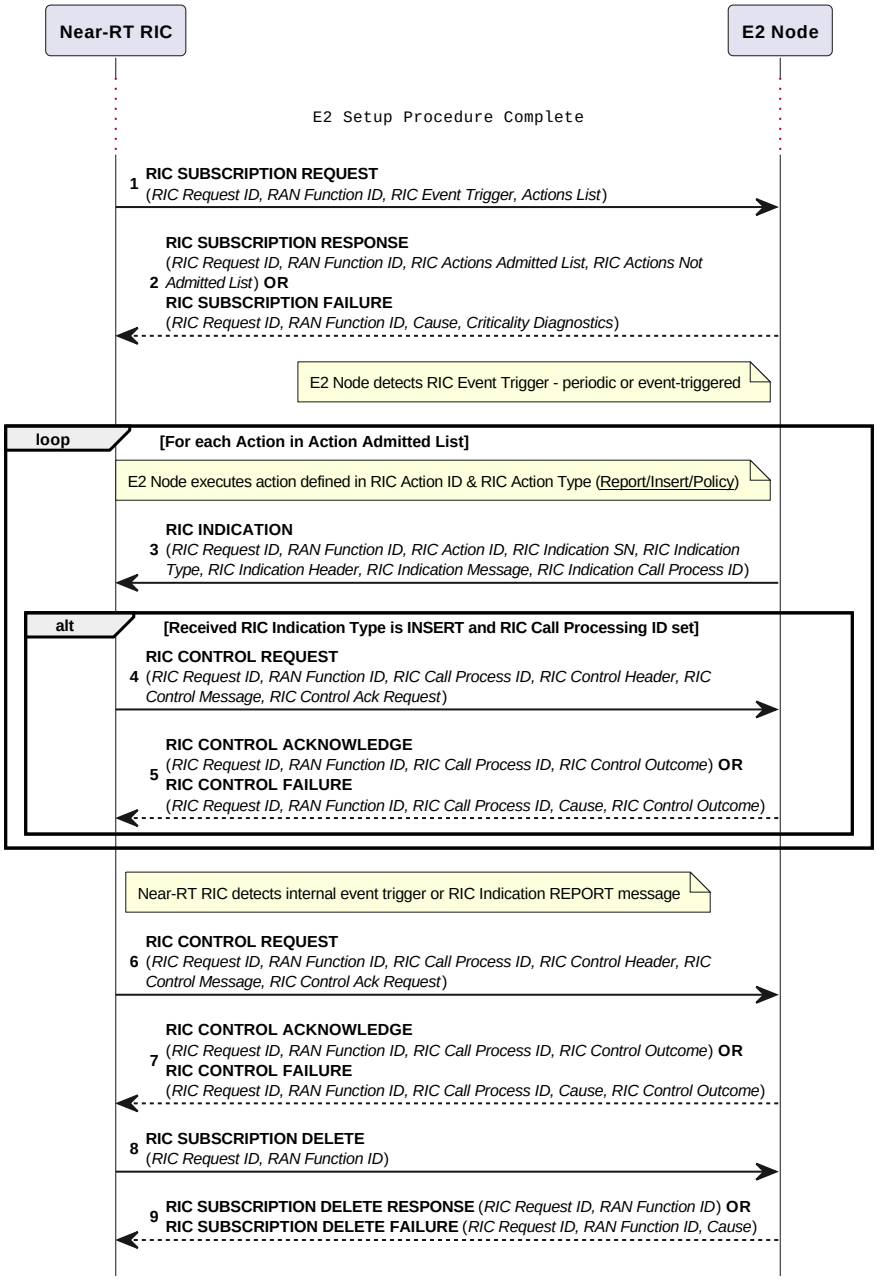


Figure 2.3: RIC Services, Procedures and their component E2AP messages.

Manager. The Subscription Manager validates the subscription request and assigns a new subscription ID to the subscription request. It then forwards the request to the E2 Terminator, which, in turn, forwards it to the E2 Node over the E2 interface. Upon receiving a subscription response in the same return path, the subscription manager requests the Routing Manager to generate updated routing policies by mapping the subscription ID to the requesting xApp. The xApp can then interact directly with the E2 Node through the E2 Terminator using the unique subscription ID.

3. **Deregistration**: The OSC Near-RT RIC allows the xApp Manager to deregister an xApp manually and clean up resources after gracefully terminating the microservice. When the xApp Manager triggers deregistration for a running xApp, the xApp removes all subscriptions it has with one or more E2 Nodes by triggering the Subscription Delete Procedure. The xApp can then terminate gracefully, following which, the xApp Manager requests the Routing Manager to update the routing table to remove any routes pointing to the terminated xApp.

The subsequent chapters build upon concepts discussed in this chapter to realize use cases that involve the execution of multiple xApps for simultaneous RAN control and for characterizing the security aspects of the Near-RT RIC towards the E2 interface.

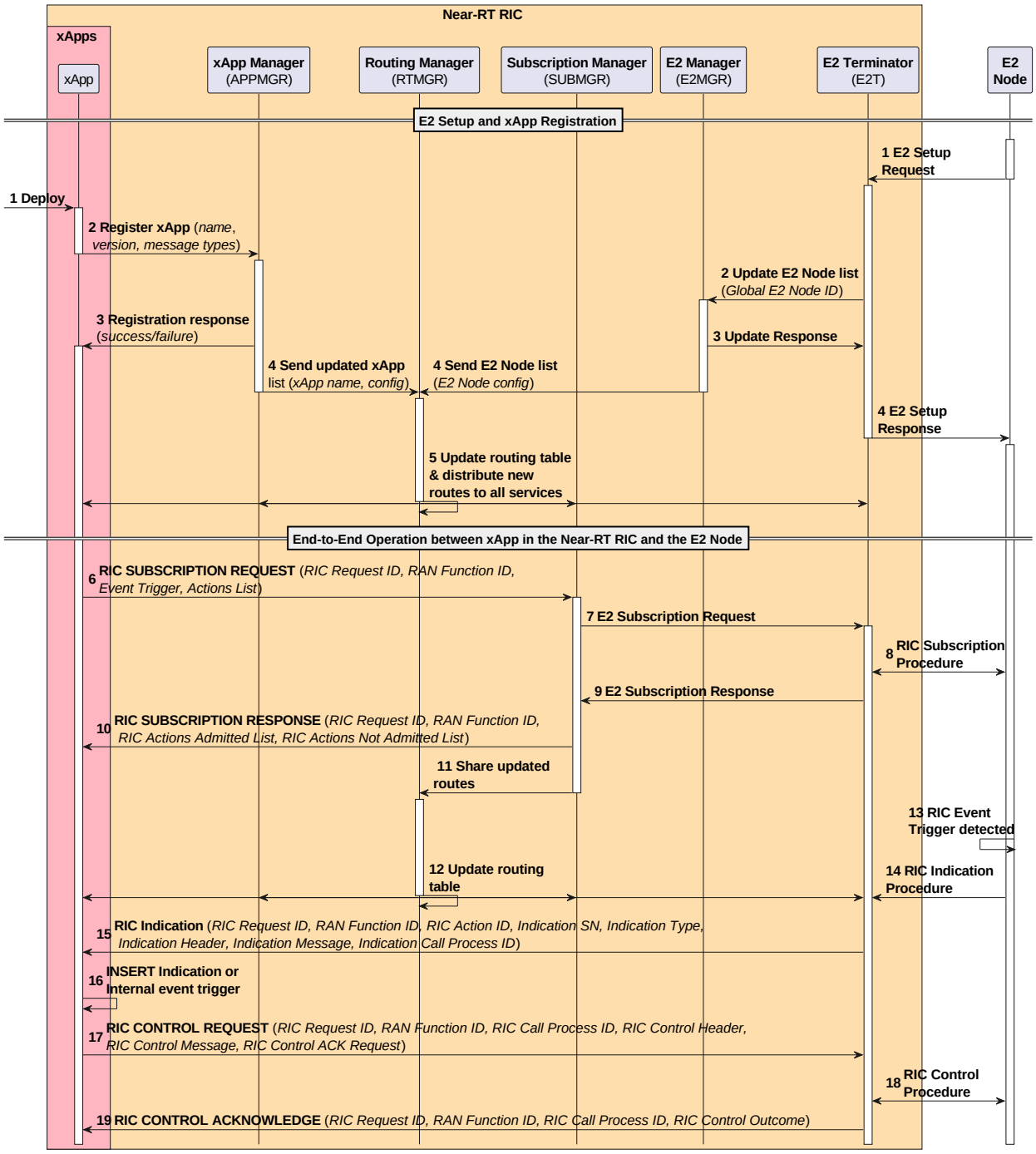


Figure 2.4: E2 Node setup and xApp lifecycle management.

Chapter 3

Multi-xApp RAN control

In this chapter, we leverage the concepts discussed in the previous chapter to explain an O-RAN use case concerning slice-aware UE admission control that requires simultaneous control of the target E2 Node by multiple xApps deployed on the same Near-RT RIC platform (**Contribution 3**). Furthermore, we describe our experimental setup on the CCI xG Testbed for implementing and validating the aforementioned use case on an end-to-end O-RAN-compliant open-source software radio stack. As part of the results, we demonstrate the inherent limitations in the routing mechanism of the existing OSC Near-RT RIC for routing E2 control messages among multiple xApps (**Contribution 1**).

3.1 E2SMs for xApp-driven RAN Control

In Section 2.2, we highlighted the aspect of combining RIC Services into an E2SM to enable complex signaling workflows between the Near-RT RIC and the RAN. The O-RAN specifications define several E2SMs targeting specific behaviors of the E2 Nodes. For example, the Key Performance Measurement (KPM) E2SM installs a simple REPORT RIC Service that instructs the E2 Node to send the key performance metrics in a REPORT-type RIC INDICATION message. The event trigger can be periodic or based on a condition defined by the xApp, e.g., when a serving E2 Node receives a UE measurement report with Signal to Interference & Noise Ratio (SINR) value lower than a set threshold. Likewise, the RAN

Control (RC) E2SM installs multiple RIC Services, including INSERT and CONTROL services that enable the E2 Node to seek near-real-time guidance from the Near-RT RIC to manage transient conditions such as mobility handover and UE admission control.

3.2 Slice-aware UE Admission Control Use Case

RAN slicing has been well-researched in fourth generation (4G) as well as 5G mobile networks [24]. While open-source solutions already exist for realizing slicing control on the RAN scheduler [11], no Open RAN-based open-source solutions exist to perform dynamic slice profile management. A typical requirement for enabling RAN slicing is a slice resource optimizer that can dynamically allocate each UE to a slice based on a slice profile that caters to the Quality of Service (QoS) requirements of the end-users. We consider such a use case and design an O-RAN workflow that extends an open-source closed-loop subframe slicing xApp called NexRAN [11] by adding the ability to dynamically add UEs to or remove them from an existing slice managed by the xApp on the RAN.

Figure 3.1 shows the system architecture for realizing the slice-aware UE admission control use case using the OSC Near-RT RIC. The setup consists of 4 components:

- Evolved Packet Core (EPC): Hosts the 4G mobile core. We use srsEPC in our implementation [25].
- evolved NodeB (eNB): O-RAN-compatible monolithic RAN software radio stack. It connects to the EPC over the backhaul S1 interface and to the Near-RT RIC over the E2 interface. A Universal Software Radio Peripheral (USRP) X310 SDR acts as the radio front-end for over-the-air radio transmission.
- Near-RT RIC: Provides infrastructure support for xApps to connect to and control

the eNB.

- UE: The end-user device that connects to the 4G network. We use the srsUE application on a compute node connected to a USRP X310 for over-the-air signal transmission and reception.

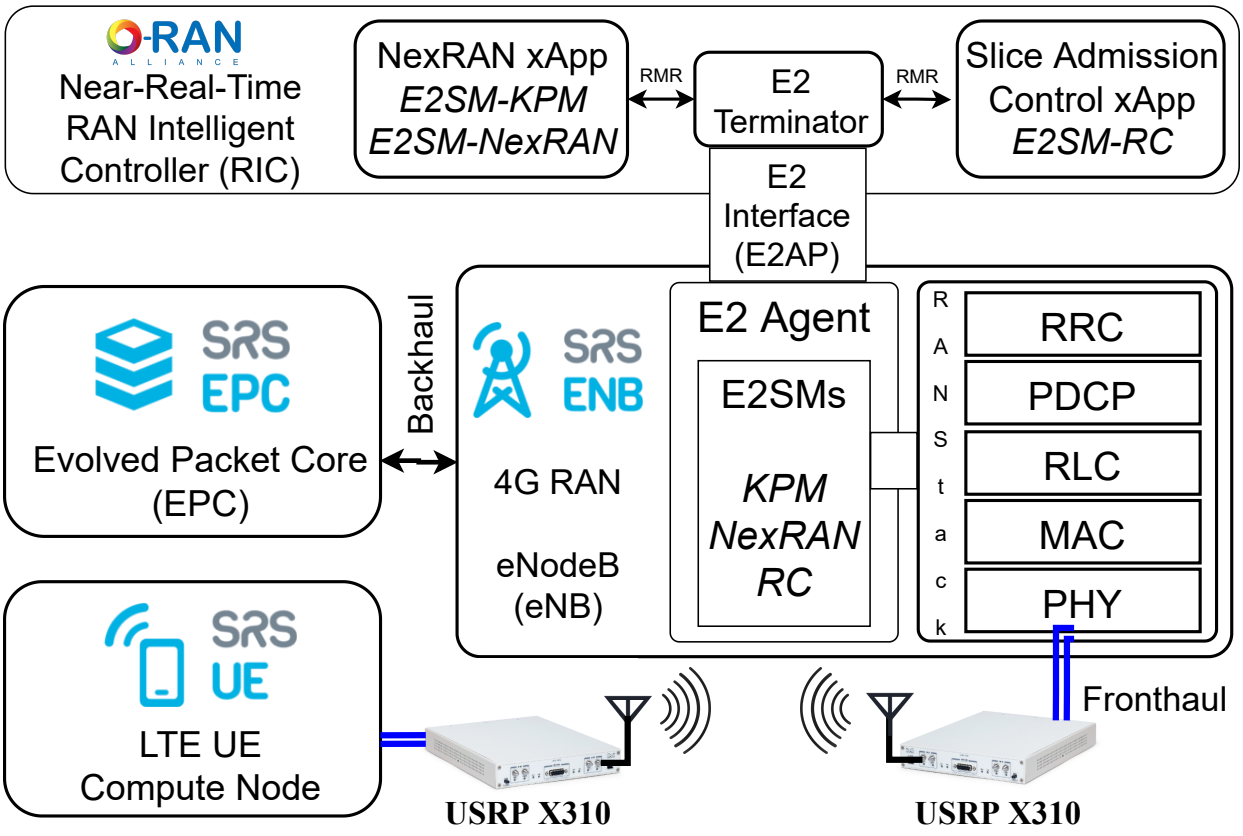


Figure 3.1: System architecture of the experimental setup.

The NexRAN xApp employs a custom E2SM that utilizes the REPORT RIC Service to periodically monitor the KPM metrics of the RAN, and the CONTROL RIC Service to dynamically modify the slice share of the Proportional Fair (PF) subframe scheduler in the Medium Access Control (MAC) layer of the Long Term Evolution (LTE) radio stack. This xApp requires the operator to manually bind and unbind UEs to a slice on the RAN. We prototype the SAC xApp that automates the admission of an incoming UE into an existing

slice based on RAN resource availability and the UE's QoS requirements. The prototype implements the RC E2SM by subscribing to the INSERT RIC Service on the RAN. The subscription is configured such that the E2 Node sends a RIC INDICATION message of type INSERT whenever a new UE needs to be admitted into the network. The SAC xApp analyzes the RAN resource utilization status from the NexRAN xApp and accordingly sends a RIC CONTROL REQUEST informing the RAN about whether the new UE should be admitted or rejected. It is pertinent to note that the RIC Control Procedures utilized by both the xApps are completely non-conflicting; that is, the RIC CONTROL REQUEST messages sent by each xApp affect the RAN mutually exclusively. The NexRAN xApp adjusts the slice share by modifying the number of Physical Resource Blocks (PRBs) allocated to each slice, while the Slice Admission Control xApp accepts or rejects a UE into a RAN slice.

3.3 Experimental Setup

As part of our experimental validation, we upgrade the NexRAN xApp with the current version of the E2AP (E2AP v02.03) to work with the G-Release of the OSC Near-RT RIC. Additionally, we prototype the E2AP signaling logic and the RC E2SM for the SAC xApp to facilitate slice-aware UE admission control on slices managed by the NexRAN xApp (**Contribution 3**). The two xApps are deployed on the Near-RT RIC platform simultaneously and configured to target a single E2 Node to perform non-conflicting RAN control actions: change of slice shares by the NexRAN xApp, and UE admission control by the SAC xApp. We use a modified version of the open-source srsRAN 4G software radio stack for deploying the eNB, the 4G core, and the UE application. [11].

Figure 3.2 shows the number of RIC CONTROL messages exchanged between the two xApps and the E2 Node over the E2 interface. At ①, the NexRAN xApp initiates a REPORT

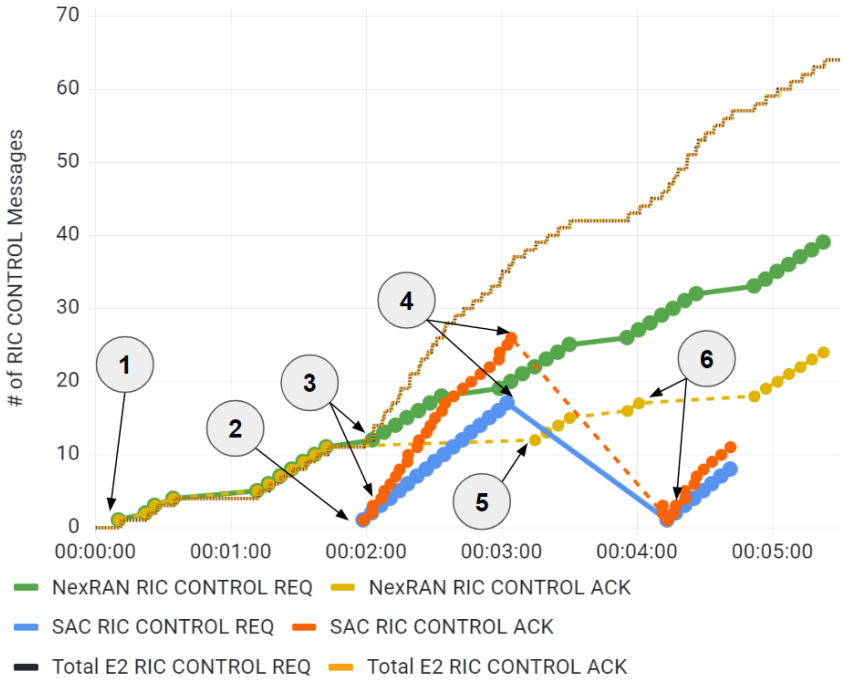


Figure 3.2: Control message routing for simultaneous xApp control in the current OSC Near-RT RIC.

subscription to the E2 Node and triggers the RIC CONTROL Service based on received RIC INDICATION reports. At ②, the SAC xApp sends an INSERT subscription to the E2 Node and triggers a separate RIC CONTROL Service whenever the RIC INDICATION arrives. Soon after the SAC xApp begins execution, the NexRAN xApp stops receiving its control response messages (③). Interestingly, those control response messages get routed to the SAC xApp, as witnessed by the steady increase in its control response messages as compared to the control requests sent. This status continues until the SAC xApp is terminated (④), after which control response messages to the NexRAN xApp get delivered as expected (⑤). This control message routing behavior indicates that the RIC Control Procedure of the first xApp is interrupted whenever both xApps execute the CONTROL RIC Service as observed again in ⑥. The following chapter analyzes the root cause behind this message routing anomaly.

Chapter 4

Message Routing in the Near-RT RIC

This chapter analyzes how the OSC Near-RT RIC accomplishes internal message routing. It also discusses the limitations of the current design of the Routing Manager and the RMR in disambiguating between routing endpoints for RIC CONTROL REQUEST messages and their security ramifications. Finally, we propose an improved message routing mechanism to remedy the routing inconsistencies discussed in the previous section and to enable multiple xApps to control the E2 Node simultaneously (**Contribution 2**).

4.1 Routing Table and Policies

In chapters 2, and 3, we explain how communication over the E2 interface between the E2 Terminator in the Near-RT RIC and the E2 Node is accomplished using the E2AP. After the E2 messages reach the Near-RT RIC platform, the RMR messaging library and the Routing Manager together enable them to be routed amongst all platform components, including the xApps. The Routing Manager generates new routing policies and distributes the routing table to all connected Near-RT RIC services whenever a new entity is detected or lost by the Near-RT RIC platform. These transient entities could be xApps, E2 Terminator instances, or E2 Nodes. The routing table contains route entries with fields separated by a vertical bar (|) and satisfying the following syntax:

```
mse | <msg-type>[,<sender-endpoint>] | <sub-id> |
      <endpoint-group>[;<endpoint-group>;...]
```

Each route entry starts with the label `mse` and contains 4 fields (indicated within angled brackets <>) in the syntax above. The sub-fields denoted within square brackets [] are optional. The RMR routes messages using two key fields: `msg-type` and `sub-id`, to determine the correct recipient of an RMR message. A route entry is considered valid if no sender endpoint is specified, or if the sender endpoint matches correctly with an application's hostname and RMR listening port. Message types are mapped to constant message IDs for identifying the correct receiver endpoint or endpoint groups by the RMR using the routing policies generated by the Routing Manager. Table 4.1 lists the common message types and their corresponding message IDs used by the RMR.

Message ID	Message Type
12010	RIC SUBSCRIPTION REQUEST
12011	RIC SUBSCRIPTION RESPONSE
12012	RIC SUBSCRIPTION FAILURE
12020	RIC SUBSCRIPTION DELETE REQUEST
12021	RIC SUBSCRIPTION DELETE RESPONSE
12022	RIC SUBSCRIPTION DELETE FAILURE
12040	RIC CONTROL REQUEST
12041	RIC CONTROL ACKNOWLEDGE
12042	RIC CONTROL FAILURE
12050	RIC INDICATION

Table 4.1: Common RMR message types used for routing.

For messages of the same message type to be routed to different receiver endpoints, the RMR uses the `sub-id` as a secondary key for determining the correct destination for the message. This key can be considered logically optional since the RMR always routes certain message types to a fixed endpoint. For example, all subscription requests from xApps are always routed to the Subscription Manager. The `sub-id` in such cases is set to -1 to indicate that routing should be based solely on the message type. Moreover, for a given message

type, if there exist multiple routes with identical `sub-id`s, then the RMR uses the *last* valid route entry in the routing table. As we shall see in Section 4.3, this design severely hampers routing when the sender does not have a live subscription to the E2 Node.

4.2 Subscription and Control Messaging Flows

Messaging workflows initiated by xApps can be broadly classified into two types: 1) Subscription messaging and 2) Internally-triggered control messaging, as described next.

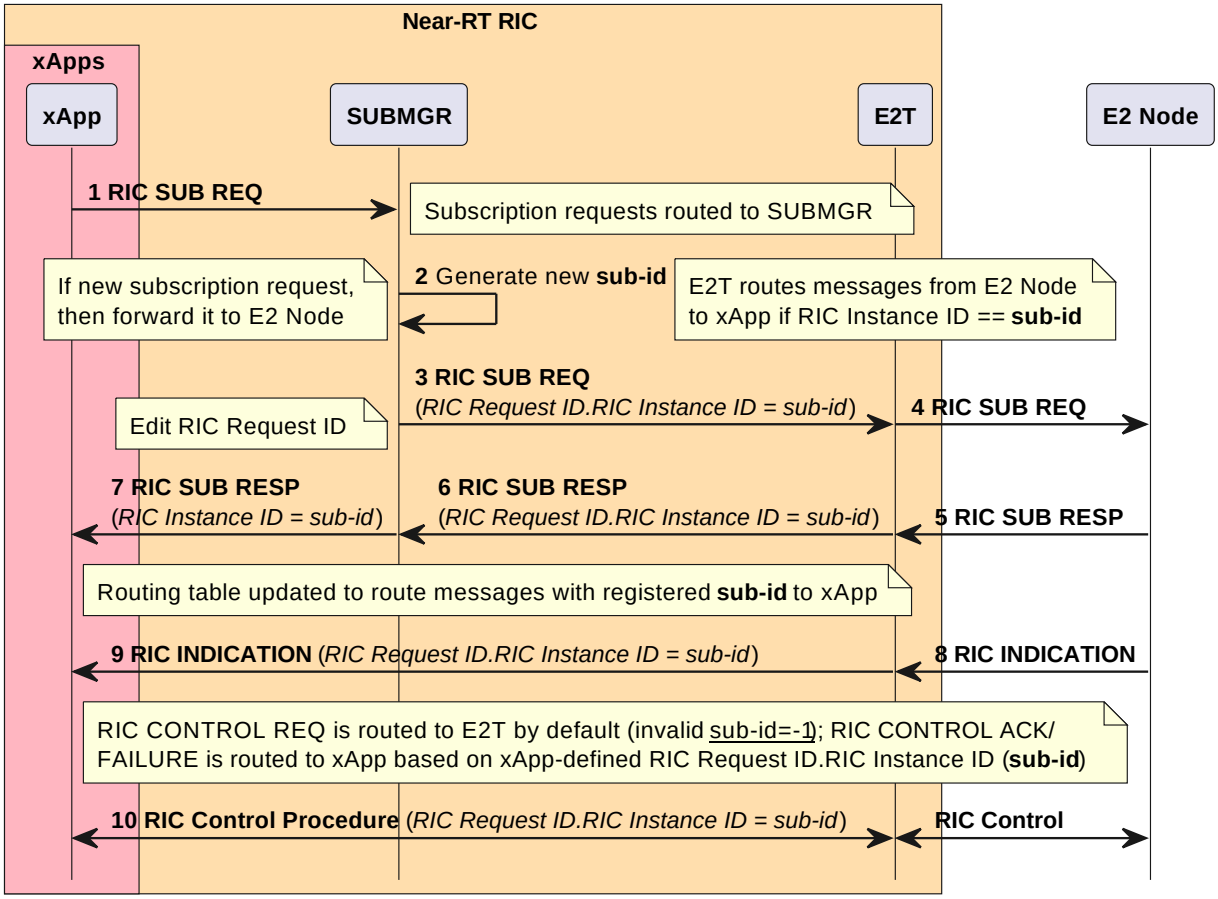


Figure 4.1: Subscription messaging workflow.

4.2.1 Subscription Messaging

xApps usually require data from the E2 Node before they can control and optimize RAN behaviors. This data can either be generated through a subscription to the E2 Node or retrieved from the Near-RT RIC platform database (R-NIB or UE-NIB). Given that xApps need to operate in the near-real-time timescale of $10ms - 1s$, they normally aggregate data through new subscriptions to the E2 Node. The subscription workflow is shown in Figure 4.1. When an xApp sends a fresh subscription request over RMR, it does not have a `sub-id` yet. Since all subscription-related messaging is facilitated by the Subscription Manager, the `sub-id` key is set to -1, in which case, the RMR routes the subscription request using only the `msg-type` in consultation with the routing table. The route entry for this routing policy could be:

```
mse | 12010, <xapp-rmr-endpoint-name> | -1 |
      <submgr-service-rmr-endpoint-name:rmr-port>
```

Similarly, the route entries to route subscription messages between the Subscription Manager and the connected E2 Node(s) also exist in the routing table. Upon reception of the subscription request message from the xApp, the Subscription Manager creates a unique `sub-id` and assigns this value to the *RIC Instance ID* field within the *RIC Request ID* IE of the subscription request message and forwards it to the E2 Terminator which in turn sends the message to the E2 Node. Considering a positive outcome, wherein the E2 Node successfully registers the request, it sends a subscription response to the E2 Terminator, which forwards the message to the Subscription Manager. Sensing the successful subscription, the Subscription Manager informs the Routing Manager to update routes exposing the xApp's RMR endpoints directly to the E2 Terminator for subsequent messaging. The Subscription Manager also forwards the subscription response to the xApp with the *RIC Instance ID* set to the `sub-id`. The E2 Terminator decodes the incoming message payload from the E2 Node

to figure out the *RIC Instance ID* value to set the `sub-id` for the RMR message to be sent to the receiver endpoint. This enables the xApp to receive RIC INDICATION messages for this subscription from the E2 Node and to reuse the same `sub-id` for further control messaging with the same E2 Node. The control messaging thus established is assumed to be synchronous, wherein the xApp is "triggered" to control the E2 Node based upon the reception of a RIC INDICATION message of type INSERT.

4.2.2 Internally-triggered Control Messaging

As shown in Figure 4.2, this type of control messaging happens when an xApp asynchronously initiates the RIC Control Procedure on a target E2 Node without an explicit INSERT-type RIC INDICATION message from the E2 Node. This messaging flow is necessary in two prominent cases: 1) the xApp wants to control RAN behavior based on REPORT-style RIC INDICATION messages from a merged subscription shared by multiple xApps. 2) the use case for the xApp does not require an explicit subscription to any RIC Service, but wants to initiate the RIC Control Procedure based on an internal event trigger.

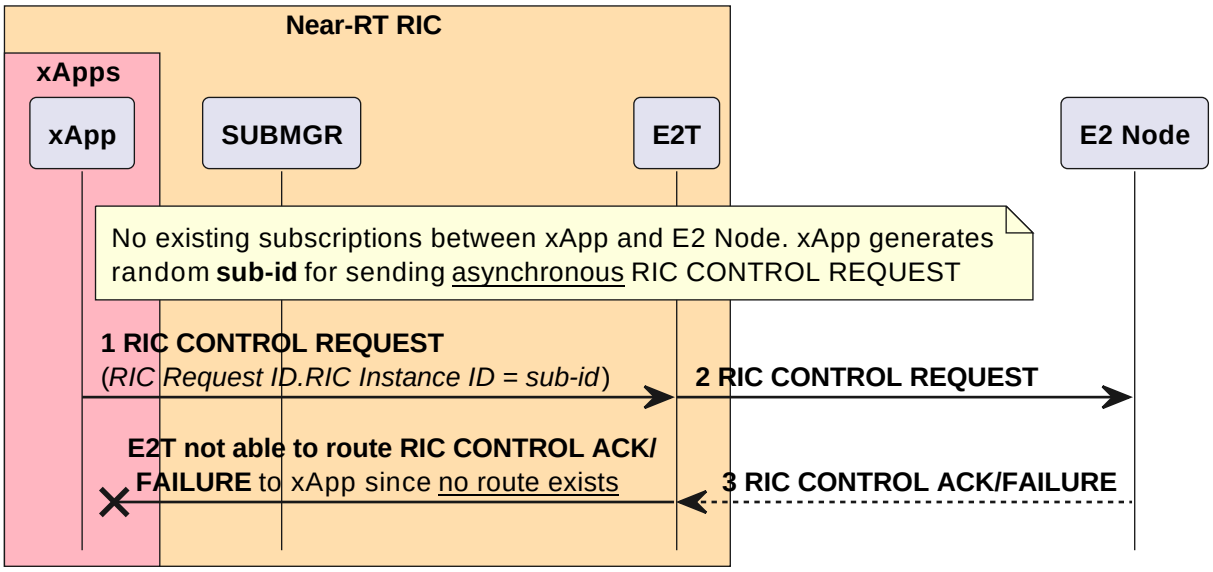


Figure 4.2: Internally-triggered control messaging workflow.

Allowing xApps to control RAN behavior without first subscribing to a RIC Service could be seen as a potential security vulnerability as malicious and compromised xApps can make spurious RAN control modifications that could have disastrous consequences for both the E2 Node and the Near-RT RIC. Also, since the Subscription Manager can merge subscriptions with the same event triggers and actions, allowing a subscribed xApp to reuse the same `sub-id` for control messaging that requires a control response from the E2 Node could be counterintuitive. This is because the RMR routes messages from the E2 Node to any and all xApps with that `sub-id`. For example, xApps listening to RIC Indication messages through a merged subscription, but not registered to use the RIC CONTROL Service, might end up receiving control response messages for control requests issued by another merged subscriber xApp if message routing is solely based on a merged `sub-id`.

In cases where the xApp's use case requires leveraging the RIC CONTROL Service without first having a subscription, the xApp has to self-generate an RMR `sub-id` and set it to the *RIC Instance ID* field within the *RIC Request ID* IE in the control request. When the xApp sends the control request, the RMR forwards the message to the E2 Terminator which is the default receiver endpoint for all RIC CONTROL REQUEST RMR messages (`sub-id` $= -1$). However, when the control response arrives from the E2 Node, the E2 Terminator is not able to route the control response RMR message back to the concerned xApp since it cannot find a valid `sub-id` in the routing table corresponding to the *RIC Instance ID* value that the xApp generated.

4.3 OSC Near-RT RIC Design Limitations

The OSC Near-RT RIC currently provides complete support for subscription-based messaging. However, internally-triggered control messaging is limited to, at most, one xApp

because the Routing Manager is configured, by default, to generate routes with an invalid `sub-id` when the xApp registers itself with the xApp manager. This results in the following route table entries upon successful xApp registration:

```
mse | 12040, <xapp-rmr-endpoint-name:rmr-port> | -1 |
       <e2term-rmr-endpoint-name:rmr-port>
mse | 12041 | -1 | <xapp-rmr-endpoint-name:rmr-port>
mse | 12042 | -1 | <xapp-rmr-endpoint-name:rmr-port>
```

Here, a `sub-id` value of -1 indicates that control messages shall be routed based only on the message type. While all control requests originating from different xApps get routed seamlessly to the E2 Node through the E2 Terminator, response messages from the E2 Node are routed back to the sending xApp if and only if a single xApp is registered to send and receive control-related messages. Since the RMR considers the last valid entry in the routing table as the receiver endpoint, control response messages are always routed to the last xApp that was registered with a configuration defined to utilize the CONTROL RIC Service. This explains the anomalous message routing behavior observed in Figure 3.2. A malicious xApp could easily exploit this vulnerability in routing control responses to spoof the E2 Terminator and flood an xApp peer with unexpected control responses leading to a DoS attack on the xApp service, the Near-RT RIC, or connected E2 Nodes. This condition points to the limitation in the routing policy used by the Routing Manager. It also exposes a fundamental design flaw in the RMR library that prevents it from disambiguating between multiple xApps during control message routing from the E2 Node into the Near-RT RIC. The following chapter presents a comprehensive messaging strategy to prevent routing conflicts for control messages.

Chapter 5

Improved Routing Mechanism for xApp Control Scalability

In this chapter, we address the control message routing anomaly demonstrated in the previous chapter by implementing a foolproof message-routing mechanism that allows the Near-RT RIC to support simultaneous control messaging by multiple xApps while guaranteeing control scalability and xApp non-repudiation on the Near-RT RIC platform (**Contribution 2**). This is achieved through xApp tagging — to differentiate multiple xApps or multiple instances of the same xApp, and tag-aware message routing in the RMR — to ensure unambiguous route selection.

5.1 xApp Tagging

We propose that an xApp requesting registration in the Near-RT RIC platform should be assigned a platform-wide unique `app-id` by the xApp manager. The xApp should use this tag value for subsequent messaging within the Near-RT RIC. This scheme streamlines the xApp registration process in a way similar to how the Subscription Manager generates and manages unique and non-conflicting `sub-id`s. Upon successfully registering the xApp, the xApp manager requests the Routing Manager to update its routing policies by including the `app-id` tag value. The updated route table entry looks as shown below:

```
mse | <msg-type>[,<sender-endpoint>] | <app-id> | <sub-id> |
      <endpoint-group>[;<endpoint-group>;...]
```

where the `app-id` indicates the locally unique tag assigned to the xApp by the xApp Manager. This provides a way to disambiguate between multiple xApps within the Near-RT RIC platform irrespective of whether they have a valid subscription to an E2 Node. xApp tagging also enables the Near-RT RIC to manage multiple instances of the same xApp when they need to be scaled for resiliency and load balancing.

5.2 Robust Routing Design for RMR and E2 Terminator

In order to support message routing based on xApp tag values, we update the RMR to route messages based on the `app-id` in addition to `msg-type` and `sub-id`. This enables all Near-RT RIC platform services that use the RMR library to additionally consider the `app-id` to determine the correct receiver endpoint within the Near-RT RIC. The xApp needs to compulsorily set its tag value in the `app-id` of the RMR message, and optionally update the `sub-id` received from the Subscription Manager (otherwise set to -1 by default), before sending the RMR message to other receiver endpoints.

As for the RIC CONTROL REQUEST message to be sent over the E2 interface, the xApp needs to additionally set the `app-id` in the payload similar to how the `sub-id` is assigned to the *RIC Instance ID*. The E2AP specifications [20, 22] highlight the need to use the *RIC Request ID* IE for unique identification of messages exchanged between the xApp and the E2 Node. Notably, the IE contains two IEs, namely the *RIC Requestor ID* and the *RIC Instance ID*. The E2AP specifications [4] do not explicitly specify how these IEs should be

used to mark messages that are exchanged over the E2 interface. In the OSC Near-RT RIC, only *RIC Instance ID* is currently used, i.e., to tag the message with `sub-id`. We propose to use the IE *RIC Requestor ID* to carry the `app-id` that uniquely identifies different xApps or even different instances of the same xApp, since the xApp Manager allocates unique tags to each registered xApp. Figure 5.1 shows the updated workflow to ensure guaranteed control request and response delivery between the xApps and E2 Nodes.

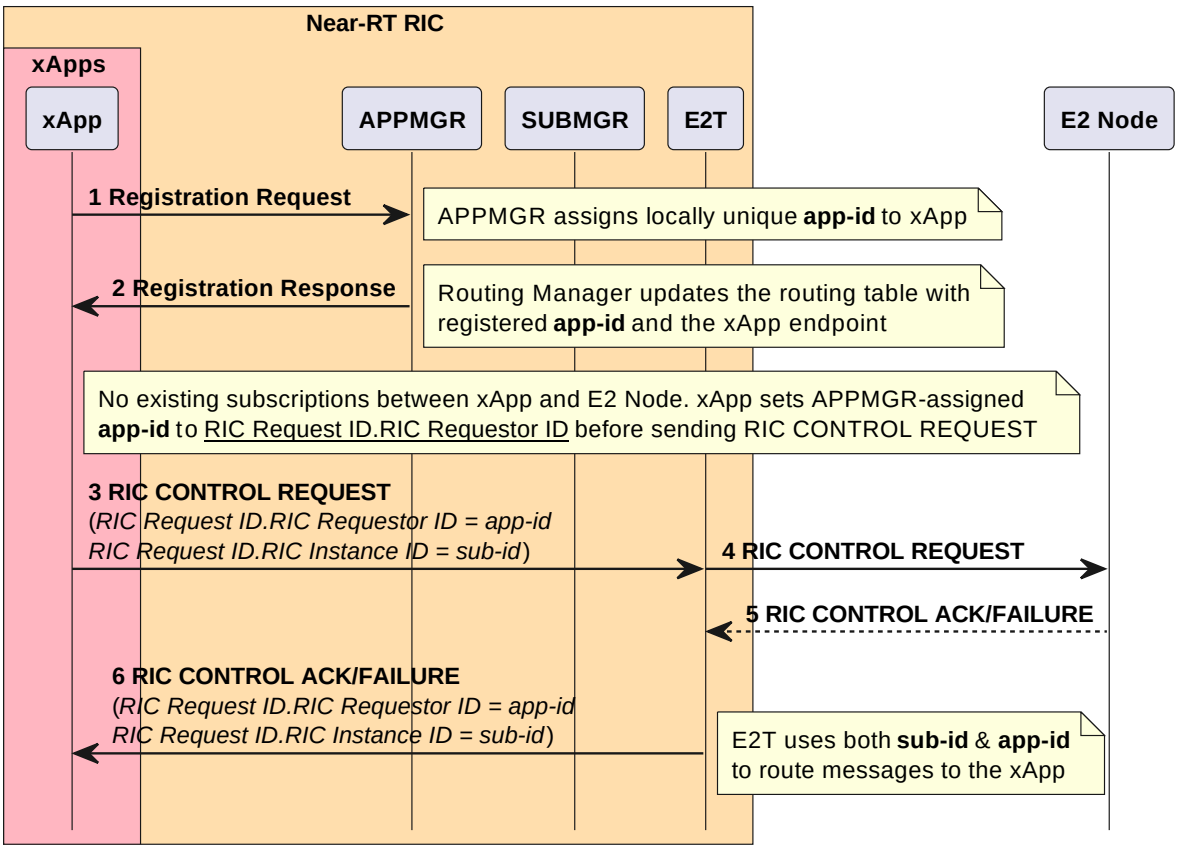


Figure 5.1: xApp-triggered control messaging workflow with the proposed routing mechanism.

When the response message arrives from the E2 Node, the E2 Terminator decodes the payload and sets the *RIC Requestor ID* and the *RIC Instance ID* values to the `app-id` and `sub-id` fields of the RMR message respectively. With the updated routing table, the RMR message unambiguously determines the correct receiver endpoint and delivers the message

to the correct xApp.

5.3 Evaluation of the Improved Routing Mechanism

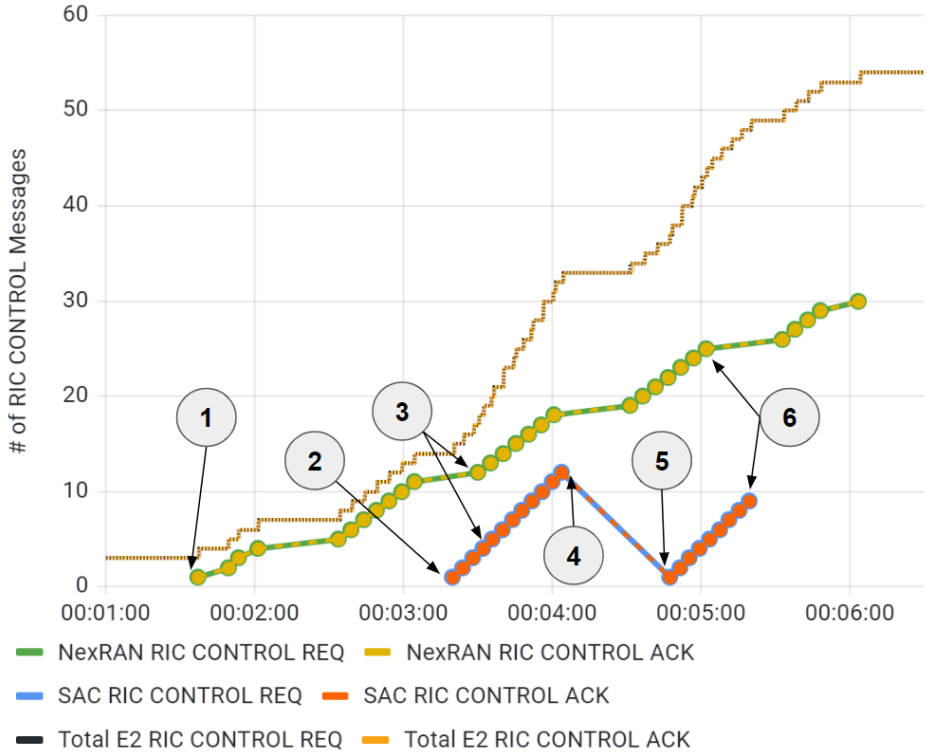


Figure 5.2: Control message routing for simultaneous xApp control with the proposed routing mechanism.

We validate the performance of the proposed message routing mechanism by rerunning the same experiment for the slice-aware UE admission control use case discussed in Chapter 3. Figure 5.2 shows the number of control messages exchanged between the NexRAN and SAC xApps towards the target E2 Node, and the total number of control messages processed by the E2 Terminator. Unlike the routing behavior observed in Figure 3.2, even after the SAC xApp subscribes to the E2 Node and initiates RIC CONTROL service (②), both xApps continue to receive control response messages corresponding to their control requests (③).

We notice consistent message routing behavior between xApp restarts, as indicated in ④ and ⑤. We also observe no routing conflicts and 100% message delivery to both xApps, irrespective of the subscription status of each xApp within the Near-RT RIC.

This improved routing mechanism was deployed in a related work concerning the development of an AI/ML-based end-to-end O-RAN solution for policy-based RAN slicing. A demonstration of this closely-related work was premiered at the CCI exhibition booth during the Mobile World Congress held in September 2022 (**Contribution 7**). Another related work titled "RIC-O: An Orchestrator for the Dynamic Placement of a Disaggregated RAN Intelligent Controller" on optimizing the positioning of disaggregated Near-RT RIC components to reduce the control loop signaling latency over the E2 interface was demonstrated at the IEEE INFOCOM Conference (**Contribution 8**) [26].

In the following chapter, we analyze further security aspects of the Near-RT RIC that involve E2AP messaging over the E2 interface and utilize the multi-xApp RAN control capability enabled through our tag-based message routing mechanism to monitor the signaling latency of control messages over the E2 interface.

Chapter 6

Near-RT RIC Security

This chapter studies the resiliency and security aspects of the Near-RT RIC platform for E2 signaling by leveraging the message routing mechanism implemented in the previous chapter. In particular, we investigate the scope for connected E2 Nodes to orchestrate an application-level DoS attack targeting the Near-RT RIC over the southbound E2 interface. Furthermore, we extend the multi-xApp RAN control capability demonstrated in the previous chapter to implement a latency monitoring xApp to monitor the health of the E2 interface and detect unusual signaling activity over the E2 interface that might lead to degraded performance or complete unavailability of the Near-RT RIC. Although we highlight strategies to mitigate such availability attacks, a complete preventive solution is out of scope for this thesis work.

6.1 Security in the E2 Interface

The O-RAN Alliance has defined various security specifications to ensure confidential transmission and reception of information over the E2 interface [27]. This requires the use of Internet Protocol (IP) Security (IPSec) on the IP network layer to support confidentiality, integrity, replay protection, and data origin authentication [28]. We note, however, that the OSC Near-RT RIC does not yet provide any APIs to implement these security features for its E2 communications. Even then, although the specifications guarantee the security until the transport layer, they do not outline any safeguards to detect or prevent attacks

targeting the application layer of the E2 interface implemented using the E2AP. This allows a threat actor to target the Near-RT RIC over the application layer, if not over the network, or transport layers. Figure 6.1, for instance, highlights a sample E2AP SETUP REQUEST message exchanged between the E2 Node and the Near-RT RIC over the E2 interface. One can observe that this E2AP application layer message is exchanged in cleartext between the Near-RT RIC and the E2 Node increasing the probability for a potential man-in-the-middle attack.

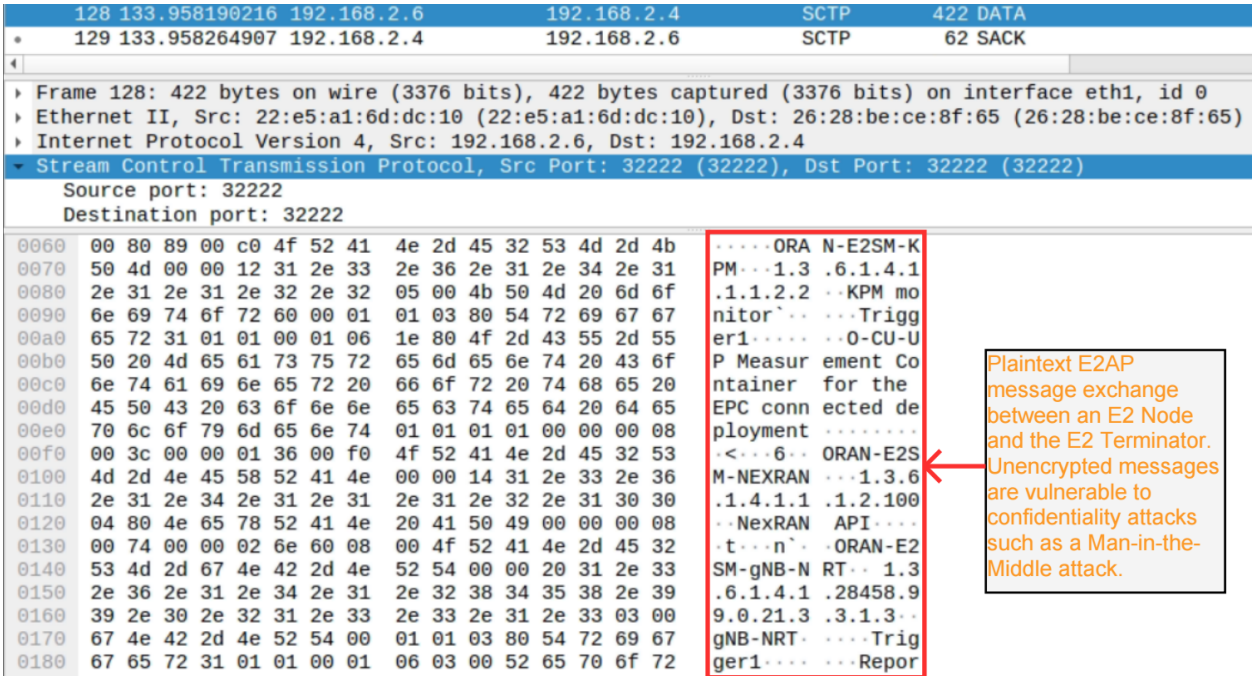


Figure 6.1: Wireshark packet capture of E2AP signaling on the E2 interface.

As discussed in Section 2.1.3, the E2 Terminator acts as the gateway for the E2 interface at the Near-RT RIC, moderating all signaling between the E2 Node and other components within the Near-RT RIC. In the OSC Near-RT RIC implementation, the E2 Terminator is deployed as a microservice listening for SCTP connections on the service port 36422, and the node port 32222. The node port allows entities external to the Near-RT RIC to connect to the E2 Terminator. While this setting is necessary to expose the Near-RT RIC

for southbound control of E2 Nodes, it also makes the E2 Terminator, and consequently, the Near-RT RIC, vulnerable to DoS attacks exploiting the O-RAN-specified E2AP signaling protocol on the application layer.

6.2 E2 Application layer DoS Attacks

Although it is possible to detect and prevent DoS attacks on the network and transport layers, application layer DoS attacks are more challenging to identify or mitigate since the attack payload looks pretty similar to normal signaling traffic [29]. Such attacks tend to disrupt extant communications through resource exhaustion, session starvation, or timeout exploitation. We analyze two strategies that an attacker can utilize to exploit the E2 interface for orchestrating application layer DoS attacks on the Near-RT RIC.

6.2.1 Resource Exhaustion DoS Attack

This approach involves performing a volumetric DoS attack on the E2AP application layer by flooding the E2 Terminator with E2 SETUP REQUEST messages. If the E2 Terminator is not able to quickly handle all incoming messages, it starts queueing messages, thereby resulting in increased resource usage (memory or CPU cores). This might have undesirable effects on the Near-RT RIC such as resource starvation for other critical Near-RT RIC components or resource exhaustion leading to an application crash of the E2 Terminator microservice. Figure 6.2 shows an example workflow of a resource exhaustion DoS attack on the Near-RT RIC using the simple E2 Setup procedure.

The attack targets the E2 Terminator in the Near-RT RIC servicing an existing E2 Node with live subscriptions by one or more xApps (①). This attack can be orchestrated by a

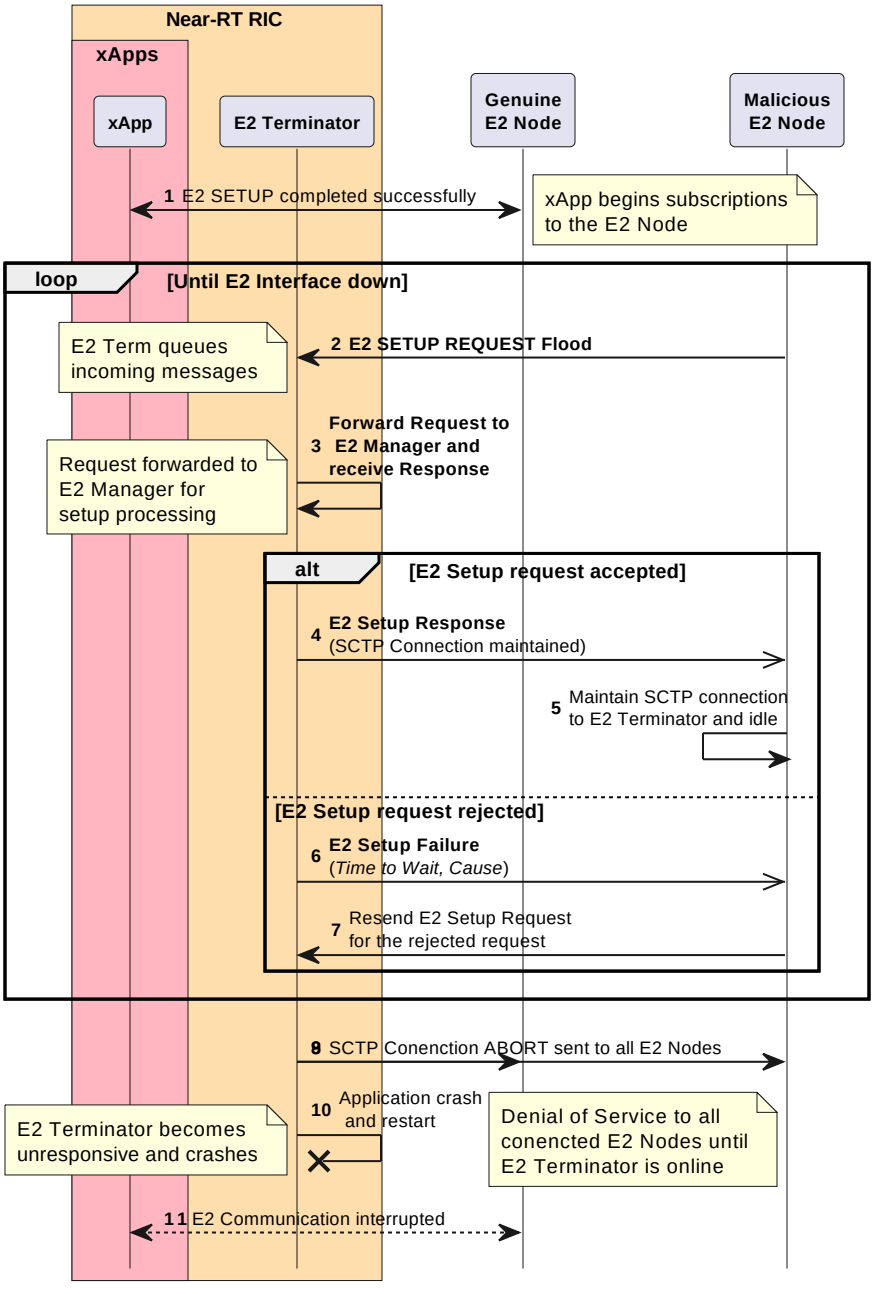


Figure 6.2: Sequence workflow for a Resource Exhaustion DoS attack on the Near-RT RIC exploiting the E2 Setup procedure.

single malicious E2 Node within the same network as the Near-RT RIC or a distributed "botnet" of E2 Nodes targeting the same Near-RT RIC instance. The E2 Node continuously sends a flood of E2 SETUP REQUEST messages to the E2 Terminator (②) in an attempt to open up as many E2 Interface connections as possible or until the E2 Terminator is no longer able to serve any more connections. The E2 Terminator forwards all connection setup requests to the E2 Manager (③) in a sequential fashion since the addition of E2 Nodes involves atomic operations to the Near-RT RIC platform database. Therefore, flooding the E2 Terminator with E2 SETUP REQUEST messages leads to an increase in Near-RT RIC resource utilization (such as memory or CPU cores). The attacking E2 Node maintains the open SCTP connection in case of a successful outcome (④ and ⑤) and can conditionally reissue a new request if the Near-RT RIC rejects the setup request (⑥ and ⑦). Upon breaching a certain threshold, the E2 Terminator becomes unresponsive, closing existing connections to all peer E2 Nodes (⑧ and ⑨) thereby causing horizontal service unavailability (⑩). Although such a scenario could be considered highly improbable owing to the stringent O-RAN security specifications [27] for the network and transport layers, it could still leave damaging effects on the Near-RT RIC if successfully orchestrated because of the percolation of signaling into other platform components such as the E2 Manager, SDL, and the Routing Manager. Considering the nature of all volumetric attacks, solutions attempting to detect and thwart this type of DoS attack need to be localized in the E2 Terminator in the interest of effecting a quick response. A primitive solution to mitigate the effects of this attack could be throttling E2 Setup signaling beyond a threshold tolerable to the E2 Terminator service.

6.2.2 Signaling Storm DoS Attack

The signaling storm DoS attack can be orchestrated over the E2 interface by flooding the E2 Terminator with RIC INDICATION messages from one or more connected E2 Nodes. Com-

pared to the resource exhaustion DoS attack discussed earlier, this attack targets timeout exploitation for time-critical signaling messages between xApps and subscribed E2 Nodes. Such conditions could arise due to a compromised E2 Node or could even result from an unprecedented increase in signaling load handled by the Near-RT RIC. Figure 6.3 shows the attack workflow for a sample signaling storm DoS. This attack can lead to either degradation or complete denial of the E2 Terminator service to the E2 Nodes and xApps.

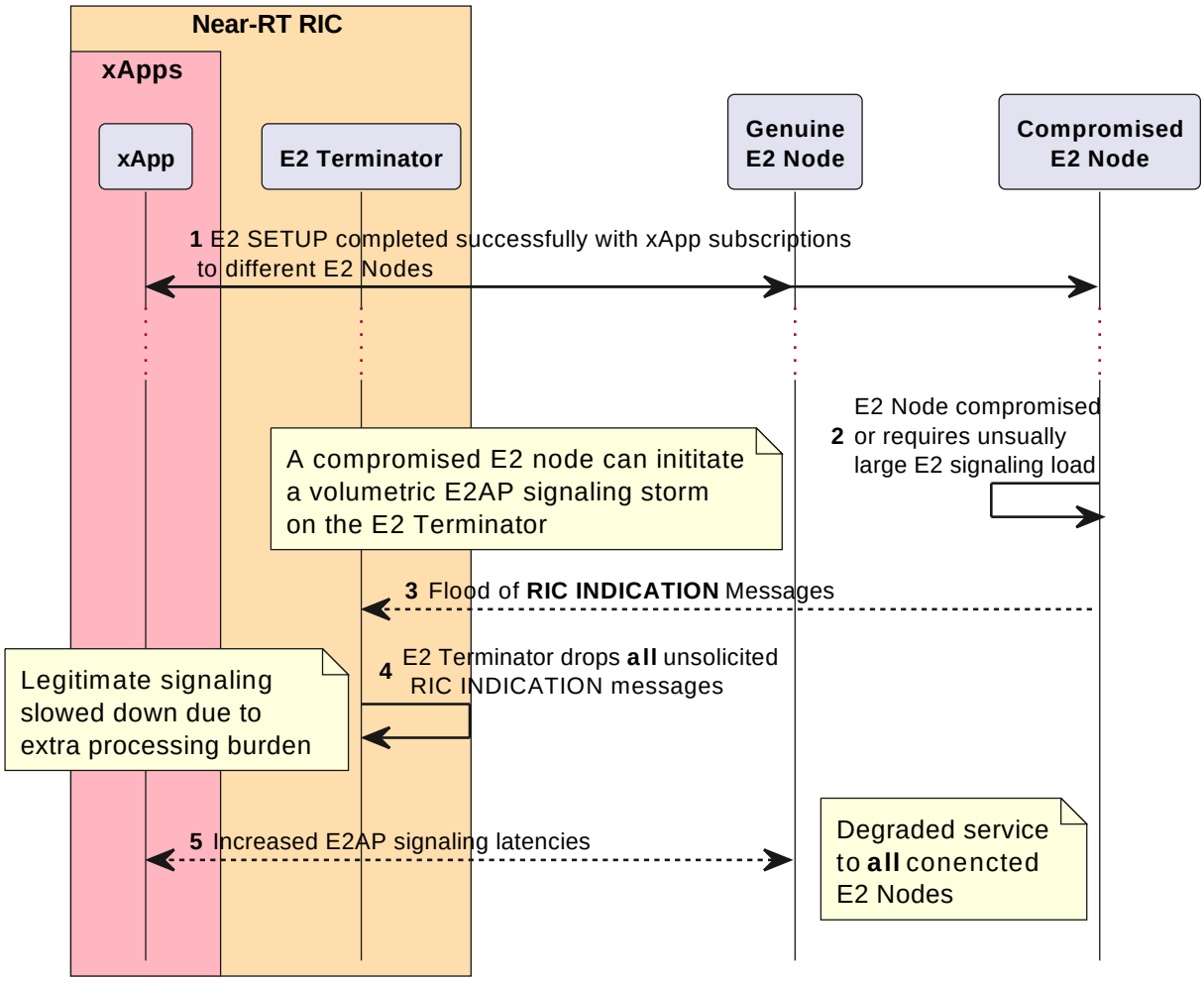


Figure 6.3: Sequence workflow for a Signaling Storm DoS attack.

In this attack, a malicious E2 Node without a live subscription or a compromised E2 Node connected to the Near-RT RIC and with subscriptions (① and ②) could flood the E2 Termi-

nator with bogus RIC Indication messages with a large message payload (③). This barrage of messaging traffic overwhelms the E2 Terminator and forces it to perform waste work (④). In the process, the service gets bogged down and starts lagging on other time-critical signaling between other E2 Nodes and their target xApps (⑤). This could lead to increased latencies in services between other E2 Nodes and xApps and eventually trigger control loop violations, resulting in service degradation. If the E2 Terminator is not able to handle this anomalously heavy signaling traffic, it could go down, causing DoS on all connected E2 Nodes. Since this attack vector only focuses on the E2 Terminator service, it can be comparatively easily detected and mitigated, for example, using an xApp-based latency monitoring solution, as discussed in the following section.

6.3 xApp-based E2 Application Latency Monitoring

Since xApps operate in the near-real-time control loop, they can be leveraged to monitor the status of the E2 interface by checking the signaling latency between the E2 Terminator service and multiple connected E2 Nodes. The OSC provides an open-source xApp called Bouncer that is claimed to benchmark the Near-RT RIC using E2 signaling latencies. However, we observe that the latency calculated by the Bouncer xApp only involves the processing delay between the reception of a message by the xApp and the transmission of the processed message back to the E2 Node. To perform an application layer "ping" on the target E2 Node, we implement the RC E2SM to develop a new E2 latency monitoring xApp for the Near-RT RIC. As we shall further see in chapter 7, this xApp can also be used to monitor signaling latency over multiple E2 Nodes connected to the Near-RT RIC.

Chapter 7

Orchestrating E2 DoS Attacks

This chapter assesses the resiliency of the Near-RT RIC platform against DoS attacks orchestrated over the E2 interface. In particular, we study the effects of the DoS strategies discussed in the previous chapter on the OSC Near-RT RIC and characterize the behavior of the platform components affected in such scenarios. As part of this research study, we develop a DoS attack agent for a malicious E2 Node actor (**Contribution 5**). We also describe the latency monitoring xApp discussed in Section 6.3 to detect and analyze signaling storm conditions on the Near-RT RIC (**Contribution 4**).

7.1 Experimental Setup

We study DoS attacks on the Near-RT RIC using the proof-of-concept experimental setup shown in Figure 7.1. The components responsible for and affected by the DoS attack are highlighted inside a red dashed box. The setup consists of the following key components:

- DoS attack agent: The malicious E2 Node(s) simulation agent capable of orchestrating a DoS attack on the Near-RT RIC;
- E2 Nodes: srsRAN-based benign E2 Nodes that utilize RAN control and optimization capabilties provided by the Near-RT RIC;

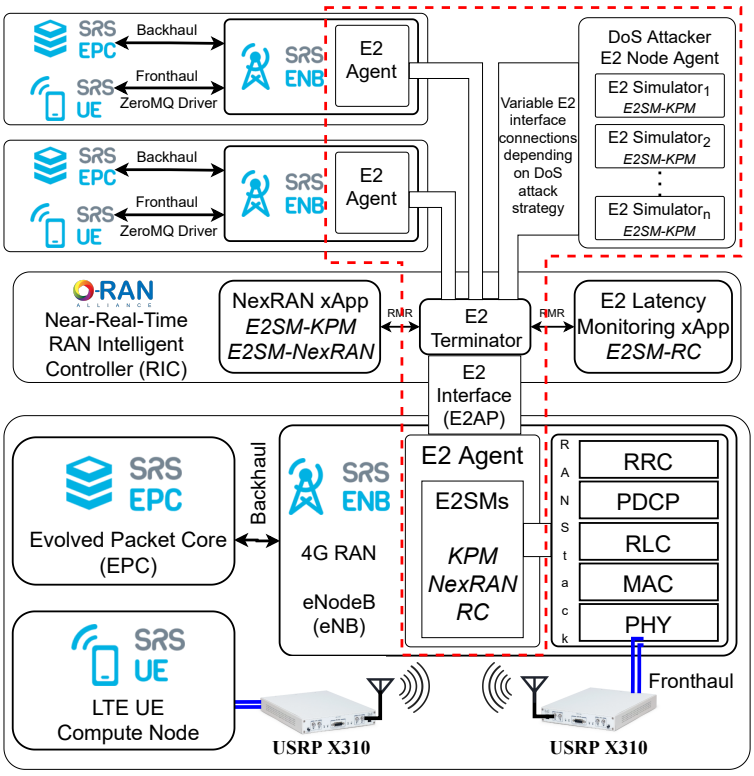


Figure 7.1: Proof-of-concept experimental setup for orchestrating DoS attacks.

- Near-RT RIC: Hosts all platform components including the E2 Terminator, responsible for enabling xApp-E2 Node communication over the E2 interface;
- xApps: Multiple xApps that subscribe to one or more E2 Nodes.

The following subsections delve into how each component involved in the attack is implemented or impacted.

7.1.1 DoS Attack Agent

We design an attack agent aimed at orchestrating the strategies outlined in Sections 6.2.1 and 6.2.2. In positive terms, the attacker is represented as a performance benchmarking tool to assess the scalability of the Near-RT RIC to simultaneously support multiple E2

Nodes. The OSC provides an E2 Node simulator that instantiates a 5G Next Generation NodeB (gNB) implementing the E2AP over the E2 interface. This application, however, has two limitations: 1) It only allows a single E2 Node instance to connect to the Near-RT RIC, and 2) It uses an older version of the E2AP (v1.01) which is incompatible with the G-Release of the Near-RT RIC. We enhance this application by allowing support for multiple E2 Node instances through multi-threading along with E2AP v2.03 support. The attacker uses the KPM E2SM to generate RIC INDICATION messages of type REPORT and flood the Near-RT RIC, resulting in an E2AP signaling storm. Currently, the DoS agent uses multi-threaded E2 Node instances to orchestrate the E2 Setup-based resource exhaustion attack and a single-threaded E2 Node instance to realize the RIC Indication-based signaling storm attack.

7.1.2 E2 Nodes

We deploy E2 Nodes running the srsRAN LTE software radio stack and connected to the Near-RT RIC to study the effects of the DoS attack on the platform components. The E2 Nodes are deployed on physical and virtualized compute nodes situated in geographically distinct locations. As depicted in Figure 7.2, one E2 Node is located in the same deployment site (CCI xG Testbed, Arlington, Virginia) as the Near-RT RIC, and two other E2 Nodes are deployed in the Virginia Tech Blacksburg campus. All E2 Nodes connect internally to their own EPC through the S1 interface and additionally, to the Near-RT RIC through the E2 interface established using a Virtual Private Network (VPN) hosted at the controller site. While all E2 Nodes are capable of interfacing with SDR radio front-ends, only two E2 Nodes, one located at the xG Testbed, and the other deployed on the CORNET Testbed in Blacksburg, are configured to use the USRP X310 SDRs for Physical (PHY) layer processing. All other E2 Nodes simulate the PHY layer using the ZeroMQ messaging library [30]. UEs

are instantiated using the srsUE application and are configured to use either USRP X310 SDRs for over-the-air transmission or the ZeroMQ library for simulated connectivity.

Although all E2 Nodes deployed in this experiment run a monolithic eNB, the experiment is totally 5G-compatible in that the E2 Nodes can support a gNB in addition to the eNB in a Non-Stand-Alone (NSA) or Stand-Alone (SA) configuration. We limit our deployment to LTE networks since the open-source NexRAN xApp only supports subframe slicing control for eNBs over the E2 interface. [1]

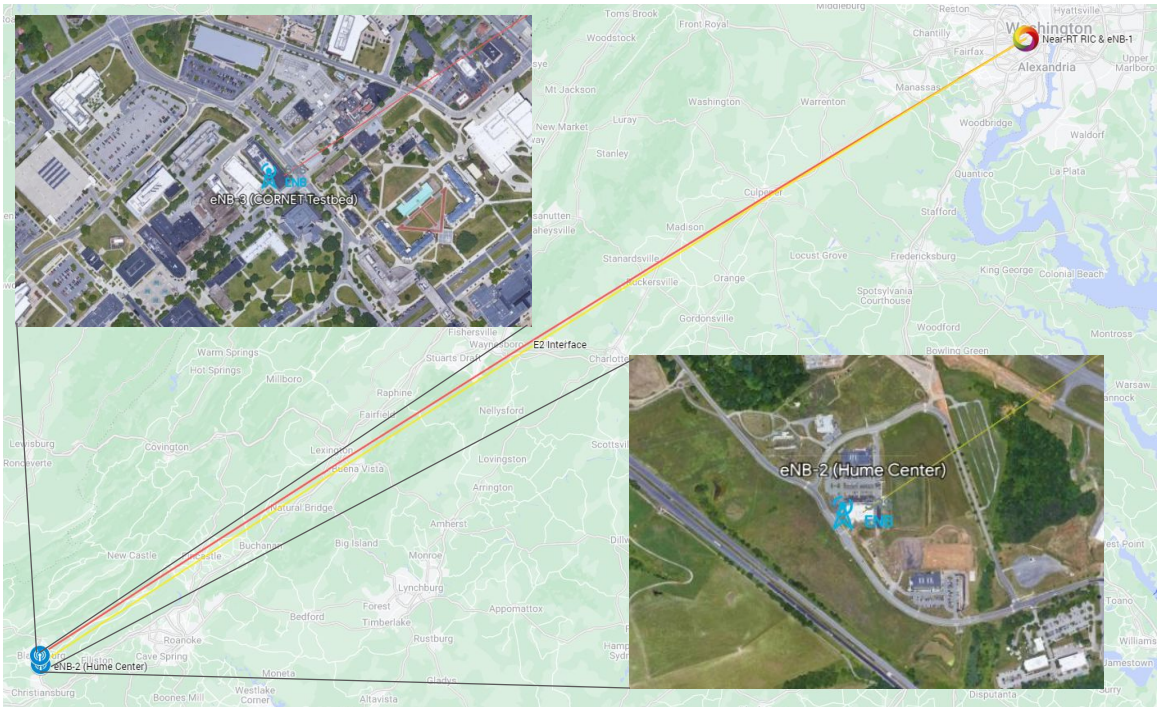


Figure 7.2: Location of Near-RT RIC and E2 Nodes for the experimental setup.

7.1.3 Near-RT RIC

The Near-RT RIC platform is deployed as a single-node Kubernetes cluster with all platform components managed as microservices using Docker containers. We run the G-Release of the

[1] The NexRAN xApp does not support slicing for 5G networks since 5G schedulers implement variable numerologies for slot-based slicing, different from subframe slicing for 4G schedulers.

OSC Near-RT RIC that supports E2AP v02.03. The default deployment recipe launches the core platform components with a single instance (termed a replica in Kubernetes parlance). Each component is exposed to the cluster through one or more services with an internal ClusterIP service. However, since the E2 Terminator listens for connections from remote E2 Nodes as well, it is exposed node-wide using the NodePort service type. Persistent data storage for platform components is provided using Redis Database as a Service (DBaaS). The platform also enables basic performance monitoring for the Near-RT RIC deployment using Prometheus. We additionally deploy the InfluxDB time-series database for authorized xApps to write and read application-specific data obtained through subscriptions to the E2 Node or other platform components. Finally, all platform components that interact with one another utilize the new tag-based messaging routing architecture to enable simultaneous RAN control through multiple xApps.

7.1.4 xApps

The NexRAN xApp is deployed to perform closed-loop RAN slicing on the benign E2 Node. It is configured to throttle slice shares to a set target throughput based on the throttling period and the bandwidth usage of a network slice. We also deploy the latency monitoring xApp discussed in Section 6.3 to monitor the baseline E2 control loop signaling latency between the Near-RT RIC and other legitimate E2 Nodes. Additionally, we set up the xApp to perform simultaneous latency monitoring on all E2 Nodes connected to the Near-RT RIC. This would help make a comparative analysis of the status of the E2 interface at the Near-RT RIC by factoring in the latencies on all connected links. As we shall see in the following section, the E2 signaling latency determined by this xApp is used to flag a signaling storm attack on the E2 Terminator of the Near-RT RIC.

7.2 PoC DoS Attack Workflow

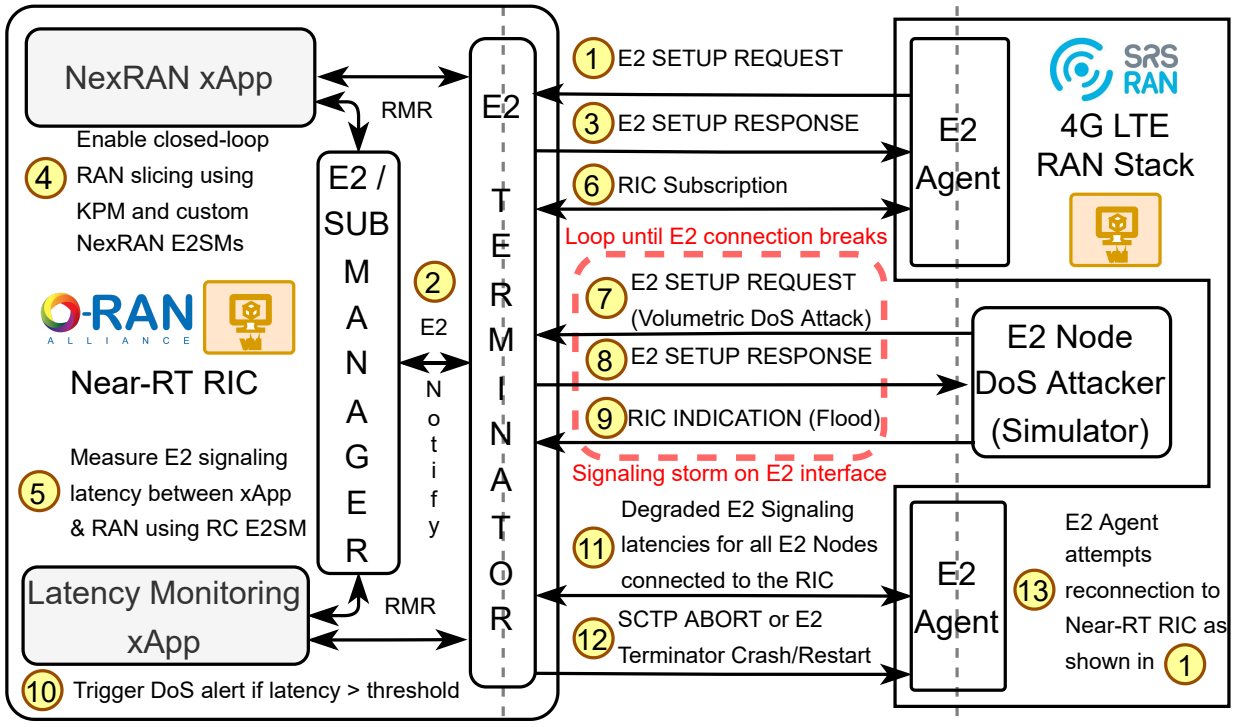


Figure 7.3: PoC DoS attack workflow on the experimental setup.

The proof-of-concept workflow for the DoS attacks demonstrated and studied using the experimental setup discussed in the previous section is shown in Figure 7.3.

The expected use case involves legitimate E2 Node(s) connecting to the E2 Terminator on the Near-RT RIC. The E2 Nodes complete the E2 Setup Procedure with the Near-RT RIC (①, ②, and ③). It must be noted that the admission of an E2 Node into the Near-RT RIC is controlled by the E2 Manager, and not directly by the E2 Terminator. Following this step, all registered xApps subscribe to the connected E2 Nodes based on their control configurations. In our setup, we deploy the NexRAN xApp and configure it to control the RAN slicing behavior using the MAC scheduler on a single connected E2 Node (④). Multiple other legitimate E2 Nodes from different locations also connect to the Near-RT RIC using the

E2 Setup Procedure. Additionally, the latency monitoring xApp checks the SDL for all connected E2 Nodes and sends subscriptions to all of them for measuring the E2 control signaling latency using the RC E2SM (⑤). The subscriptions from both xApps are forwarded to the E2 Terminator and routed to the corresponding E2 Nodes (⑥). The E2 Nodes then begin sending RIC INDICATION messages of type REPORT or INSERT depending on the nature of the subscriptions from the xApps. In Figure 7.3, sequences ① to ⑥ cover the interactions for the legitimate operation of the end-to-end O-RAN mobile network. We observe the performance metrics of the Near-RT RIC and the E2 interface at this stage and consider these values our baseline for further comparative assessment during the DoS attack. The relevant performance metrics are described in Chapter 8.

At this stage, we initiate the DoS attack on the Near-RT RIC. The DoS agent initially spawns multiple E2 Node simulator instances and sends a flood of E2 SETUP REQUEST messages targeting the E2 Terminator (⑦ and ⑧). This approach is aimed at exhausting the Near-RT RIC cluster's compute resources such as memory and CPU. If the request is accepted, then the E2 connection is maintained indefinitely, draining the resources of the Near-RT RIC. Additionally, the DoS agent can also launch a signaling storm by flooding the E2 Terminator with RIC INDICATION messages without getting any subscriptions from the Near-RT RIC (⑨). The E2 Terminator wastes its resources to parse the incoming messages to determine that the messages are invalid and discard them all. At this point, the latency monitoring xApp detects anomalies in the control signaling latency to flag conditions stressing the E2 Terminator (⑩). If this condition is left unaddressed, time-critical control messaging between the xApps and multiple legitimate E2 Nodes could take a hit leading to abnormally high latencies compromising the control loop criteria for near-real-time operations such as mobility handoff or UE admission. This leads to service degradation, or, in the worst case, Near-RT RIC service unavailability due to application crash and restart of the

E2 Terminator microservice.

In the following chapter, we describe the metrics used to characterize the DoS attack and analyze the results of the orchestrated attacks on the proof-of-concept experimental setup described in this chapter.

Chapter 8

Metrics and Results

This chapter studies the metrics necessary to characterize volumetric DoS attacks on the OSC Near-RT RIC. As part of the results, we also assess the resiliency of the Near-RT RIC platform components when subjected to application layer DoS attacks (**Contribution 6**).

8.1 Metrics

The core metrics employed to study the Near-RT RIC platform components include: 1) the CPU and memory usage of the containers running the microservices and; 2) the network usage of the Kubernetes pods. We fetch these metrics from the Prometheus monitoring server available within the OSC Near-RT RIC cluster. In addition to these metrics, we also monitor the message counter internal to the E2 Terminator to track the number of different E2 messages processed by it. The E2 Terminator application exposes these counter values as Prometheus metrics that are scraped by the Prometheus server outlined earlier. Furthermore, we fetch the application layer latency measurements between the latency monitoring xApp and the E2 Nodes from the InfluxDB. For confirmation of any signaling storm conditions in the application layer DoS attack, we also record the network latency using the ping plugin of the Telegraf utility for InfluxDB. All collected metrics are visualized using the open-source Grafana dashboard utility.

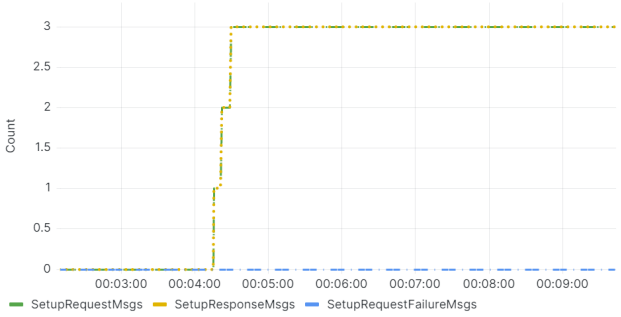
8.2 Results

This section elaborates on the performance metrics of the Near-RT RIC and the E2 interface during normal operation of the experimental setup described in Section 7.1 and when subjected to resource exhaustion and signaling storm DoS attacks. We assess the E2 interface performance primarily based on signaling latency and Near-RT RIC performance via resource metrics such as CPU, memory, and network usage within the platform.

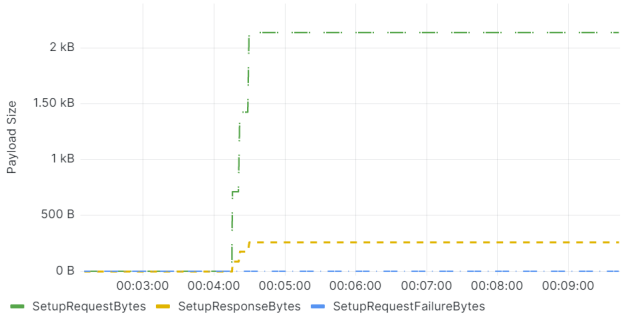
8.2.1 Normal Operation

Figures 8.1 and 8.2 depict the metrics discussed in the previous section. While Figure 8.1 displays the various message counter states at the E2 Terminator, Figure 8.2 shows the CPU and memory resource utilization, along with the network utilization of those platform components and entities involved in facilitating communications between xApps and one or more E2 Nodes. The Near-RT RIC platform consumes negligible resources until the first E2 Setup procedure is triggered. At about 4 minutes into normal operations, the E2 Nodes (one situated locally alongside the Near-RT RIC deployment, and two other situated in the Blacksburg campus) connect to the Near-RT RIC one after the other, as shown in Figure 8.1a. From Figure 8.1b, we observe that the size of a setup request is around 700 bytes, while that of a setup response message is around 100 bytes. We also notice a slight increase in the memory and CPU consumption of the E2 Terminator and the E2 Manager, shown in Figure 8.2a. This is expected since all incoming setup request messages over the E2 interface are forwarded by the E2 Terminator to the E2 Manager for acceptance or rejection of the requesting E2 Node. The network utilization by the E2 platform components, shown in Figure 8.2b, confirms the same behavior.

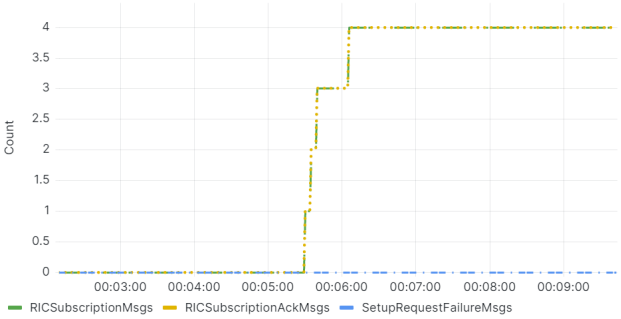
After about 5 minutes of normal operation, the NexRAN and Latency Monitoring xApps are



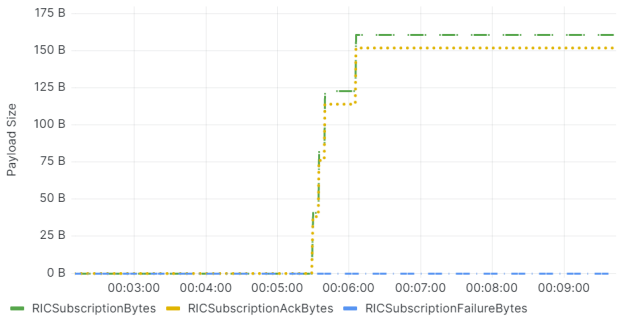
(a) E2 Setup messages.



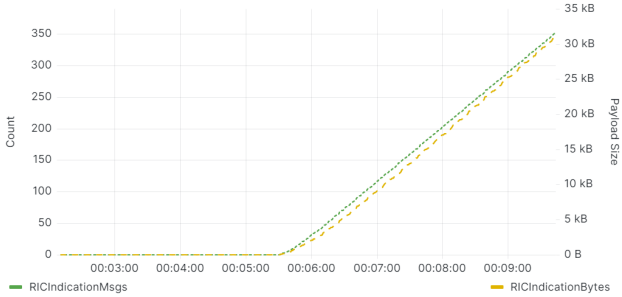
(b) E2 Setup message payload.



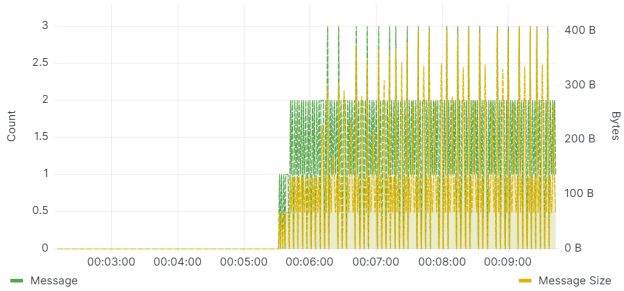
(c) RIC Subscription messages.



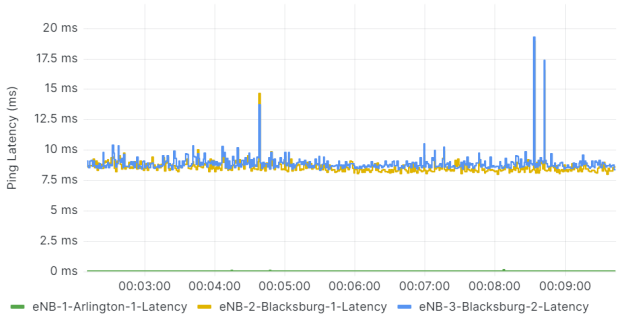
(d) RIC Subscription message payload.



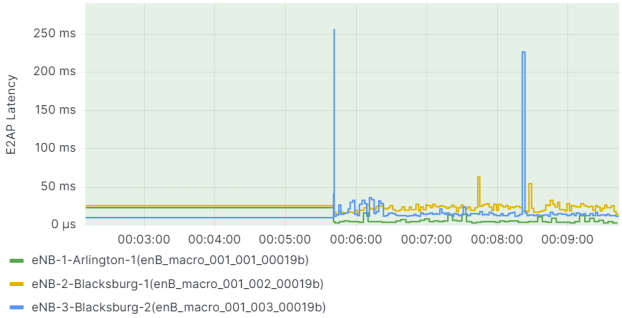
(e) RIC Indication messages.



(f) RIC Indication messages per second.

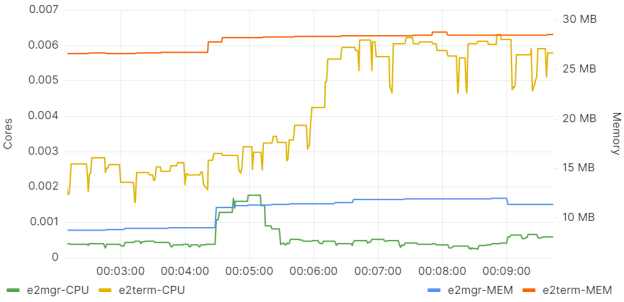


(g) E2 interface network layer latency.

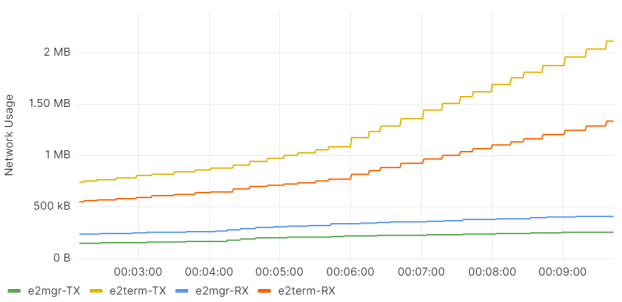


(h) E2 interface application layer latency.

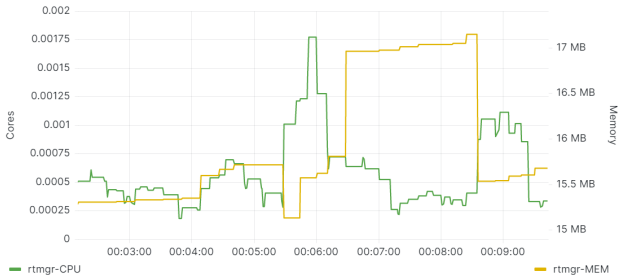
Figure 8.1: E2 Terminator message counters and latencies during normal operation.



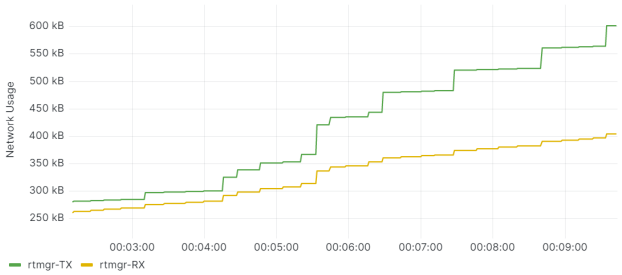
(a) E2 application resource utilization.



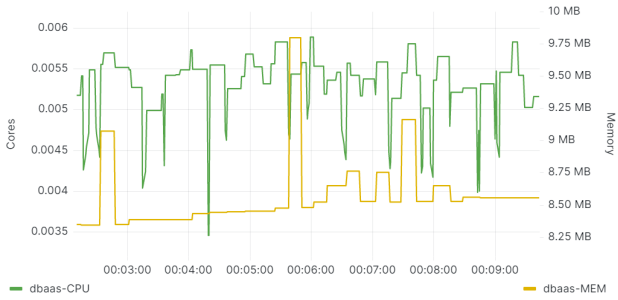
(b) E2 application network utilization.



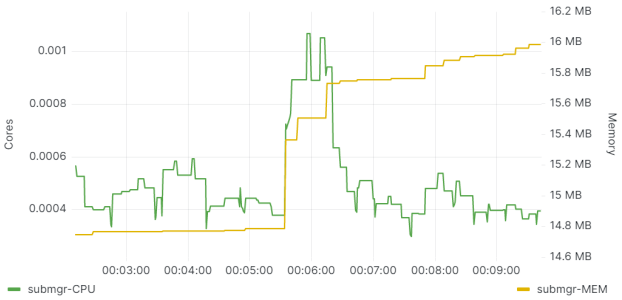
(c) Routing Manager resource utilization.



(d) Routing Manager network utilization.



(e) DBaaS application resource utilization.



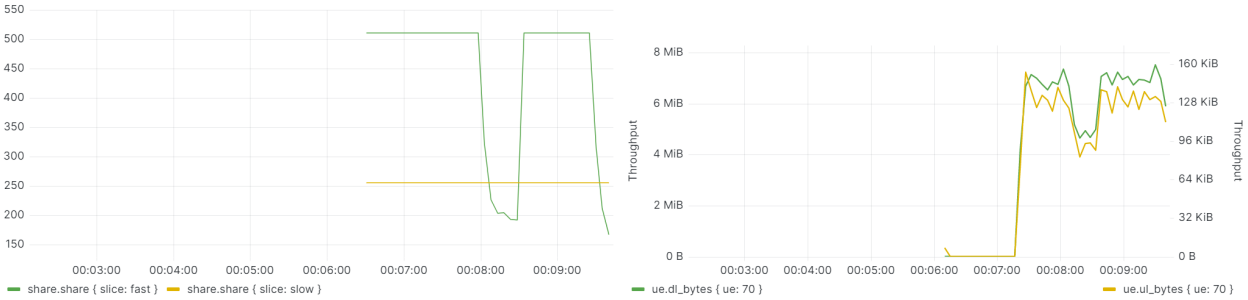
(f) Subscription manager resource utilization.

Figure 8.2: Performance statistics of Near-RT RIC components during normal operation.

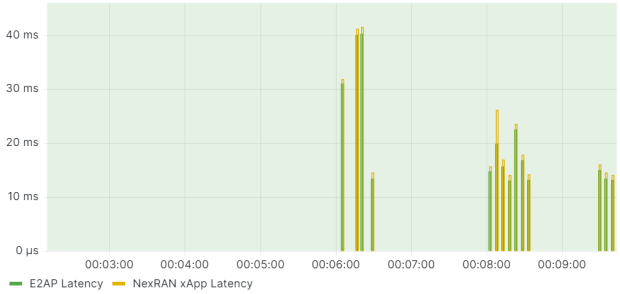
deployed sequentially. This is validated by Figures 8.1c and 8.1d, showing the subscription message count and size, respectively. While the NexRAN xApp targets one of the E2 Nodes located at the CORNET testbed in Blacksburg, the Latency Monitoring xApp subscribes to all E2 Nodes connected to the Near-RT RIC. The subscription procedure triggers activities on the Subscription Manager, E2 Terminator, Routing Manager, and the DBaaS platform components, leading to a marginal increase in memory and CPU consumption (shown in Figures 8.2c-8.2f). Upon successful subscription, the E2 Nodes begin to send RIC Indications

to the subscribed xApps through the E2 Terminator. Figure 8.1f shows the number of such messages handled by the E2 Terminator and the corresponding payload size. Furthermore, Figure 8.1e shows a per-second delta for the processed indications. Since each E2 Node receives subscriptions requiring RIC Indications with a frequency between two and four seconds, the E2 Terminator processes up to 3 indication messages every second with the payload size remaining below 400 bytes. Figure 8.1g plots the network layer latency between the compute node hosting the Near-RT RIC cluster and each compute node hosting the E2 Node (running the srsRAN application suite) using the ping application. The E2 Node collocated with the Near-RT RIC platform experiences less than 1 ms latency while the two E2 Nodes located in the Blacksburg campus experience an average latency of $8-10$ ms. There is some network jitter causing the latency to increase momentarily to about $20ms$. Although negligible, this jitter is expected since we run the E2 interface over the OpenVPN VPN link. Figure 8.1h shows the application layer latency determined by the Latency Monitoring xApp over the E2 interface. The values indicate some jitter along E2 interface connections to a specific E2 Node (eNB-3 in this case). Otherwise, the application layer latency follows the network layer latency with an added delay of approximately 4 ms.

After subscribing to the E2 Node at around the sixth minute of the experiment, the NexRAN xApp starts receiving RIC Indications from its target E2 Node and begins to enforce closed-loop throttling of the configured RAN slices based on predefined constraints for the slice profile. Figures 8.3a-8.3c show the slice share status, UE throughput status, and the control plane signaling latency for the xApp, respectively. We particularly note that the signaling latency lies under 50 ms satisfying the near-real-time control loop requirements of 10 $ms-1$ s.



(a) Slice share configuration by NexRAN xApp. (b) UE throughput control through RAN slicing.

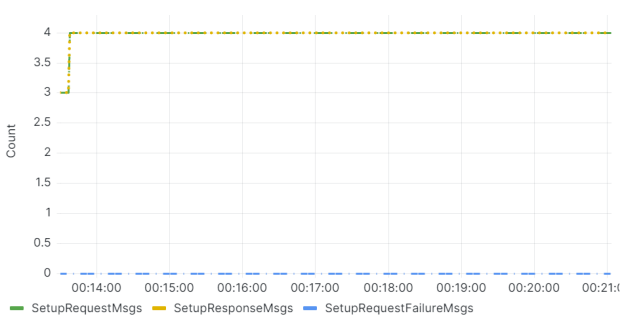


(c) NexRAN xApp control plane signaling latency.

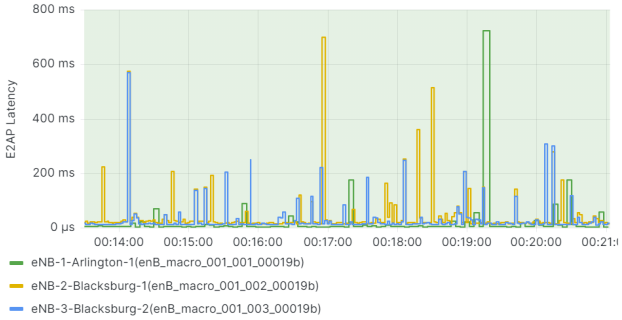
Figure 8.3: NexRAN xApp statistics during normal operation.

8.2.2 E2 Signaling Storm

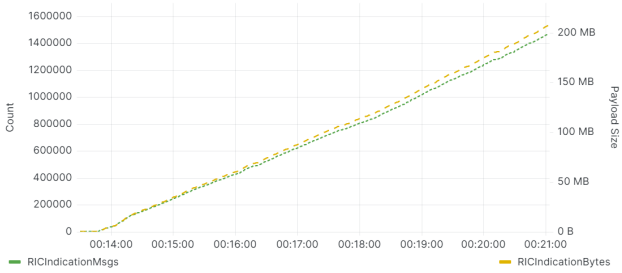
Figure 8.4 characterizes the signaling storm attack on the Near-RT RIC and the E2 interface between the E2 Terminator and E2 Nodes. For achieving the signaling storm scenario on the E2 interface, the DoS attacker agent connects to the E2 Terminator with a single E2 Node instance (Figure 8.4a). Shortly after successful E2 setup, the attacker launches a flood of RIC Indication messages targeting the Near-RT RIC as shown in Figure 8.4c. This can better be visualized through Figure 8.4d, which depicts a best-effort message flooding approach topping around 6000 messages every second. Considering that the RIC Indication message generated by the attacker is 141 bytes in length, we estimate a peak throughput of 846 kB of attack traffic directed towards the E2 Terminator. Although this traffic volume pales in comparison to the traffic volume in typical volumetric DoS attacks, we point to the fact



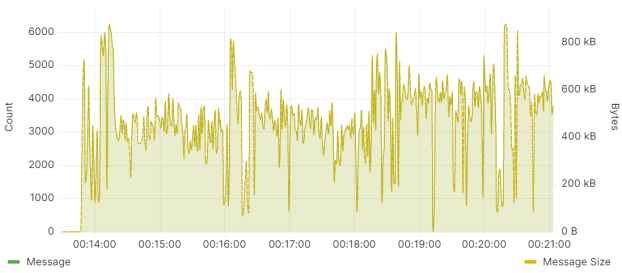
(a) E2 Setup message count.



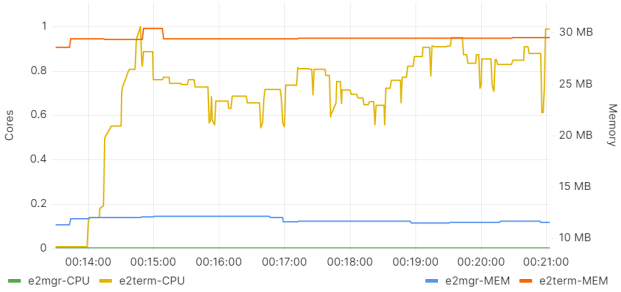
(b) E2 interface application layer latency.



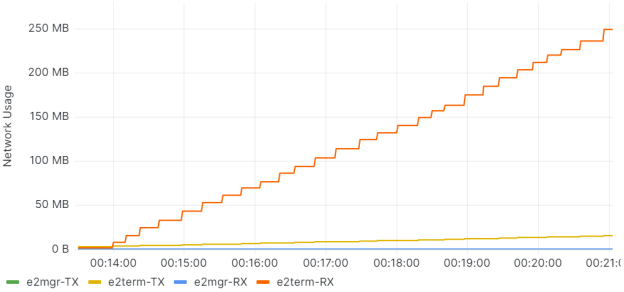
(c) RIC Indications.



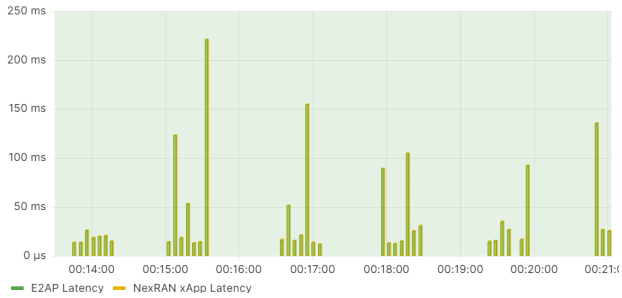
(d) RIC Indications per second.



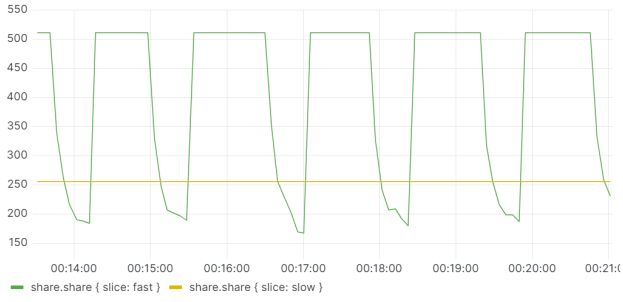
(e) E2 application resource utilization.



(f) E2 application network utilization.

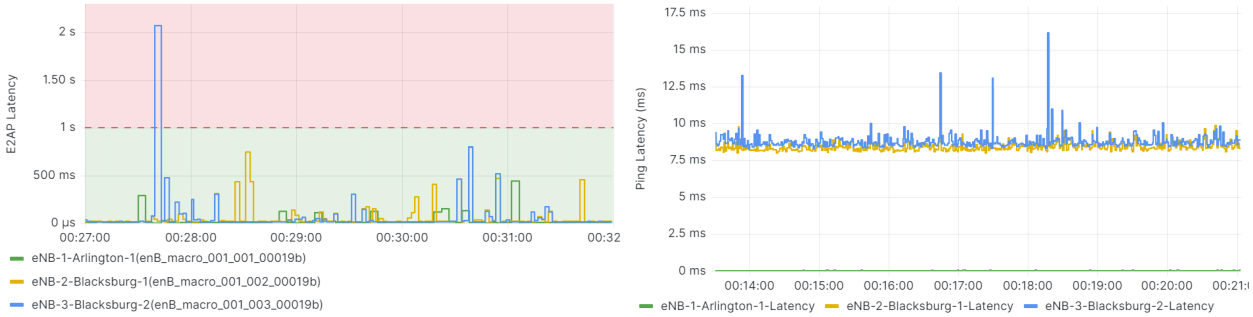


(g) NexRAN control signaling latency.



(h) Imprecise slice share control.

Figure 8.4: Signaling storm attack characteristics on the Near-RT RIC and the E2 interface.



(i) E2 control loop latency violation.

(j) Network layer latency during the attack.

Figure 8.4: Signaling storm attack characteristics on the Near-RT RIC and the E2 interface.

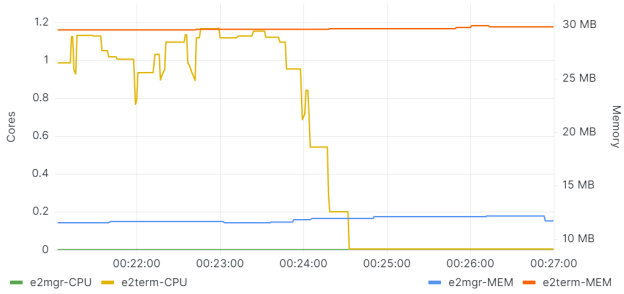
that we target a single Near-RT RIC platform service, namely the E2 Terminator, through the signaling storm. Therefore, we only require a high-intensity yet considerably low-volume attack to cause service disruption on the Near-RT RIC. The processing strain on the E2 Terminator is confirmed from Figures 8.4e and 8.4f, which point to a sudden spike in the CPU usage of the E2 Terminator coupled with high network usage of about 250 MB within a short period of 5 minutes. The CPU usage increases by almost 1000% consuming more than one CPU core within the Kubernetes cluster. From Figures 8.4d and 8.4e, we observe that the resource usage is directly proportional to the volume of attack traffic directed at the E2 Terminator.

The signaling storm attack is detected using the Latency Monitoring xApp. From Figure 8.4b, we observe that during the attack, the control plane signaling latencies of all connected E2 Nodes increases 70 times from approximately 10 ms to 700 ms. This is observable even with the sentinel E2 Node collocated to the Near-RT RIC host. Factoring in the latency reports of all E2 Nodes can inform the Near-RT RIC about the status of the E2 interface and trigger an alert in case a signaling storm attack is detected. Since the anomalous increase in signaling latencies during the signaling storm attack does not scale proportionally between multiple E2 Nodes situated at different geographical locations, the Latency Monitoring xApp can leverage the collected data to train ML models that can de-

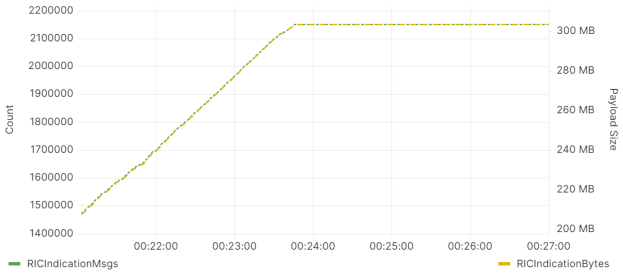
termine a dynamic threshold to detect such attacks on the E2 Terminator. The effects of the signaling storm attack are observable on the NexRAN xApp's functioning. For example, Figure 8.4g shows a 5x increase in the control plane signaling latency which affects the ability of the E2 Node to adjust its slice configuration dynamically, as seen in Figure 8.4h. The effects are prominent between the fifteenth and eighteenth minutes of the experiment, where the control messages from the xApp experience more delay impacting how quickly the slice share can be reconfigured by the NexRAN xApp.

A persistent and elongated signaling storm, as shown in Figure 8.4 could lead to prohibitively higher latencies on the E2 interface. For example, Figure 8.4i shows a signaling latency of 2 s violating the near-real-time control loop requirements of 10 $ms - 1$ s resulting in degradation of service, or, in the worst case, compromise of service availability for all connected E2 Nodes. We confirm the volumetric application layer DoS attack by the fact that the network layer latency (shown in Figure 8.4j) remains fairly unaffected for the duration of the attack.

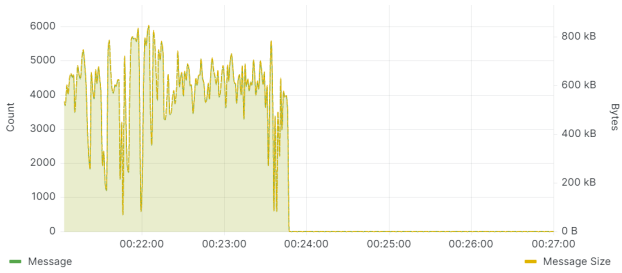
Figure 8.5 characterizes the Near-RT RIC performance after the signaling storm attack is stopped. The E2 Terminator gives up excess CPU usage (Figure 8.5a) since the RIC Indications plateau to the normal messaging frequency (Figures 8.5b and 8.5c). This reflects in the application layer latency measured by the Latency Monitoring xApp as shown in Figure 8.5d. There is also a marked improvement in the control plane signaling latency of the NexRAN xApp, with the latency for all control messages staying within 50 ms (Figure 8.5e). As expected from an application layer DoS attack, the network layer latency remains the same over all periods of the experiment, as shown in Figure 8.5f.



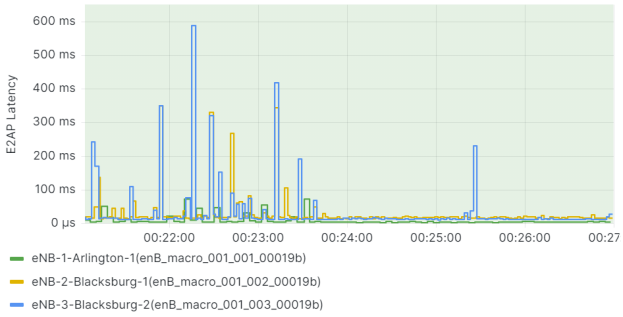
(a) E2 application resource utilization.



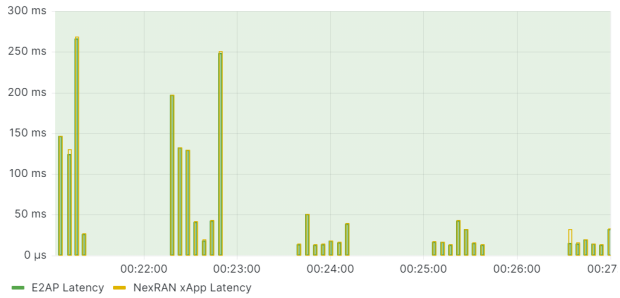
(b) RIC Indications.



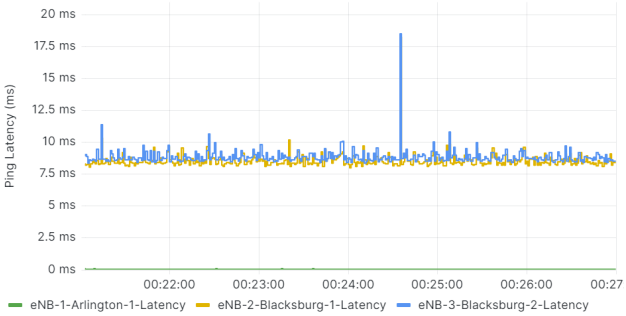
(c) RIC Indications per second.



(d) E2 interface application layer latency.

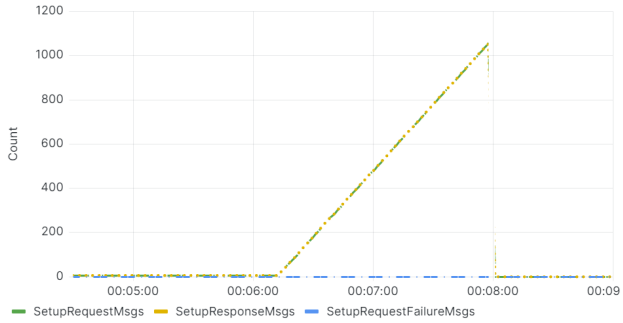


(e) NexRAN control loop signaling latency.

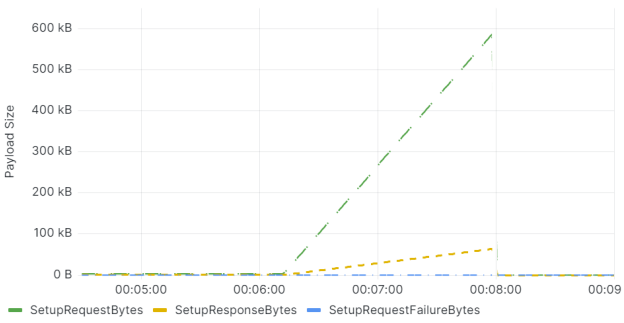


(f) E2 interface network layer latency.

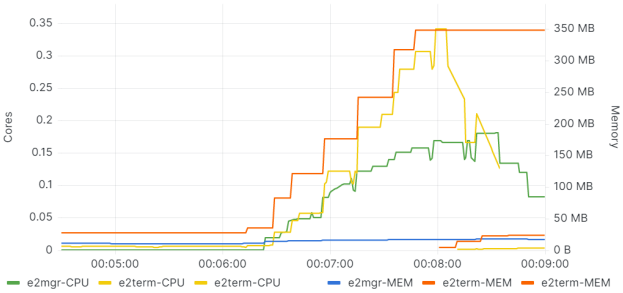
Figure 8.5: Near-RT RIC statistics post signaling storm attack.



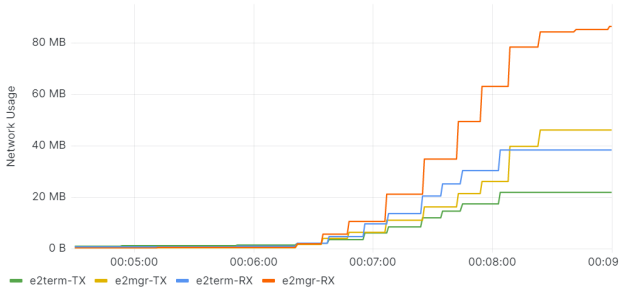
(a) E2 Setup DoS attack message count.



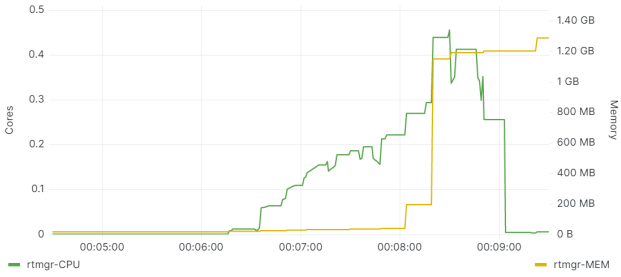
(b) E2 Setup DoS attack message payload size.



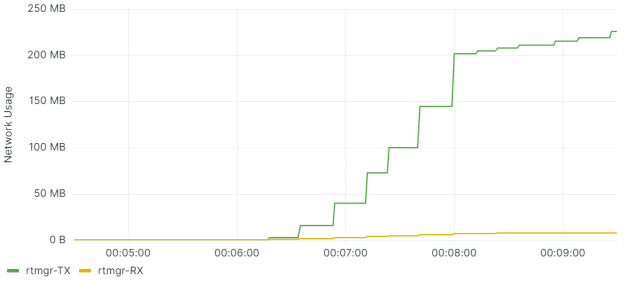
(c) E2 application resource utilization.



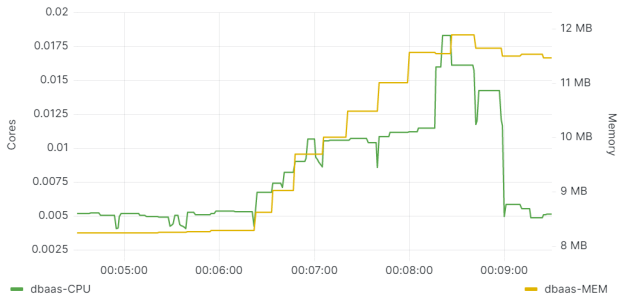
(d) E2 application network utilization.



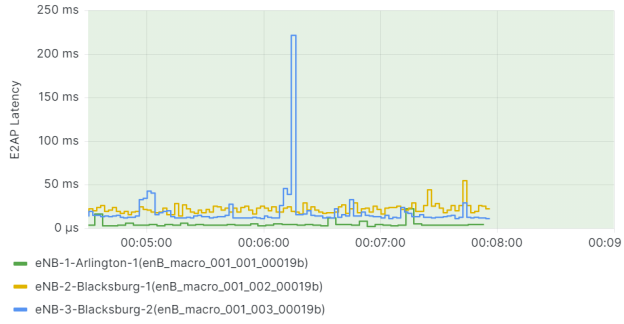
(e) Routing manager resource utilization.



(f) Routing manager network utilization.

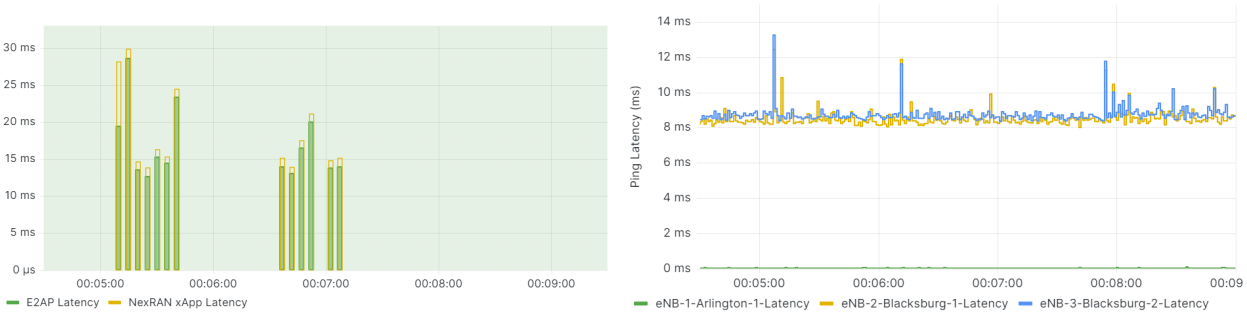


(g) DBaaS application resource utilization.



(h) E2 interface application layer latency.

Figure 8.6: DoS attack characteristics on the Near-RT RIC and the E2 interface.



(i) NexRAN xApp control signaling latency.

(j) E2 interface network layer latency.

Figure 8.6: DoS attack characteristics on the Near-RT RIC and the E2 interface.

8.2.3 E2 Setup DoS Attack

Figure 8.6 shows the Near-RT RIC performance statistics and application layer latency during the E2 Setup DoS attack. We rerun the experiment and ensure normal Near-RT RIC operation as described in Section 8.2.1. Shortly after the sixth minute, the multi-threaded DoS attacker agent spawns multiple E2 Node simulators to send an E2 SETUP REQUEST message from each simulated E2 Node. In our case, we simulate 2000 E2 Nodes connecting to the Near-RT RIC. Figures 8.6a and 8.6b show the number of messages processed by the E2 Terminator. Unlike the signaling storm attack, with multiple E2 Setup messages, we notice a sharp increase in the memory usage (1000% more from 30 MB to almost 350 MB) of the E2 Terminator in addition to increased CPU consumption (Figure 8.6c). We also note a corresponding increase in its network usage (Figure 8.6d). Additionally, we observe the attack spill over to other Near-RT RIC platform components. Notably, the E2 Manager, Routing Manager, and the DBaaS applications experience a sharp increase in CPU utilization shortly after the attack targets the E2 Terminator (Figures 8.6e-8.6g). The Routing Manager understandably faces the most pressure since it has to create and distribute routes to all platform components upon the addition of each new E2 Node to the Near-RT RIC. The resource consumption pattern indicates that the E2 Setup flood can amplify the DoS

attack by targeting multiple Near-RT RIC platform services. Interestingly, however, we find no detrimental effects on the application latency for E2 interface control signaling between the subscribed xApps and E2 Nodes (Figures 8.6h-8.6j). This is attributable to the nature of the SCTP connection in ensuring individual transport network streams for facilitating communications for established E2 connections. Also, the attack primarily targets the memory consumption of the E2 Terminator rather than CPU usage as witnessed during the signaling storm attack (Section 8.4e).

At around the eighth minute, and within two minutes into the DoS attack, the E2 Terminator becomes unresponsive and unable to maintain existing as well as new E2 connections. Connecting a little more than 1000 E2 Nodes to the E2 Terminator disrupts the RMR communications internal to the Near-RT RIC and the liveness and readiness probes for the pod fail. Since Kubernetes enforces self-healing to kill and restart unhealthy pods, all existing E2 signaling facilitated by the E2 Terminator is reset. Thus, any E2 Node controlled by one or more xApps on the Near-RT RIC gets disconnected, thereby impairing RAN optimization and control on all affected E2 Nodes.

Given the intensity of this resource exhaustion attack, an xApp-based prevention strategy cannot provide a fool-proof solution to alleviate such attacks from compromising the availability of the Near-RT RIC. The detection strategy should rather be localized and integrated within the Near-RT RIC platform. In this regard, a proof-of-concept strategy to optimize the placing of the E2 Terminator to reduce control plane signaling latency is proposed in a related work [31]. We suggest a few potential approaches to tackle this problem along with conclusions for this thesis work in Chapter 9.

Chapter 9

Conclusion and Future Work

This chapter summarizes the contributions of this thesis and recapitulates the results obtained from this experimental research, along with scope for future work.

9.1 Summary of Contributions and Results

This thesis investigated the scalability and security posture of the Near-RT RIC platform and its components, focusing on the E2 interface. We demonstrated the limitation in the design of the OSC Near-RT RIC to support multiple xApps to control the RAN simultaneously (**contribution 1**). We also traced this limitation to the lack of clarity in one of the IEs defined in the E2AP specifications by O-RAN WG3 (**contribution 2**). We implemented an improved message routing mechanism that enables simultaneous xApp control (**contribution 3**). We developed an xApp to monitor the application layer latency over the E2 interface (**contribution 4**). We designed a multi-threaded E2 Node simulator for orchestrating resource exhaustion DoS attacks and signaling storm attacks on the Near-RT RIC over the southbound E2 interface (**contribution 5**). Finally, we experimentally assessed the security posture of the OSC Near-RT RIC by characterizing the performance of the platform components and the signaling latency between the E2 Terminator and one or more E2 Nodes (**contribution 6**).

Our results demonstrate that the E2 Terminator application can be impacted through sig-

naling storms to violate the control loop latency causing undesirable effects on the E2 Node. Additionally, we showed that an xApp-based latency monitoring solution can be effective in detecting signaling storms at the Near-RT RIC. We also demonstrated a high-intensity spillover DoS attack exploiting the E2 Setup procedure to starve multiple Near-RT RIC components of physical computing resources like CPU and memory. The control signaling results indicate that the detection of severe resource exhaustion attacks cannot be accomplished using xApps and that detection and mitigation strategies need to be put in place within the platform to quickly identify such attacks and prevent service outage.

9.2 Future Work

This thesis mainly aimed to establish the possibility of DoS attacks and characterize their effects on the Near-RT RIC platform components. While the Latency Monitoring xApp is a decent way to observe the attack, further work is required to reduce false positives and enable dynamic latency threshold triggers for flagging a potential signaling storm. For example, AI/ML solutions can be incorporated into the xApp logic to adapt the trigger threshold based on the historical time-series latency data available from the xApp. Also, more research is required to prevent resource exhaustion through E2 Setup attacks. Currently, the E2 Terminator is not state-aware and blindly forwards incoming requests to the E2 Manager. Similarly, the E2 Manager does not check the connection load of the requesting E2 Terminator before accepting an incoming setup request. Enabling such features will increase the resilience and load balancing capability of the Near-RT RIC and thwart DoS attacks from malicious RAN elements.

Bibliography

[1] O-RAN Alliance, "O-RAN About us." https://www.o-ran.org/about, 2022. Last Accessed: 08/23/2022.

[2] O-RAN Working Group 1, "O-RAN Architecture Description," Technical Specification (TS) O-RAN.WG1.O-RAN-Architecture-Description-v06.00, O-RAN Alliance, 3 2022. Last Accessed: 07/31/2022.

[3] Commonwealth Cyber Initiative, "Commonwealth Cyber Initiative/ CCI xG Testbed." https://cyberinitiative.org/xg-testbed.html, 2023. Last Accessed: 02/28/2023.

[4] O-RAN Working Group 3, "Near-Real-time RAN Intelligent Controller, E2 Application Protocol (E2AP)," Technical Specification (TS) O-RAN.WG3.E2AP-v02.02, O-RAN Alliance, 7 2022. Last Accessed: 07/31/2022.

[5] X. Foukas, N. Nikaein, M. M. Kassem, M. K. Marina, and K. Kontovasilis, "FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, (New York, NY, USA), p. 427–441, Association for Computing Machinery, 2016.

[6] R. Schmidt, M. Irazabal, and N. Nikaein, "FlexRIC: An SDK for next-Generation SD-RANs," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '21, (New York, NY, USA), p. 411–425, Association for Computing Machinery, 2021.

[7] E. Coronado, S. N. Khan, and R. Riggio, "5G-EmPOWER: A Software-Defined Net-

working Platform for 5G Radio Access Networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 715–728, 2019.

[8] Open Networking Foundation, "SD-RAN." https://opennetworking.org/open-ran/, 2023. Last Accessed: 02/28/2023.

[9] T. Karamplias, S. T. Spantideas, A. E. Giannopoulos, P. Gkonis, N. Kapsalis, and P. Trakadas, "Towards Closed-loop Automation in 5G Open RAN: Coupling an Open-Source Simulator with xApps," in *2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pp. 232–237, 2022.

[10] M. Dryjański, Ł. Kułacz, and A. Kliks, "Toward modular and flexible open RAN implementations in 6G networks: Traffic steering use case and O-RAN xapps," *Sensors (Basel)*, vol. 21, p. 8173, Dec. 2021.

[11] D. Johnson, D. Maas, and J. Van Der Merwe, "NexRAN: Closed-Loop RAN Slicing in POWDER -A Top-to-Bottom Open-Source Open-RAN Use Case," in *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental Evaluation & CHaracterization*, WiNTECH'21, (New York, NY, USA), p. 17–23, Association for Computing Machinery, 2022.

[12] A. Huff, M. Hiltunen, and E. P. Duarte, "Rft: Scalable and fault-tolerant microservices for the o-ran control plane," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 402–409, 2021.

[13] O-RAN Security Focus Group, "Study on Security for Near Real Time RIC and xApps," Technical Report (TR) O-RAN.SFG.Security-for-Near-RT-RIC–xApps-v01.00, O-RAN Alliance, 7 2022. Last Accessed: 07/31/2022.

[14] S. Poretsky and J. Jardal, "Security Considerations of Cloud RAN," tech. rep., Ericsson, 2021. Last Accessed: 07/31/2022.

[15] 5G Americas, "Security for 5G," tech. rep., 5G Americas, 2021. Last Accessed: 07/31/2022.

[16] Altiostar, "Security in Open RAN," tech. rep., Altiostar, 2021. Last Accessed: 07/31/2022.

[17] Mavenir, "SECURITY IN OPEN RAN," tech. rep., Mavenir, January 2021. Last Accessed: 07/31/2022.

[18] Deutsche Telekom, Orange, Telefónica, TIM, and Vodafone, "Open RAN Security Whitepaper," tech. rep., Telecom Infra Project (TIP), 2022. Last Accessed: 07/31/2022.

[19] Cybersecurity and Infrastructure Security Agency (CISA), and National Security Agency (NSA), "Open RAN Security Whitepaper," tech. rep., CISA and NSA, 2022. Last Accessed: 07/31/2022.

[20] O-RAN Working Group 3, "Near-Real-time RAN Intelligent Controller: Near-RT RIC Architecture," Technical Specification (TS) O-RAN.WG3.RICARCH-v03.00, O-RAN Alliance, 10 2022. Last Accessed: 02/28/2023.

[21] Scott Daniels, "gRPC Evaluation." https://wiki.o-ran-sc.org/display/RICP/gRPC+Evaluation, 9 2020. Last Accessed: 02/27/2023.

[22] O-RAN Working Group 3, "Near-Real-time RAN Intelligent Controller Architecture & E2 General Aspects and Principles," Technical Specification (TS) O-RAN.WG3.E2GAP-v02.02, O-RAN Alliance, 7 2022. Last Accessed: 07/31/2022.

[23] O-RAN Working Group 3, "Near-Real-time RAN Intelligent Controller E2 Service Model (E2SM," Technical Specification (TS) O-RAN.WG3.E2SM-v02.01, O-RAN Alliance, 3 2022. Last Accessed: 07/31/2022.

[24] A. Ksentini and N. Nikaein, "Toward Enforcing Network Slicing on RAN: Flexibility and Resources Abstraction," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, 2017.

[25] Software Radio Systems, "srsRAN 4G." https://www.srslte.com/4g, 2023. Last Accessed: 02/28/2023.

[26] IEEE INFOCOM, "IEEE INFOCOM 2023 Posters and Demos." https://infocom2023.ieee-infocom.org/postersdemos, 2023. Last Accessed: 04/15/2023.

[27] O-RAN Security Focus Group, "Security Protocols Specifications," Technical Specification (TS) O-RAN.SFG.Security-Protocols-Specifications-v03.00, O-RAN Alliance, 11 2021. Last Accessed: 07/31/2022.

[28] O-RAN Security Focus Group, "O-RAN Security Requirements Specificaions," Technical Specification (TS) O-RAN.SFG.Security-Requirements-Specifications-v03.00, O-RAN Alliance, 3 2022. Last Accessed: 07/31/2022.

[29] N. Tripathi and N. Hubballi, "Application Layer Denial-of-Service Attacks and Defense Mechanisms: A Survey," *ACM Comput. Surv.*, vol. 54, May 2021.

[30] ZeroMQ, "ZeroMQ: An open-source universal messaging library." https://zeromq.org/, 2023. Last Accessed: 04/15/2023.

[31] G. M. de Almeida, G. Z. Bruno, A. Huff, M. A. Hiltunen, E. P. Duarte, C. B. Both, and K. V. Cardoso, "RIC-O: Efficient placement of a disaggregated and distributed RAN In-

telligent Controller with dynamic clustering of radio nodes," *ArXiv*, vol. abs/2301.02760, 2023.