https://doi.org/10.7840/kics.2023.48.1.85

# Riverbed(OPNET) Modeler를 이용한 TDMA 기반 MANET 구현

임 진 택•, 김 수 진*, 유 지 상*, 강 홍 구°

# Implementation of a Simulator for TDMA-Based MANET Using Riverbed(OPNET) Modeler

Jin-Taek Lim•, Sujin Kim*, Jisang You*, Hongku Kang°

요 약

패킷 단위의 이산 사건 기반의 시뮬레이터인 Riverbed(OPNET) Modeler는 복잡한 통신 프로토콜과 대규모의 네트워크를 모의하기 용이하여, 금융, 국방, 사물인터넷 등 다양한 분야에서 모델링 및 성능분석에 널리 활용되고 있다. 하지만, Riverbed Modeler에서 제공하는 기본 MANET(Mobile Ad-Hoc Network) 프레임워크를 활용하여 새로운 라우팅 알고리즘을 구현하기에는 해당 프레임워크에 대한 심도 깊은 이해와 방대한 양의 수정을 요구한다. 이러한 Riverbed Modeler의 라우팅 알고리즘 추가에 대한 어려움을 해소하기 위하여, 본 논문에서는 라우팅 알고리즘을 모의하기 위한 핵심 모듈만을 자체 개발하여 TDMA(Time Division Multiple Access) 기반의 MANET 프레임워크를 제안한다. 제안하는 라우팅 프로토콜 프레임워크는 IP 프로토콜 등 타프로세서와 연관되는 모듈이 없으며, 기본적인 패킷의 지연시간 및 패킷 손실률을 보기 위해 설계/개발하였기 때문에 새로운 라우팅 프로토콜을 최소한의 수정으로 추가하여 모의할 수 있도록 구성되었다. 라우팅 프로토콜 프레임워크를 이용하여 라우팅 프로토콜을 수정하는 경우, 수정해야 하는 부분은 라우팅 알고리즘과 일부 패킷의 헤더 정보뿐이기 때문에, 보다 수정이 쉽고 오류의 가능성을 줄일 수 있다. 제안한 라우팅 프로토콜 프레임워크를 이용하여 다익스트라 알고리즘을 이용한 나무형 MANET의 예제의 패킷의 지연시간 및 패킷 손실률의 측정을 확인하였다.

**키워드 :** 시분할 다중접속, 이동형 에드혹 네트워크, 네트워크 시뮬레이터, 리버베드, 옵넷
**Key Words :** TDMA, MANET, Network Simulator, Riverbed, OPNET

ABSTRACT

Riverbed(OPNET) Modeler is a packet-level discrete event-driven simulator that can easily simulate complex communication protocols and large-scale networks. However, it requires a deep understanding and extensive modifications to implement a new routing algorithm using the built-in MANET(Mobile Ad-Hoc Network) framework provided by Riverbed Modeler. In this paper, we propose a TDMA(Time Division Multiple Access)-based MANET framework that can make it easier and reduce the possibility of errors in order to solve the difficulty of adding routing algorithm to Riverbed Modeler. The proposed TDMA-based MANET framework has no modules related to other basic processors, such as IP protocol, and was developed in-house

※ 본 연구는 2022년 정부의 재원으로 수행된 연구입니다.
• First Author : Agency for Defense Development, jtlim870708@gmail.com, 정회원
° Corresponding Author : Agency for Defense Development, help_u@naver.com, 정회원
* Agency for Defense Development
논문번호 : 202208-170-D-RN, Received August 5, 2022; Revised October 11, 2022; Accepted October 20, 2022

to measure the basic packet latency time and packet loss rate. When the routing algorithm is modified using the proposed TDMA-based MANET framework, only the routing algorithm and header information of some packets need to be modified. Using this proposed framework, an example of a tree-type MANET using Dijkstra's algorithm was presented to confirm the measurement of packet latency time and packet loss rate.

## Ⅰ. 서 론

네트워크 시뮬레이터는 크게 오픈소스 기반의 NS-3[1]와 상용 소프트웨어인 Riverbed Modeler(舊 OPNET Modeler)[2]가 널리 이용된다. NS-3와 Riverbed Modeler 모두 패킷 수준의 이산 사건 시뮬레이터(Discrete Event Simulator)이기 때문에 복잡한 통신 프로토콜과 대규모의 네트워크를 설계하기에 적합하다. 그 중 Riverbed Modeler의 경우, 실제 상용 제품의 상당수가 Model 라이브러리로 제공이 되기 때문에 실환경을 모의하기에 보다 용이한 특징을 가지고 있다. 또한, Riverbed Modeler는 네트워크 각 계층의 프로토콜 및 각종 동작을 스테이트 다이어그램(state diagram)을 기반으로 직관적으로 모델링 할 수 있기 때문에, 네트워크 시뮬레이터로서 신뢰성을 인정받고 있다. 따라서, 우리나라뿐만 아니라 미국 등 여러 나라의 금융 네트워크[3], 사물인터넷[4], 국방 무기체계[5] 통신 분야의 모델링, 네트워크의 설계 및 성능분석에 광범위하게 활용되고 있다.

Riverbed Modeler에서의 MANET(Mobile Ad-Hoc Network) 라우팅 프로토콜은 IP 계층 프로세서의 한 종류인 manet_mgr 프로세서에 의해 관리되어 구현된다. 따라서, 라우팅 프로토콜의 인식 및 처리하는 부분은 IP 계층 프로세서 및 IP 관련 외부 소스 파일과 깊게 얽혀 있다. 이러한 조건 하에서 새로운 라우팅 프로토콜을 구현하여 추가하거나 기존의 라우팅 프로토콜을 수정하는 작업은 관련된 모든 부분을 식별하여 새 프로토콜에 맞게 추가 및 변경해 주어야 한다. 이러한 작업은 Riverbed Modeler 사용에 익숙하지 않은 사용자로 하여금 누락이나 잘못된 수정으로 인한 전체 라우팅 프로토콜의 이상 동작을 초래할 수 있다. 해당 문제점을 극복하기 위하여, 기존의 라우팅 프로세서를 기반으로 최소한으로 내부 동작을 변형하고 차이점을 추가하는 방법들이 제안되었다.[6-8] 하지만, 이 방법들 역시 기존의 라우팅 프로세서를 활용하기 때문에 사용자가 프로세서의 상당 부분을 이해하여야 이용이 가능한 실정이다.

본 논문에서는 이러한 라우팅 프로토콜 추가의 실무적 어려움을 제한적이나마 TDMA(Time Division Multiple Access) 기반 하에서 줄일 수 있는 MANET 라우팅 프로토콜 프레임을 제공한다. 제공하는 TDMA 기반의 MANET 라우팅 프로토콜 프레임워크는 TDMA 기반에 한정되어 있지만 Riverbed Modeler의 복잡한 IP 계층의 프로세서 및 외부 소스 파일을 완전히 배제하고 순수 제안하는 라우팅 프로토콜만을 이산 사건 시뮬레이팅하기 위한 최적의 환경을 뒷받침한다. 제안하는 라우팅 프로토콜이 IP 계층을 통해 타 네트워크와 연결되어 이에 따른 성능분석이 필요한 것이 아니라면 Riverbed Modeler의 복잡한 IP 계층 프로세서 및 외부 소스 파일을 이용할 필요는 없다. 제공하는 라우팅 프로토콜 프레임워크는 응용 패킷이 생성/파괴 되는 상위 단부터 최종 송/수신이 이루어지는 PHY 계층까지 구현이 되어 있고 MAC 계층의 최소한의 추가/수정과 필요한 패킷 구조의 수정으로 새로운 라우팅 프로토콜을 모의해볼 수 있도록 지원하며, 기존 Riverbed 모듈을 활용한 라우팅 프로토콜의 추가/수정 과정의 실수에서 발생할 수 있는 오류를 줄여 주는데 큰 역할을 할 수 있다.

논문의 구성은 다음과 같다. 2장에서는 구현된 라우팅 프로토콜 프레임워크와 시뮬레이터의 특성에 대하여 살펴보고, 3장에서는 제안하는 프레임워크와 Riverbed 내 내장된 프레임워크의 차이점을 분석한다. 4장에서는 제안하는 프레임워크를 실제 예제를 통하여 검증해 본다. 마지막 5장에서 본 논문의 결론을 맺는다.

## Ⅱ. 시뮬레이터 구성

### 2.1 통신 프레임 구조

제안하는 라우팅 프로토콜 프레임은 TDMA기반의 MANET을 구현하기 위해 제어 슬롯과 데이터 슬롯으로 구분되는 시간 프레임 구조를 제공한다. 프레임의 길이는 망이 적용될 환경(채널 변화 주기, 최대 홉 수 등)을 반영하여 결정할 수 있다. 결정된 프레임 길이에서 선두는 제어 슬롯을 위해 고정되며 노드의 총 개수만큼 할당된다. N개의 노드가 존재하면 프레임의 선두에는 N개의 제어 슬롯이 할당된다(그림 1). 제어 슬롯의 길이의 결정은 제어 패킷에 담길 정보의 양과

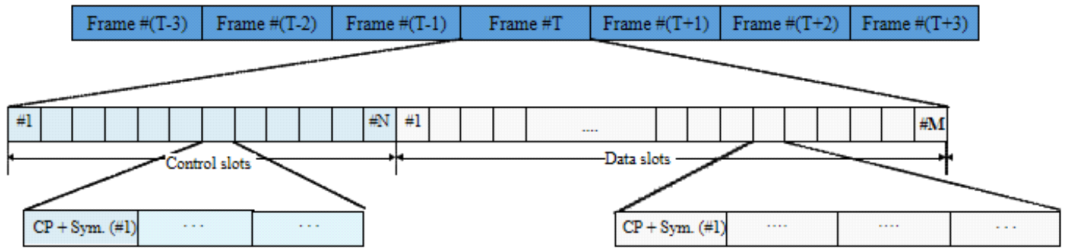


그림 1. 통신 프레임 구조
Fig. 1. Communication frame structure.

PHY 계층의 설계와 관련이 있다. 제어 패킷은 일반적으로 안정적인 전송을 위해 가장 낮은 MCS level로 전송이 된다. 프레임의 나머지 구간은 데이터 패킷을 위해 할당이 되며 총 M개의 데이터 슬롯이 할당될 수 있다. 데이터 슬롯은 실제 데이터 트래픽이 전달되는 슬롯이고 자원할당 정보에 따라 MCS level을 가변하여 이용된다.

제어 패킷에는 망의 토폴로지 정보, 자원할당 정보, 전역 채널품질 정보가 실리게 된다. 망의 토폴로지 정보와 자원할당 정보는 NC(Network Controller) 역할을 수행하는 노드가 계산하여 최초로 제어 패킷에 실어 전파를 하고 그 외의 노드들은 해당 내용을 복사하여 자신의 제어 패킷에 실어 전파한다. 각 노드가 동일한 전역 채널품질 정보를 통해 동일한 망의 토폴로지 및 자원할당 정보를 구할 수 있지만, 망의 동기화 및 계산 부하를 줄이기 위해 NC가 계산하여 전파하는 방식을 취하도록 했다. 전역 채널품질 정보는 각 노드들이 인접한 이웃 노드들의 제어 패킷을 우선 수신하여 이웃 채널품질 정보를 획득한 뒤 이를 제어 패킷의 전역 채널품질 정보의 일부에 갱신하여 연속적으로 전송하는 방식으로 완성된다. 수 프레임이 지나면 NC는 모든 노드들의 이웃 채널 품질이 기록된 전역 채널품질 정보를 획득할 수 있다. 노드들의 이동성이 큰 환경이면, NC가 전역 채널 품질 정보를 획득하는 시간과 현재 채널 품질 정보 사이에 이격이 발생할 수 있다. 따라서, 채널 정보 교환이 빠르게 이루어지기 위해 적절한 프레임 길이의 설계가 필요하다.

### 2.2 노드의 설계

설계한 Riverbed Modeler의 노드 종류는 단일 노드(Node.nd)로 구성된다. 바꿔말하면, 시뮬레이션을 위해 생성되는 모든 노드는 동일한 Node.nd를 통해 생성된다. 노드의 이름을 MSx, BSx 등으로 설정하면 해당 노드의 세 번째부터 오는 x 숫자를 자신의 index로 인식하게 설계하였다. 그리고, 만약 자신의 index가 0이면 NC로 동작하고 나머지는 일반 노드로 동작한다. 노드가 NC로 동작할 경우, 망의 토폴로지와 자원할당을 계산하여 전파하는 역할을 수행한다. 일반 노드는 NC로부터 전파된 망의 토폴로지와 자원할당을 이용하여 데이터를 송/수신한다.

Node.nd의 Processor 구성은 상위부터 APP_Main.ps, Intra_Main.ps, Intra_Control.ps, MAC_Main.ps, MAC_Control.ps, PHY_Main.ps, Transmitter.ps, Receiver.ps, Antenna.ps로 총 9개로 구성된다 (그림 2).

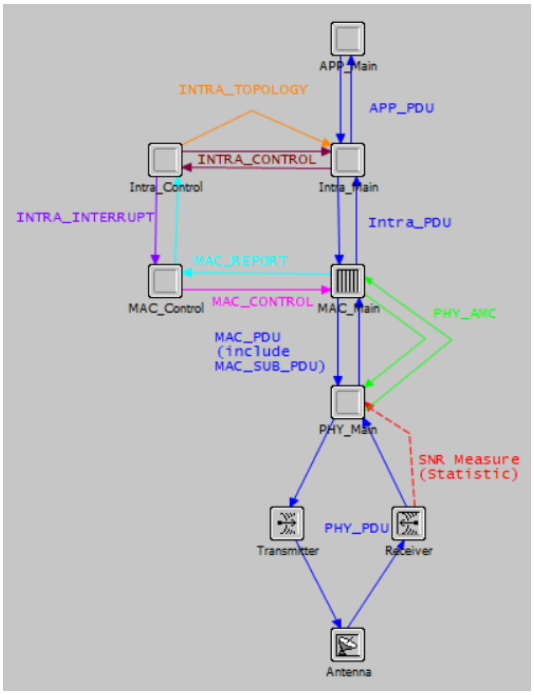


그림 2. Node.nd의 Processor 다이어그램
Fig. 2. Processor diagram of Node.nd

#### 2.2.1 APP_Main Processor

APP_Main.ps는 데이터 패킷의 생성/송신 및 수신/파괴와 통신 성능 분석(패킷의 지연시간, 패킷의 손실률)의 기능을 갖는다. 생성된 데이터 패킷은 APP Header를 붙여 APP_PDU 패킷으로 Intra_Main.ps로 전달된다. APP Header는 AppCode, AppSour, AppDest, quality of service(QoS), sequence number(SN), length로 구성된다. AppCode는 총 2 bit로 구성되며 음성, 전문, 위치, 영상을 구분한다. AppSour와 AppDest은 총 12 bit로 각 위치의 bit는 노드 index를 의미한다. 예를 들어, AppSour=0001 0000 0000은 8번 노드가 송신자라는 의미이고 AppDest=0000 0000 1111은 0, 1, 2, 3번 노드가 수신자라는 뜻이다. QoS는 총 2 bit로 해당 트래픽의 우선 순위를 나타낸다. 그리고, 수신된 데이터 패킷은 총 2 bytes의 SN을 통해 정렬하여 복원한다. length는 헤더 뒤의 payload의 길이를 의미하여 총 2 bytes를 차지한다.

#### 2.2.2 Intra_Main Processor

Intra_main.ps의 경우, 크게 3가지의 역할을 하는 프로세서이다. 첫 번째로 MAC_Main으로부터 전달되는 Intra_PDU 패킷을 분류한다. Intra_PDU를 해석하여 제어 패킷이 수신되었을 경우, 이를 INTRA_CONTROL 내부 패킷으로 변환하여 Intra_Control.ps에 전달한다. 반면에, 데이터 패킷이 수신되었고, App_PDU 헤더의 AppSour, AppDest의 해석을 통해 이것이 본 수신 노드의 데이터 패킷이면 Intra_PDU의 Header를 제거하여 APP_Main.ps로 올려 최종 수신/파괴를 수행하게 한다, 예를 들어, AppSour에 1번 노드가 있고 AppDest에 2번 노드가 있을 때, 수신노드가 2번 이면 최종 수신하는 식이다. 중계해야 할 패킷으로 판단하기 위해서는 Intra PDU 헤더의 TxNode와 RxNode 필드도 같이 보아야 한다. 예를들어, AppSour에 1번 노드가 있고 AppDest에 4번노드가 있을 때 수신노드가 3번이라고 가정하자. 이때, 수신노드가 가진 토폴로지 정보를 통해서 자신의 부모 노드가 Intra_PDU 헤더의 TxNode와 같고, RxNode가 자신이라면, TxNode를 자신의 숫자로 고치고 AppDest 2번에 전달이 가능한 토폴로지 내 다음 중계노드를 RxNode에 기입하여 MAC_Main.ps에 전달하여 다음 중계를 이어가는 방식이다. 둘째, Intra_Control.ps로부터 제어 메시지의 해석결과(토폴로지)가 수신되면 이를 기반으로 중계를 위한 내부 테이블 업데이트를 수행한다. 앞서, 중계를 판단할 때 사용한 토폴로지가 이것이다. 마지막으로, Intra_Control.ps로부터 생성된 제어 패킷을 전달 받을 경우 이를 MAC_Main.ps에 넘기는 역할을 수행한다.

#### 2.2.3 MAC_Main Processor

MAC_Main.ps은 Intra_Main.ps로부터 전달되는 패킷을 각 종류에 맞는 큐에 저장을 한 뒤, 현 송신하려는 채널의 품질에 따라 AMC(Adaptive Modulation and Coding)를 이용하여, MCS Level을 조절하고 이에 맞게 MAC_PDU를 생성하여 PHY_Main.ps에 전달하는 역할을 수행한다. MAC_PDU의 길이는 PHY의 구현에 의존한다. OFDM Symbol의 개수와 carrier 주파수의 개수 등에 의해 한 슬롯 당 보낼 수 있는 데이터량을 구할 수 있다. 예를 들어, Symbol당 50 bytes를 보내도록 설계가 되어 있고, 슬롯당 5개의 symbol이 존재하면 총 250 bytes의 MAC_PDU를 생성할 수 있다. MAC 헤더의 길이 20 bytes를 제외한다면, 총 230 bytes를 실을 수 있고, 이 230 bytes를 채우기 위해, 각 데이터의 큐를 우선순위로 조회한다. 데이터의 우선 순위가 음성>전문>위치>영상이라면 먼저 음성 큐에서 Intra_PDU를 꺼내어 sub MAC header를 붙여 MAC_PDU의 payload에 넣는다. 위의 우선 순위로 payload를 채우다가, 꺼낸 Intra_PDU의 크기가 payload의 남은 공간을 넘어설 경우, fragmentation을 수행한다. sub MAC 헤더의 길이를 제외한 보낼 수 있는 만큼만 잘라서 MAC_PDU를 최종 완성하고 패킷의 나머지 부분은 해당 데이터 큐의 선두에 재삽입된다. fragmentation 된 정보는 sub MAC 헤더에 입력된다. 본 패킷이 fragmentation 된 몇 번째 패킷인지, 혹은 마지막 패킷인지가, frag_ID와 frag_end에 표시되어 수신단에서는 이를 통해 fragmentation 된 패킷들을 모아 본 패킷을 복원할 수 있다. 복원된 패킷은 Intra_Main.ps에 전달된다. MAC_Main.ps가 MAC_PDU 패킷을 PHY_Main.ps에 전달하는 시점은 MAC_Control.ps의 MAC_Control 패킷을 통해서 이뤄진다. MAC_Control.ps는 제어 패킷을 통해 공유된 자원할당 정보를 토대로 현 프레임 내 데이터 슬롯의 송신 노드가 무엇인지, MCS 레벨이 어떤 것인지 판별할 수 있고, 본 노드가 송신 노드라면 MAC_PDU 패킷을 생성하여 PHY_Main.ps에 전달하여 송신한다. 반면, 자신이 송신 노드가 아니라면 PHY_Main에 MCS 레벨만을 전달하여 PHY_Main의 MCS 레벨 조절을 통해 타 노드로부터 전송되는 패킷을 수신할 수 있도록 한다. 이외에도, PHY_Main.ps으로부터 PHY_AMC 패킷으로 보고되는, 본 노드를 수신점으로 하는 각 노드들과의 채널 품질 (SNR dB

값)을 MAC_Control.ps을 통해 MAC_Report의 형태로 Intra_Control.ps로 전달한다. Intra_Control.ps는 측정된 채널 품질 정보를 토대로 제어 패킷의 전역 채널 품질 정보를 완성한다.

#### 2.2.4 PHY_Main Processor

PHY_Main.ps는 송신할 MAC_PDU를 PHY_PDU로 변환하여 Transmitter.ps를 통해 송신하거나 PHY_PDU를 수신하여 MAC_PDU로 전환하여 MAC_Main.ps로 전달한다. 또한, Receiver.ps로 부터오는 SNR 측정 통계 정보를 토대로 채널의 품질을 PHY_AMC 패킷으로 전달한다.

#### 2.2.5 Intra_Control Processor

Intra_Control.ps는 제어 패킷의 생성을 담당한다. 제어 패킷에는 토폴로지 정보와 자원할당 정보가 담기는데 NC가 이를 계산하여 방송하기 위해서는, 각 노드가 하위단으로부터 전달받은 MAC_Report 패킷의 채널 품질 정보를 이용한다. MAC_Report 패킷을 통해 본 노드의 주변 노드들과의 채널 품질 정보를 파악하고, 타 노드로부터 전송된 제어 패킷으로부터 그 외 노드들 기준의 채널 품질 정보를 파악할 수 있다. 제어 패킷의 전역 채널 품질 정보에, 본 노드의 최근 채널 품질 정보만을 업데이트하여 업데이트된 채널의 SN만을 증가시켜 재전송하면, 이러한 절차가 각 노드에서 반복되어 수행되고 난 뒤 NC 노드는 전역 채널 품질 정보를 획득할 수 있게 된다. 이를 통해, 토폴로지와 자원할당을 계산하는 방법은 다양하다. 본 논문에서는 토폴로지와 자원할당의 방법을 제안하지는 않는다. 계산된 토폴로지는 각 노드의 부모 노드가 무엇인지를 제어 패킷에 기입하여 전달된다. 자원할당 정보는 앞서 언급했듯이 각 데이터 슬롯의 송신 노드가 무엇인지, 그리고 해당 슬롯의 송신 MCS 레벨은 어떤 것 인지를 테이블의 형태로 만들어 제어 패킷을 통해 전달한다.

#### 2.2.6 MAC_Control Processor

NC가 아닌 노드들은 제어 패킷을 통해 토폴로지와 자원할당 정보를 전달받고 이를 MAC_Control.ps로 인터럽트 테이블을 전달한다 (INTRA_INTERRUPT 패킷). MAC_Control.ps는 해당 테이블을 토대로 프레임 내 데이터 슬롯의 타이밍(인터럽트)가 오면 MAC_Main에 MAC_Control 패킷을 보내어 해당 타이밍에 송/수신을 가능토록 한다.

#### 2.2.7 Transmitter/Receiver/Antenna Processor

Transmitter.ps와 Receiver.ps는 송/수신기 관련 특징을 품은 processor로써, noise figure channel(data rate, bandwidth, frequency, processing gain), modulation, margin 등 Riverbed가 지원하는 일반적인 송수신 변수들을 설정할 수 있다.

Antenna.ps는 Riverbed가 지원하는 기본 processor로써 다양한 안테나 패턴과 오리엔테이션을 설정하는 processor이다.

#### 2.2.8 Header Files

전체 시뮬레이션의 변수와 결과를 관리하기 위해 헤더 파일들을 두고 이를 각 프로세서의 모듈에서 include하는 것이 편리하다. 본 프레임워크에서는 전체 시뮬레이터의 시스템 파라미터를 관장하는 Environments.h을 두어, 디버깅을 위한 모니터링용 LOG를 on/off할 수 있는 파라미터를 설정할 수 있도록 하고, 전체 노드의 개수 및 다중 트래픽의 모델링, 프레임의 길이 및 심볼 길이, 헤더 사이즈, MCS 비트 심볼사이즈 등을 설정할 수 있도록 하였다. 또한 각 트래픽의 송신 및 수신 노드 설정도 가능하도록 하였다. 나머지 하나인, Variable.h에서는 결과 도시와 관련된 전역 변수들을 선언하였다.

## Ⅲ. Riverbed 내장 MANET 프레임워크와의 비교

제안하는 MANET 프레임워크와 Riverbed 내 기존 MANET 프레임워크의 차이를 비교하여 표 1에 나타내었다. Riverbed 내에서 제공하는 MANET 프레임워크는 총 16개의 프로세서들로 구성이 되고 전체 코드의 길이를 합산하면 약 20만 줄에 육박한다. 이중 IP 프로토콜과 관련된 프로세서들이 약 15만 줄로 큰 비율을 차지하며, 해당 프레임워크의 모든 연결관계를 모두 파악하여 추가/수정하기 어려운 실정이다. 개발한 라우팅 알고리즘을 구현하여 테스트를 할 때, 타 상위 장치들과의 IP 프로토콜 기반 데이터 교환을 가정하지 않는다면, IP 프로토콜 관련 프로세서들은 고려하지 않아도 된다. 제안하는 프레임워크는 사용자 설계 트래픽 모드, TDMA 기반 자원 할당 구조, 제어 패킷, 데이터 패킷, AMC 등을 모두 지원할 수 있고 상대적으로 구현에 필요한 코드의 수가 짧아 개발한 라우팅 알고리즘을 추가/수정하여 시뮬레이션하기 보다 용이하다.

표 1. 제안 MANET 프레임워크와 기존 MANET 프레임워크의 비교
Table 1. Comparison between a proposed MANET framework and a built-in MANET framework

| | | 기존 MANET 프레임워크 | | 제안 MANET 프레임워크 | |
|---|---|---|---|---|---|
| 구성 프로세서의 수 (개) | | 16 | | 9 | |
| 사용자 설계 트래픽 모델 | | 지원 | | 지원 | |
| TDMA 기반 자원 할당 | | 지원 | | 지원 | |
| 제어 패킷 | | 지원 | | 지원 | |
| 데이터 패킷 | | 지원 | | 지원 | |
| AMC | | 지원 | | 지원 | |
| IP 프로토콜 호환 | | 지원 | | 미지원 | |
| 총 코드 규모 (줄) | IP 프로토콜 관련 | 약 20만 | 약 15만 | 약 1만 | - |
| | IP 프로토콜 비관련 | | 약 5만 | | 약 1만 |

## Ⅳ. 시뮬레이터 수행 예제 결과

제안하는 MANET 프레임워크의 정상 동작을 확인하기 위하여 예제를 산정하여 시뮬레이션을 수행하였다. 시뮬레이션을 위해 총 12개의 노드를 그림 3과 같이 배치하였다. 해당 배치에서 0번 노드와 1번 노드, 1번 노드와 2 번 노드의 간격은 1 km로 하였고, 2번 노드와 3~11번 노드의 간격은 1 km에 180도 범위 내에서 3~11번 노드를 상호 등간격으로 설정하였다. 각 노드의 전송 전력은 27 dBm이고 채널의 경로 손실 모델로는 Okumura-Hata suburban 모델을 사용하였다. 신호의 중심 주파수는 2.4 Ghz이고 대역폭 4 MHz로 송수신하며, 제어 슬롯 하나당 총 10개의 OFDM 심볼, 데이터 슬롯 하나당 총 12개의 OFDM 심볼을 배치하였다. 심볼 하나당 데이터량은 표 2과 같고 프레임의 자세한 설계는 표 3에 요약하였다. 해당 값들은 Environments.h 에 저장한다. MCS 레벨에는 QPSK 1/2, 16QAM 1/2, 64QAM 1/2, 64QAM 2/3, 64QAM 5/6의 SNR vs BER 커브가 적용되었다.

데이터 트래픽으로는 음성, 전문, 위치, 영상 총 4가지를 산정하였으며, 음성은 25 ms를 주기로 40 Bytes 길의 패킷이, 위치은 15 bytes씩 1 s 마다 생성이 되도록 하였다. 영상은 33 ms 마다 1000~1500 bytes씩 균등한 확률 분포로 패킷을 발생시켰다. 전문의 경우는, 평균을 60 s로 갖는 exponential 랜덤변수로 발생 주기를 설정하였고, 한 주기 간격당 500 bytes의 패킷이 발생하도록 하였다. 그밖에, tree 구조의 라우팅 알고리즘 형성으로는 다익스트라 알고리즘을 사용하였다. 각 데이터 슬롯 별 송신 노드와 트래픽 종류의 할당 방법은 예제의 편의를 위해 임의 선택되어지도록 하였다. 사용자는 해당 함수를 수정하여, 제안하는 라우팅 및 자원할당 알고리즘을 모의해볼 수 있으며, 필요하다면 제공하는 패킷의 헤더를 수정하여 함수에 필요한 정보를 확장할 수 있다.

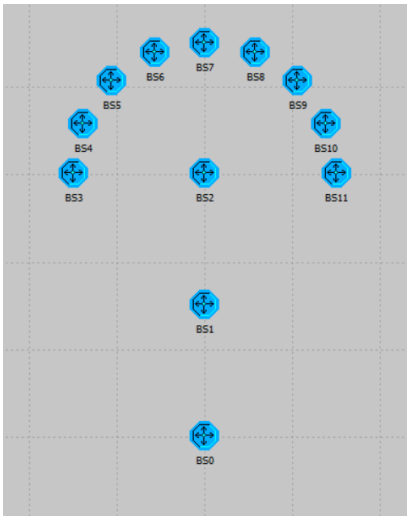


그림 3. (그림 4)의 Topology List가 생성된 실제 토폴로지
Fig. 3. Topology based on topology list of Fig. 4

표 2. MCS 레벨 별 OFDM 심볼당 데이터량
Table 2. MCS level v.s. data size per OFDM symbol

| | |
|---|---|
| MCS 1 | 70 bytes |
| MCS 2 | 140 bytes |
| MCS 3 | 210 bytes |
| MCS 4 | 280 bytes |
| MCS 5 | 350 bytes |

표 3. 프레임 구조
Table 3. Frame structure

| | |
|---|---|
| 프레임 길이 | 100 ms |
| 프레임 뮤트 길이 | 0.544 ms |
| 제어 슬롯 길이 | 1.112 ms |
| 데이터 슬롯 길이 | 1.104 ms |
| 제어 슬롯 수 | 12 개 |
| 데이터 슬롯 수 | 78 개 |

```
***** Node 0 : ( F400)  Information Display @ 30.000000 *****
----- NC Rx SNR Table For Calculating Next1 Topology -----
       |   Rx   0  |   Rx   1  |   Rx   2  |   Rx   3  |   Rx   4  |   Rx   5  |   Rx   6  |   Rx   7  |   Rx   8  |   Rx   9  |   Rx  10  |   Rx  11  |
-------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------
Tx  0  |  0.000000   6.481925   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
Tx  1  |  6.481925   0.000000   6.481925   0.437845   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.437845
Tx  2  |  0.000000   6.481925   0.000000   6.481925   6.490839   6.492782   6.490839   6.481925   6.490839   6.492782   6.490839   6.481925
Tx  3  |  0.000000   0.437845   6.481925   0.000000  22.891011  11.150677   4.651009   0.437845   0.000000   0.000000   0.000000   0.000000
Tx  4  |  0.000000   0.000000   6.490839  22.891011   0.000000  22.913431  11.162323   4.651009   0.446760   0.000000   0.000000   0.000000
Tx  5  |  0.000000   0.000000   6.492782  11.150677  22.913431   0.000000  22.913431  11.150677   4.651386   0.448703   0.000000   0.000000
Tx  6  |  0.000000   0.000000   6.490839   4.651009  11.162323  22.913431   0.000000  22.891011  11.146018   4.651386   0.446760   0.000000
Tx  7  |  0.000000   0.000000   6.481925   0.437845   4.651009  11.150677  22.891011   0.000000  22.891011  11.150677   4.651009   0.437845
Tx  8  |  0.000000   0.000000   6.490839   0.000000   0.446760   4.651386  11.146018  22.891011   0.000000  22.913431  11.162323   4.651009
Tx  9  |  0.000000   0.000000   6.492782   0.000000   0.000000   0.448703   4.651386  11.150677  22.913431   0.000000  22.913431  11.150677
Tx 10  |  0.000000   0.000000   6.490839   0.000000   0.000000   0.000000   0.446760   4.651009  11.162323  22.913431   0.000000  22.891011
Tx 11  |  0.000000   0.437845   6.481925   0.000000   0.000000   0.000000   0.000000   0.437845   4.651009  11.150677  22.891011   0.000000
```

그림 4. Simulation Console 내 전역 채널 품질 정보 디스플레이 (노드 0)
Fig. 4. Global channel state information display (Node 0) in the simulation console

시뮬레이션 완료 후 결과는 Text로 확인할 수 있도록 하였으며, Riverbed 내의 시뮬레이션 콘솔창을 통해서도 이를 확인할 수 있다. Text 파일 또한 생성되며 결과 파일이 생성되는 경로와 파일 이름은 APP_Main.ps 내의 Show_Result state 내에서 수정이 가능하도록 하였다.

그림 4에 노드 0이 지니고 있는 전역 채널 품질 정보를 표시하였다. 세로 행은 송신 노드 가로 열은 수신 노드를 나타낸다. 총 12 x 12로 각 링크간 채널의 품질을 SNR dB scale로 표기한다. 다만, 여기서 dB값이 음수값(-)가 나오지 않는 이유는 채널 품질을 측정하는 방식 때문이다. Riverbed 내 Receiver.ps는 공중을 통해 전송되는 패킷을 패킷으로 인식하기 위한 최소 SNR 크기를 요구한다. 이 값이 현재 0이 되도록 설정되었기 때문에 실제 SNR이 음수값인 패킷도 Receiver.ps에 입력은 되나 무시가 되기 때문에, 채널 측정 값으로 PHY_Main.ps로 전송이 되지않고 전역 채널 품질 정보 내 최소 값이 0으로 표시되는 것이다.

```
 ----- Topology List -----
Present :  100  0  1  2  2  2  2  2  2  2  2
 ----- Frame Allocation : PRESENT -----

0 Slot (29.925000):  0 Tx node (Beacon, MCS 1)
1 Slot (29.926312):  1 Tx node (Beacon, MCS 1)
2 Slot (29.927624):  2 Tx node (Beacon, MCS 1)
3 Slot (29.928936):  3 Tx node (Beacon, MCS 1)
4 Slot (29.930248):  4 Tx node (Beacon, MCS 1)
5 Slot (29.931560):  5 Tx node (Beacon, MCS 1)
6 Slot (29.932872):  6 Tx node (Beacon, MCS 1)
7 Slot (29.934184):  7 Tx node (Beacon, MCS 1)
8 Slot (29.935496):  8 Tx node (Beacon, MCS 1)
9 Slot (29.936808):  9 Tx node (Beacon, MCS 1)
10 Slot (29.938120):  10 Tx node (Beacon, MCS 1)
11 Slot (29.939432):  11 Tx node (Beacon, MCS 1)
12 Slot (29.940744):  3 Tx node (Voice, MCS 1)
13 Slot (29.942048):  4 Tx node (Voice, MCS 1)
14 Slot (29.943352):  5 Tx node (Voice, MCS 1)
15 Slot (29.944656):  6 Tx node (Voice, MCS 1)
16 Slot (29.945960):  7 Tx node (Voice, MCS 1)
17 Slot (29.947264):  8 Tx node (Voice, MCS 1)
18 Slot (29.948568):  9 Tx node (Voice, MCS 1)
19 Slot (29.949872):  10 Tx node (Voice, MCS 1)
20 Slot (29.951176):  11 Tx node (Voice, MCS 1)
21 Slot (29.952480):  9 Tx node (Video, MCS 1)
22 Slot (29.953784):  2 Tx node (Video, MCS 1)
23 Slot (29.955088):  1 Tx node (Video, MCS 1)
24 Slot (29.956392):  9 Tx node (Video, MCS 1)
25 Slot (29.957696):  2 Tx node (Voice, MCS 1)
26 Slot (29.959000):  2 Tx node (Video, MCS 1)
27 Slot (29.960304):  1 Tx node (Video, MCS 1)
28 Slot (29.961608):  9 Tx node (Video, MCS 1)
29 Slot (29.962912):  2 Tx node (Video, MCS 1)
30 Slot (29.964216):  1 Tx node (Voice, MCS 1)
31 Slot (29.965520):  1 Tx node (Video, MCS 1)
32 Slot (29.966824):  9 Tx node (Video, MCS 1)
33 Slot (29.968128):  2 Tx node (Video, MCS 1)
34 Slot (29.969432):  1 Tx node (Video, MCS 1)
```

그림 5. 시뮬레이션 콘솔 내 토폴로지와 자원할당 정보 (노드 0)
Fig. 5. Topology and resource allocation information display (Node 0) in the simulation console

그림 5에는 노드 0이 가지고 있는 토폴로지와 자원할당 정보의 결과가 표시되어있다. Topology List 하단의 12개의 숫자는 각각 노드 0~11의 부모 노드의 숫자를 의미한다. 여기서 숫자 100은 부모 노드가 존재하지 않고 해당 노드가 본 토폴로지의 Root(NC)라는 뜻으로 사용된다. 다익스트라 알고리즘을 통해, 그림 5의 채널 하에서 현재 노드 0은 NC, 노드 1은 노드 0에, 노드 2은 노드 1에, 나머지 노드들은 노드 2에 연결된 토폴로가 형성된 것을 확인할 수 있다 (그림 3). Frame Allocation(자원할당 정보)은 각 슬롯의 용도(Beacon, Voice, Video), 송신노드 및 MCS level로 자원할당 정보를 표시한다. 총 12+78=90개의 슬롯이 존재하고 선두 12개는 제어 패킷으로 각 노드에 MCS로 할당된 것을 확인할 수 있다. 그 다음에는 데이터 슬롯들이 할당이 되고 있는데 송신 노드와 전송 트래픽은 임의로 선택 되어지고 송신 MCS 레벨은 모두 1로 통일된 것을 볼 수 있다. 노드가 움직이지 않아 해당 토폴로지가 변하지 않는다면 노드들은 해당 자원할당 정보만을 이용해 자신의 데이터를 송신한다.

시뮬레이션 결과 파일 최하단부에는 데이터 트래픽 종류별 성능지표가 표기된다. 그림 6에서는 이를 일부분 발췌했다. 영상 트래픽의 성능지표를 먼저 확인을 할 수 있는데, 송신 노드가 9이고 이를 수신한 노드들이 열로 구분되어 있으며, 몇 개의 패킷을 받았는지에 따라서 패킷 전송 성공률(1-패킷 전송 손실률)과 지연시간(최대, 최소, 평균)의 통계가 행으로 나타나있다. 하단으로 스크롤을 내리면 위치, 음성, 전문 순으로 동일한 성능지표를 확인할 수 있다. 지연시간의 경우,

```
---------- Simulation Results ----------
 Topology Algorithm : Traffic Concentration Considering  : 57
-------------------------------------------------------------------------------------------------------------------------------------------------------

 Video Traffic Interval: [ 0.032000, 0.034000 ], Size: [ 1000, 1500]
 Video Traffic Results:
   Tx node 9
   Rx Node : /    NC    /    1     /     2     /     3     /     4     /     5     /     6     /     7     /     8     /     9     /    10     /    11
   No Tx:    /      698 /      698 /      698 /      698 /      698 /      698 /      698 /      698 /      698 /  NOT Tx  /      698 /      698
   No Rx:    /      698 /      698 /      698 /      698 /      698 /      698 /      698 /      698 /      698 /  NOT Rx  /      698 /      698
Pro. Suc:    / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 /  NOT Rx  / 1.000000 / 1.000000
Ave. Del:    / 0.070106 / 0.042621 / 0.019422 / 0.047824 / 0.047824 / 0.024627 / 0.024626 / 0.024624 / 0.024622 /  NOT Rx  / 0.024622 / 0.024624
Max Del:     / 0.085267 / 0.062231 / 0.085267 / 0.085269 / 0.067433 / 0.067433 / 0.067433 / 0.067433 / 0.067433 /  NOT Rx  / 0.067433 / 0.085269
Min Del:     / 0.046593 / 0.024010 / 0.008260 / 0.029213 / 0.029213 / 0.013465 / 0.013464 / 0.013462 / 0.013460 /  NOT Rx  / 0.013460 / 0.013462
-------------------------------------------------------------------------------------------------------------------------------------------------------

 Position Traffic  Period: 1.000000, Size: 15
 Position Traffic Results:
-------------------------------------------------------------------------------------------------------------------------------------------------------
   Tx node 0
   Rx Node : /    NC    /    1     /     2     /     3     /     4     /     5     /     6     /     7     /     8     /     9     /    10     /    11
   No Tx:    /  NOT Tx  /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24
   No Rx:    /  NOT Rx  /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24
Pro. Suc:    /  NOT Rx  / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000
Ave. Del:    /  NOT Rx  / 0.042508 / 0.049028 / 0.054233 / 0.055585 / 0.055585 / 0.055585 / 0.055585 / 0.055585 / 0.055585 / 0.055585 / 0.054233
Max Del:     /  NOT Rx  / 0.085881 / 0.074158 / 0.080678 / 0.080678 / 0.080678 / 0.080678 / 0.080678 / 0.080678 / 0.080678 / 0.080678 / 0.080678
Min Del:     /  NOT Rx  / 0.017521 / 0.024041 / 0.029246 / 0.030561 / 0.030561 / 0.030561 / 0.030561 / 0.030561 / 0.030561 / 0.030561 / 0.029246
-------------------------------------------------------------------------------------------------------------------------------------------------------
   Tx node 1
   Rx Node : /    NC    /    1     /     2     /     3     /     4     /     5     /     6     /     7     /     8     /     9     /    10     /    11
   No Tx:    /       24 /  NOT Tx  /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24
   No Rx:    /       24 /  NOT Rx  /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24 /       24
Pro. Suc:    / 1.000000 /  NOT Rx  / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000 / 1.000000
Ave. Del:    / 0.033696 /  NOT Rx  / 0.038898 / 0.038901 / 0.053586 / 0.053586 / 0.053586 / 0.053586 / 0.053586 / 0.053586 / 0.053586 / 0.038901
Max Del:     / 0.077408 /  NOT Rx  / 0.072205 / 0.078725 / 0.078725 / 0.078725 / 0.078725 / 0.078725 / 0.078725 / 0.078725 / 0.078725 / 0.078725
Min Del:     / 0.008401 /  NOT Rx  / 0.013604 / 0.013606 / 0.027961 / 0.027961 / 0.027961 / 0.027961 / 0.027961 / 0.027961 / 0.027961 / 0.013606
-------------------------------------------------------------------------------------------------------------------------------------------------------
```

그림 6. 트래픽 속성 별 성능 지표
Fig. 6. Performance metric for each traffic

APP_Main.ps로 패킷이 수신이 되면, 해당 패킷을 열어 오리지날 APP 트래픽 패킷을 복원한 뒤 이를 완료한 시간과 해당 패킷이 생성된 시간을 빼 계산을 하고 이를 variable.h내 전역 변수(rx_delay)에 모두 저장하여 평균과 최대, 최소를 구한 것이다. 해당 성능 지표는 제공하는 프레임워크에서 기본적으로 구현한 것이고, APP_Main.ps과 variable.h의 수정을 통해 관찰하고자 하는 성능지표를 확장할 수 있다.

## V. 결 론

본 논문에서는 Riverbed Modeler의 라우팅 알고리즘을 추가/수정하기 힘든 점을 극복하기 위해 수정이 용이한 TDMA 기반의 라우팅 프로토콜 프레임워크를 제안하였다. 제안하는 라우팅 프로토콜 프레임워크는 IP 프로토콜 등 타프로세서와 연관되는 프로세서 없이 설계/개발되었기 때문에 새로운 라우팅 프로토콜을 최소한의 수정으로 추가하여 성능분석 모의할 수 있도록 구성되었다. 라우팅 프로토콜 프레임워크를 이용하여 라우팅 프로토콜을 수정하는 경우, 수정해야 하는 부분은 라우팅 알고리즘과 일부 패킷의 헤더 정보뿐이다. 본 논문에서는 프레임워크의 구조에 대한 상세한 설명을 통해 이용자들이 추가/수정을 쉽도록 하였으며, 또한 다익스트라 라우팅 프로토콜을 제안한 프레임워크를 이용하여 구현하여, 기본적인 성능지표들을 얻어 라우팅 프로토콜 프레임워크가 타당하게 설계 되었음을 확인하였다. 본 라우팅 프레임워크를 이용하여 추후 라우팅 프로토콜 시뮬레이션에 이용하면 개발기간을 크게 단축시킬 수 있을 것으로 기대된다.

## References


[1] NS-3, *NS-3 Network Simulator*, Retrieved Jul. 29, 2022, from http://nsnam.org/about/what-is-ns-3.

[2] Riverbed, *Riverbed Modeler*, Retrieved Jul. 29, 2022, from http://www.riverbed.com/products/npm/riverbed-modeler.html.

[3] A. D. Tesfamicael, V. Liu, E. Foo, and W. Caelli, "Modeling for performance and security balanced trading communication systems in the cloud," in *Proc. 2017 IEEE 36th IPCCC*, San Diego, CA, USA, Dec. 2017. (https://doi.org/10.1109/PCCC.2017.8280506)

[4] P. Mounika, "Performance analysis of wireless sensor network topologies for Zigbee using riverbed modeler," in *Proc. 2018 IEEE 2nd ICISC*, Coimbatore, India, Jan. 2018. (https://doi.org/10.1109/ICISC.2018.8399050)

[5] S. Lee, J. Ali, and B.-H. Roh, "Performance comparison of software defined networking

simulators for tactical network: Mininet vs. OPNET," in *Proc. 2019 IEEE ICNC*, Honolulu, HI, USA, Feb. 2019. (https://doi.org/10.1109/ICCNC.2019.8685572)

[6] K. Kim, C.-W. Lee, S.-H. Shin, B.-H. Roh, B. Roh, and M.-H. Han, "Effective routing protocol implementation framework on riverbed (OPNET) modeler and its example for AntHocNet," *J. KICS*, vol. 41, no. 8, pp. 974-985, Aug. 2016. (https://doi.org/10.7840/kics.2016.41.8.974)

[7] V. Hnatyshin, H. Asenov, and J. Robinson, "Practical methodology for modeling wireless routing protocols using OPNET Modeler," in *Proc. IASTED MS 2010*, vol. 696, no. 23, Jul. 2010. (https://doi.org/10.2316/p.2010.696-023)

[8] R. Al-maharmah, G. Bruck, and P. Jung, "Practical methodology for Adding new MANET routing protocols to OPNET," in *Proc. SIMUL 2013*, pp. 73-80, Oct. 2013.

**임 진 택** (Jin-Taek Lim)

2012년 8월 : 연세대학교 전기전자공학부 학사
2014년 8월 : 한국과학기술원 전기 및 전자공학과 석사
2019년 2월 : 한국과학기술원 전기 및 전자공학과 박사
2019년 3월~현재 : 국방과학연구소 선임연구원

<관심분야> 이동통신, 사물인터넷, 보안통신

[ORCID:0000-0002-9649-0459]

**김 수 진** (Sujin Kim)

2006년 2월 : 이화여자대학교 정보통신학과 학사
2008년 2월 : 한국과학기술원 정보통신학과 석사
2012년 8월 : 한국과학기술원 정보통신학과 박사
2012년 10월~현재 : 국방과학연구소 선임연구원

<관심분야> 정보통신, 무선통신, 모뎀/네트워크

**유 지 상** (Jisang You)

2003년 2월 : 인하대학교 전자공학과 학사
2005년 2월 : 광주과학기술원 정보통신공학과 석사
2020년 8월 : 한국과학기술원 전기 및 전자공학과 박사
2005년 2월~현재 : 국방과학연구소 책임연구원

<관심분야> 무선통신, 통신신호처리, 유무인 복합체계 원격제어

[ORCID:0000-0001-6650-0137]

**강 홍 구** (Hongku Kang)

1998년 2월 : 성균관대학교 전자공학과 학사
2000년 2월 : 광주과학기술원 정보통신공학과 석사
2004년 2월 : 광주과학기술원 정보통신공학과 박사
2004년 2월~현재 : 국방과학연구소 책임연구원

<관심분야> 무선통신, 통신신호처리, OFDMA, 무인로봇

[ORCID:0000-0002-7851-1148]