논문 16-41-08-20 The Journal of Korean Institute of Communications and Information Sciences '16-08 Vol.41 No.08
http://dx.doi.org/10.7840/kics.2016.41.8.974

# Riverbed (OPNET) Modeler의 효과적인 라우팅 프로토콜 추가 프레임워크 및 이를 이용한 AntHocNet 라우팅 구현

김 광 수[•], 이 철 웅[*], 신 승 훈[**], 노 병 희[°], 노 봉 수[***], 한 명 훈[***]

# Effective Routing Protocol Implementation Framework on Riverbed (OPNET) Modeler and its Example for AntHocNet

Kwangsoo Kim[•], Cheol-Woong Lee[*], Seung-hun Shin[**], Byeong-hee Roh[°],
Bongsoo Roh[***], Myoung-hun Han[***]

요 약

Riverbed Modeler는 복잡한 통신 프로토콜과 큰 규모의 네트워크를 설계하기 위한 패킷 수준의 이산 사건 시뮬레이터이며 그 신뢰성을 인정받아 대규모 네트워크의 성능분석에 널리 활용되고 있다. Riverbed Modeler를 활용하는 MANET 시뮬레이션 환경에서, 새로운 라우팅 프로토콜을 구현하여 추가하는 방법이 매우 복잡하고 많은 부분의 수정을 요구한다. 본 논문에서는 Riverbed Modeler의 라우팅 지원 구조에 대하여 살펴보고, 라우팅 추가에 대한 어려움을 해결하기 위하여 보다 쉽고 실수의 가능성을 줄일 수 있는 라우팅 추가 프레임워크를 제안하였다. 라우팅 추가 프레임워크는 프로토콜을 인식하는 부분에 대하여 적응적 구조 갖는 API로서 제공되며, 라우팅 프로토콜을 최소한의 수정으로 추가할 수 있도록 구성하였다. 라우팅 추가 프레임워크를 이용하여 라우팅 프로토콜을 추가하는 경우, 수정해야 하는 부분을 기존의 절반 이하로 간소화 하였다. 또한 제안한 라우팅 추가 프레임워크를 이용하여 Hybrid 라우팅 프로토콜인 AntHocNet을 구현하여 추가한 사례를 제시하여, 라우팅 추가 프레임워크가 타당하게 설계 및 적용되었음을 확인하였다.

**Key Words** : Riverbed Modeler, OPNET Modeler, Routing Adding F/W, Adapter Library, AntHocNet

ABSTRACT

Riverbed Modeler, which is a commercial packet-level discrete event simulator is used to model, design, and simulate complicated communication protocols and large-scale network. Riverbed Modeler got credit for its reliability in field of network simulation. In the MANET simulation environment using Riverbed Modeler, it is

※ 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원 (No. NRF-2015R1A2A2A01005577) 및 국방과학연구소의 지원 (ADD-IBR-245)을 받아 수행된 연구임.
※ 본 논문은 2016년 한국통신학회 동계종합학술발표대회에 발표된 논문을 확장하였습니다.[1]
• First Author : Ajou University, Graduate School, Department of Computer Engineering, zubilan@ajou.ac.kr, 학생회원
° Corresponding Author : Ajou University, Graduate School, Department of Computer Engineering, bhroh@ajou.ac.kr, 종신회원
* Ajou University, Graduate School, Department of Computer Engineering, cjfdnd369@ajou.ac.kr, 학생회원
** Ajou University, Dasan University College, sihnsh@ajou.ac.kr, 정회원
*** Agency of Defense Development, saintroh@add.re.kr, 정회원 mengddor@add.re.kr
논문번호 : KICS2016-05-082, Received May 1, 2016; Revised July 22, 2016; Accepted July 22, 2016

very complicated to add a new routing protocol into existing architecture of routing protocols because it is required lots of modifications of protocol recognition. In this paper, we propose Routing Adding Framework which can reduce errors or mistakes during modifying the existing routing support architecture. Routing Adding Framework is provided as a adapter API for protocol recognition. and it is only minimum modifications for protocol identifiers when a new routing protocol is added to the child process of manet_mgr process which manages routing protocols for IP layer. With Routing Adding Framework, we can reduce less than half modification. Then, we shows an example of implementation of a hybrid routing protocol AntHocNet using Routing Adding Framework, and we verify its design and application of the Routing Adding Framework by obtaining simulation result with similar result given by AntHocNet.

## Ⅰ. 서 론

Riverbed Modeler (舊 OPNET Modeler)는 복잡한 통신 프로토콜과 큰 규모의 네트워크를 설계하기 위한 패킷 수준의 이산 사건 시뮬레이터 (Discrete Event Simulator)이다. Riverbed Modeler는 네트워크 각 계층의 프로토콜 및 각종 동작을 직관적으로 모델링 할 수 있는 도구로서, 네트워크 시뮬레이터로서 신뢰성을 인정받아 대규모 네트워크의 설계 및 성능 분석에 광범위하게 활용되고 있으며, 이러한 신뢰성을 바탕으로 우리나라뿐만 아니라 미국 등 여러 나라의 국방 전술통신 분야의 모델링 및 시뮬레이션 도구로써 활용되고 있다[2].

Riverbed Modeler에서의 라우팅 프로토콜의 구조는 크게 세 가지의 형태로 구성되어 있는데, 그 중 IP 계층 프로세스의 MANET 라우팅 프로토콜 관리자 (manet_mgr 프로세스)에 의해 관리되는 형태의 경우, 라우팅 프로토콜의 인식 및 처리하는 부분이 IP 계층 및 ip_rte_support.ex.c 외부 소스 파일에 상당히 광범위하게 분포되어 있는 실정이다. 따라서 이러한 환경에서 새로운 라우팅 프로토콜을 구현하여 추가하는 경우, 일일이 찾아 추가 및 변경해 주어야 하는 문제점이 있으며, 누락 등의 알아채기 힘든 실수로 인하여 라우팅 프로토콜의 이상 동작을 초래할 수 있다.

이러한 어려움에 대하여, 기존의 라우팅 프로세스를 기반으로 내부 동작을 변형하고 차이점을 추가하는 방법[3]이 제안되었으나 이는 실제 프로토콜을 추가하는 방법을 다루고 있지 않았다. 이후 새로운 라우팅 프로토콜을 추가하는 방법[4]이 제시되었으나, 새로운 프로토콜의 등록에 대하여 제시되었을 뿐, IP계층 및 ip_rte_support.ex.c 외부 소스 파일의 방대한 프로토콜의 인식부분은 다루고 있지 않아 라우팅 프로토콜을 추가할 때 보다 효율적으로 추가할 수 있는 방법이 없는 실정이다.

본 논문에서는 이러한 라우팅 프로토콜 추가의 환경적/실무적 어려움을 효율적으로 극복하기 위하여, 라우팅 추가 프레임워크를 제안한다. 라우팅 추가 프레임워크는 Riverbed Modeler의 복잡한 IP 계층의 프로세스 및 ip_rte_support.ex.c 외부 소스 파일의 전반에 걸쳐 존재하는 "설정된 라우팅 프로토콜을 인식하는 부분"을 효율적으로 지원하고 손쉽게 라우팅 프로토콜을 추가할 수 있는 적응적 라이브러리 모델이다. 라우팅 추가 프레임워크는 최소한의 수정으로 새로운 라우팅 프로토콜을 추가할 수 있도록 지원하여, 라우팅 프로토콜 추가 과정에서 실수로부터 발생할 수 있는 오류를 줄여주는데 큰 역할을 할 것으로 기대한다.

논문의 구성은 다음과 같다. 2장에서는 Riverbed Modeler의 개요 및 라우팅 프로토콜의 동작구조에 대하여 살펴보고, 3장에서는 Riverbed Modeler를 위한 라우팅 추가 프레임워크의 구조 및 적용과정에 대하여 상세히 다룬다. 4장에서는 제안한 라우팅 추가 프레임워크를 기반으로 Hybrid 방식의 라우팅 프로토콜을 Riverbed Modeler에 구현하여 추가하는 방법 및 그 결과를 논의하고, 5장에서 본 논문의 결론을 맺는다.

## Ⅱ. Riverbed Modeler의 개요 및 라우팅 프로토콜의 동작 구조

Riverbed Modeler는 복잡한 통신 프로토콜과 큰 규모의 네트워크를 설계하기 위한 패킷 수준의 이산 사건 시뮬레이터 (Discrete Event Simulator)이다. Riverbed Modeler는 네트워크 각 계층의 프로토콜 및 각종 동작을 직관적으로 모델링 할 수 있는 도구로서, 매우 높은 정확도의 모델링, 확장 가능한 시뮬레이션, 다양한 유/무선 네트워크에 대한 상세 분석을 제공한다. 패킷 수준의 동작을 모델링 할 수 있는 특징을 가지며, 유/무선 프로토콜 설계 및 개발, 운영 환경에 적용하기 앞서 현실적인 시나리오에서의 기술 설계의

테스트와 시연 등을 수행할 수 있다. Riverbed Modeler는 네트워크 수준 시뮬레이터로서의 신뢰성을 인정받아, 대규모 네트워크의 설계 및 성능 분석에 광범위하게 활용되고 있으며, 이러한 신뢰성을 바탕으로 우리나라뿐만 아니라 미국 등 여러 나라의 국방 전술통신 분야의 모델링 및 시뮬레이션 도구로써 주로 활용되고 있다[2].

### 2.1 Riverbed Modeler의 개요 및 라우팅 프로토콜의 동작 구조

통신을 위한 네트워크 계층 모델에서의 IP 계층은 수집된 목적지까지의 경로를 바탕으로 패킷을 전달하는 것을 담당하며, Riverbed Modeler에서는 ip_dispatch 프로세스에서 수행한다. 기본적으로 모든 패킷은 IP 계층을 거치며, ip_dispatch 프로세스에서는 IP계층의 공용 라우팅 테이블 (cmn_route_table)을 참고하여 요청받은 패킷을 다음 노드 (nexthop)로 전달한다. 여기에서 패킷을 전달하기 위한 경로를 수집하는 프로토콜을 라우팅 프로토콜이라 하며, 라우팅 프로토콜은 네트워크 계층을 지원하기 위한 응용계층의 프로토콜로서 IP계층의 라우팅 테이블을 구성하는 역할을 수행한다.

Riverbed Modeler에서의 라우팅 프로토콜은 크게 세 가지 방법으로 적용되어 있다. 그림 1은 일반 라우터 노드 모델을 보여주며, 그림 1의 (a)는 라우터를 위한 프로토콜이 독립적인 프로세스의 형태로 네트워크 계층 모델에 적용되어 있음을 보여준다. 그림 2는 MANET (Mobile Adhoc Networks) 노드 모델로서, 그림 2의 (b) 및 (c)는 MANET 라우팅 프로토콜이 자식프로세스의 형태로 구성되어 있음을 보여준다. 본 논문에서는 그림 2의 (c)의 구조에 대하여, IP 계층의

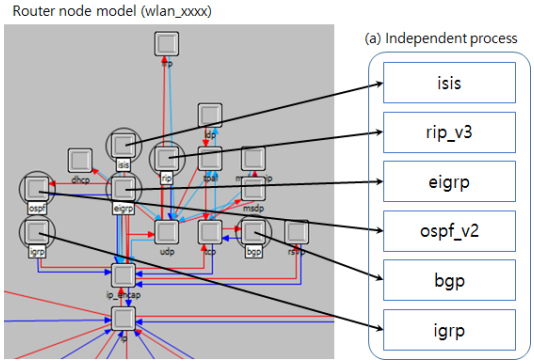



그림 1. 라우터 노드 모델 및 라우팅 프로토콜의 구조 (a) 독립적인 프로세스의 형태
Fig. 1. Router node model and the architecture of routing protocols (a) independent process

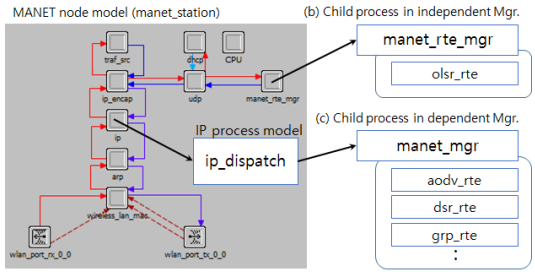



그림 2. MANET 노드 모델 및 라우팅 프로토콜의 구조 (b) 독립 프로세스에 포함된 자식 프로세스의 형태 (c) IP 계층에 포함된 프로세스의 자식 프로세스의 형태
Fig. 2. MANET node model and the architecture of routing protocols (b) child process in indep. mgr. (c) child process in dep. mgr.

라우팅 프로토콜을 손쉽게 추가할 수 있는 구조를 제안한다.

### 2.2 MANET 라우팅의 종류와 Riverbed Modeler의 적용

MANET은 기반망이 갖추어지지 않은 환경에서 이동성을 갖는 수 많은 노드가 자율적으로 통신하기 위한 네트워크 구성으로서, 자가구성, 자가관리, 자가치유 등의 능력을 갖추어야 한다. 이러한 특징으로 MANET은 전투무선망 및 재해재난망 등의 환경에 매우 적합하며, 효과적으로 통신을 유지하기 위한 많은 라우팅 방법이 연구되었다. MANET에서의 라우팅 프로토콜은 Proactive 방식과 Reactive 방식, 그리고 이 두 가지를 혼용한 Hybrid 방식으로 구분할 수 있다[5].

Proactive 방식의 라우팅 프로토콜은 경로의 사용 이전에 모든 목적지에 대하여 주기적이고, 능동적으로 경로의 정보를 수집하고 경로를 확보하여 두는 방식으로, 패킷의 전송 요청 이전에 수행되는 것을 전제로 한다. 따라서, Proactive (Table-driven) 방식의 라우팅에서는 패킷의 전달 요청 시 라우팅 테이블 (cmn_route_table)에 경로가 존재하면 즉시 해당 경로를 이용하여 전달할 수 있으나, 목적지로의 경로가 없으면 전송할 수 없는 목적지로 판단하여 패킷을 폐기한다. 이러한 방식은 토폴로지 변화에 민감하게 반응할 수 있으며, 빠르게 경로를 수정할 수 있지만, 대규모의 네트워크에 적합하지 않은 단점이 있다[5]. OLSR[6] 은 대표적인 MANET을 위한 Proactive 방식의 라우팅 프로토콜로서, 그림 2의 (b)와 같이 독립적인 관리 프로세스 (manet_rte_ mgr)의 자식 프로세스로 구성되어 있다. OLSR은 프로토콜의 동작과 패킷의 전달이 완전히 분리되어 있기 때문에 패킷의 전달 시 라우팅 프로토콜과 상호작용하지 않는다.

Reactive 방식의 라우팅 프로토콜은 Proactive 방식과는 반대로 패킷의 전달 요청이 있을 때 경로를 탐색한다. Riverbed Modeler에는 대표적인 Reactive 프로토콜인 AODV[7]와 DSR[8]이 적용되어 있다. 라우팅 등록된 라우팅 프로토콜이 Proactive 유형인 경우, 경로가 없으면 패킷을 폐기한다. 반면, Reactive 유형인 경우, 경로가 없는 경우 새로운 경로를 찾는 과정을 수행한다. AODV 등의 Distance Vector 기반의 라우팅 프로토콜은 hop-by-hop (nexthop) 방식으로서 IP 계층의 라우팅 테이블 (cmn_route_table)을 확인하여 목적지 정보가 있으면 해당 이웃 노드로 패킷을 전달하고 없는 경우에만 라우팅 프로토콜로 전달한다. 반면 DSR과 같이 소스 노드에서 경로를 지정하여 전달하는 Source Routing 기반의 프로토콜은 모든 패킷을 라우팅 프로토콜로 전달하여 Nexthop을 확인하고, 다음 노드로 패킷을 전달한다. 이러한 Reactive 방식의 라우팅 프로토콜은 IP 계층에서 패킷의 전달하는 과정에서 경로가 없으면 해당패킷을 라우팅 프로토콜에서 넘겨 받아 경로를 탐색할 수 있도록 구성되어야 한다. 따라서, 모든 Reactive 방식의 프로토콜 및 IP 계층과 상호작용을 해야 하는 라우팅 프로토콜은 모두 그림 2의 (c)와 같은 방법으로 구성되어야 한다.

Hybrid 방식의 라우팅 프로토콜은 대표적으로 ZRP[9]와 최근 집단생태기반의 AntHocNet[10]가 제안되었다. ZRP는 가까운 영역의 경로는 Proactive 방식으로 탐색 및 유지를 수행하고, 확인되지 않은 먼 영역에 대한 경로 탐색은 Reactive 방식으로 수행하는 방법으로 Scalability를 향상한다. 반면, AntHocNet은 개미의 활동을 모방한 알고리즘으로 경로의 탐색은 Reactive 방식으로 수행하고, 다중 경로의 지원을 위한 대체 경로 탐색은 경로 생성 이후 Proactive 방식으로 수행한다. Riverbed Modeler에는 Hybrid 방식의 프로토콜이 적용되어 있지 않으나, Reactive 방식의 처리를 수행할 수 있어야 하기 때문에, 그림 2의 (c)의 방법으로 적용 가능하다.

### 2.3 라우팅 프로토콜 추가의 어려움

Riverbed Modeler에서의 그림 2의 (c) 유형으로 구성되어 있는 라우팅 프로토콜은 ip_dispatch 프로세스 모델과 manet_rte 프로세스 모델, ip_rte_support 등의 외부 소스 파일과 밀접한 관련을 맺고 있다. ip_dispatch 프로세스 모델에서는 그림 3과 같이 패킷 수신 시 각 라우팅 프로토콜 별 분기를 통해 라우팅 프로토콜을 인식하고 적절한 동작을 취하도록 한다. 표 1은 새로운 라우팅 프로토콜을 manet_mgr에 추가하기 위하여 수정해 주어야 하는 부분을 보여주고 있다. 이는 새로운 라우팅 프로토콜을 추가할 때 해당 부분에 매번 적절한 라우팅 프로토콜의 동작에 해당하는 분기문 및 프로토콜을 인식하는 부분을 추가해 주어야하기 때문에 실수의 가능성이 존재하며, 그 실수를 알아채기 쉽지 않다.

```
/* Check if MANET is enabled on this interaface */
if (ip_interface_routing_protocols_contains
                    (routing_protocols_lptr, IpC_Rte_Dsr))
    {
    ip_manet_enable (&module_data, IpC_Rte_Dsr);
    iface_info_ptr->flags |= IPC_INTF_FLAG_MANET_ENABLED;
    active_manet_protocols |= IPC_RTE_PROTO_DSR;
    }
else if (ip_interface_routing_protocols_contains
                    (routing_protocols_lptr, IpC_Rte_Tora))
    {
    ip_manet_enable (&module_data, IpC_Rte_Tora);
    iface_info_ptr->flags |= IPC_INTF_FLAG_MANET_ENABLED;
    active_manet_protocols |= IPC_RTE_PROTO_TORA;
    }

// AODV
else if (ip_interface_routing_protocols_contains
                    (routing_protocols_lptr, IpC_Rte_Aodv))
    {
    ip_manet_enable (&module_data, IpC_Rte_Aodv);
    iface_info_ptr->flags |= IPC_INTF_FLAG_MANET_ENABLED;
    active_manet_protocols |= IPC_RTE_PROTO_AODV;
    num_mgtwy_interfaces++;
    }
```

그림 3. ip_dispatch 프로세스 모델에서의 라우팅 프로토콜 인식
Fig. 3. Routing protocol recognition in ip_dispatch process

표 1. 라우팅 프로토콜 추가를 위해 수정이 필요한 코드 분포
Table 1. Codes analysis for creating a new routing protocol

| | |
|---|---|
| **Process models** | |
| ip_dispatch.pr.m | 14 places |
| manet_mgr.pr.m | 2 places |
| **External sources** | |
| ip_rte_support.ex.c | 14 places |
| ip_cmn_rte_table.ex.c | 1 place |
| **Headers** | |
| ip_cmn_rte_table.h | 2 places |
| ip_rte_v4.h | 2 places |
| ip_higher_layer_proto_reg_sup.h | 1 place |
| ip_rte_support.h | 3 places |

*. in case of Riverbed Modeler v17.5

### 2.4 기존의 Riverbed Modeler를 이용한 라우팅 관련 연구

Riverbed Modeler (舊 OPNET Modeler)는 신뢰성을 인정받은 상용 이산 사건 시뮬레이터로서, 네트워크 분야의 전반에 걸쳐 광범위하게 활용되어 왔다. 참고문헌 [3]에서는 AODV를 기반으로 GPS를 활용한 라우팅 방법을 참고하여 Riverbed Modeler에 구현하였다. 구현에 사용한 방법은 “굉장히 많은 양의 소스

코드 수정이 필요한, 새로운 프로세스를 만드는 방법" 대신, AODV 프로세스 모델을 수정하는 방법으로 구현하였다. 이러한 방법은 구현 대상 알고리즘이 GPS 정보를 추가하는 수준의 수정으로 가능했기 때문으로, 새로운 프로토콜을 구현하기 위하여 새로운 프로세스 모델을 추가할 것을 권장하고 있다. 하지만, 수정하여야 하는 부분의 일부인 manet_mgr의 프로토콜 등록에 대한 부분만을 언급하고 있을 뿐, IP 계층 및 ip_rte_support 외부 소스 파일에 대한 수정은 다루고 있지 않다.

참고문헌 [4]에서는 체계적이고 쉽게 새로운 라우팅 프로토콜 모델을 Riverbed Modeler에 추가하는 방법을 제공하고 있으며, 이를 이용하여 에너지를 효율적으로 사용할 수 있도록 Cluster Head를 선출하는 MACHM (Multi-Aware Cluster Head Maintenance) 프로토콜을 구현하였다. 하지만 [4]은 새로운 프로토콜을 추가하는 방법을 제시하였으나, [3]보다 상세히 설명한 수준에 지나지 않고, 라우팅 프로토콜을 추가하기 위하여 ip_dispatch 및 ip_rte_support 외부 소스 파일에서 필수적으로 수정되어야 하는 방대한 분량의 분기문에 대한 내용은 다루고 있지 않다.

이와 같이 MAC만을 대상으로 하거나, MAC 프로토콜과 직접 연결하여 통신하는 방법으로 Cross Layer 등을 지원하는 경우 등 네트워크 차원에서는 Riverbed Modeler에서 제공하는 라우팅 프로토콜을 그대로 사용하고 있는 경우가 많으며, Riverbed Modeler의 복잡성 때문에 라우팅 프로토콜을 직접 수정하거나 새로운 라우팅 프로토콜을 구현하는 사례는 드물다[11-14].

## Ⅲ. Riverbed Modeler를 위한 라우팅 추가 프레임워크

본 장에서는 "새롭게 설계한 프로토콜을 보다 쉽게 Riverbed Modeler에 추가할 수 있도록 하는 적응적 모델 (adapter)"로서 라우팅 추가 프레임워크를 제안한다. Riverbed Modeler에서의 라우팅 프로토콜은 IP 계층 프로세스 (ip_dispatch.pr.m)와 라우팅 지원 라이브러리 (ip_rte_support.ex.c) 등과 밀접한 관련을 맺고 있다. 이러한 환경에서 새로운 프로토콜을 추가할 경우 올바른 동작을 보장하기 위하여 기존의 구조에 수정 및 추가해 주어야 할 부분이 상당하다. 따라서, 향후의 수월한 알고리즘 개발 및 효과적인 성능 분석을 위하여 개발한 프로토콜 모듈을 선택 및 인식하도록 하는 적응적 모델의 도입이 필요하고, 이를 라우팅 추가 프레임워크라 명명하였다.

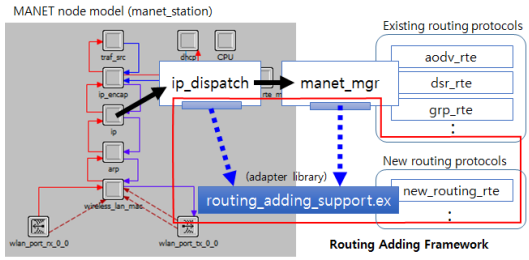



그림 4. 라우팅 추가 프레임워크의 구조
Fig. 4. Architecture of routing adding framework

제안하는 라우팅 추가 프레임워크는 Riverbed Modeler의 라우팅 프로토콜 구성 방식 중, 그림 2의 (c) IP계층 라우팅 프로토콜 관리자 (manet_mgr)의 자식 프로세스로 구성되어 있는 방식을 응용하여, 새로운 라우팅 프로토콜을 손쉽게 추가할 수 있도록 구성되어 있으며 제안하는 방식을 이용하여 Proactive 및 Reactive, Hybrid의 세 가지 MANET 라우팅 방식을 모두 지원할 수 있다.

그림 4는 제안하는 라우팅 추가 프레임워크의 구조 및 동작 방식을 나타낸다. 라우팅 추가 프레임워크는 일종의 적응적 라이브러리로, ip_dispatch 및 manet_mgr 프로세스의 각 분기문에서 반복적으로 사용되는 "프로토콜을 인식하는 부분 (프로토콜 ID 등)"을 API 함수로 구성하고, 각 분기문에 해당 API를 추가하여 프로토콜을 인식하는 부분에서 활용할 수 있도록 하여 라우팅 추가 시 작업의 양을 최소화할 수 있다.

### 3.1 프로토콜의 인식을 위한 식별자 정의

Riverbed Modeler의 IP계층 및 라우팅 추가 프레임워크에서 새로운 라우팅 프로토콜을 인식하기 위하여, 프로토콜을 위한 식별자를 부여하여 인식할 수 있도록 하여야 한다. 새로운 라우팅 프로토콜의 인식을 위하여 수정해 주어야 하는 부분은 표 2와 같다. ip_cmn_rte_table.h 헤더[1] 파일과 ip_rte_v4.h 헤더[2] 파일, ip_higher_layer_proto_reg_sup.h 헤더[3] 파일은 IP 계층에서 사용하는 라우팅 프로토콜들의 정보를

1) ip_cmn_rte_table.h 헤더 파일: 공용 라우팅 테이블의 각 경로에 기여한 프로토콜을 지정하기 위한 프로토콜 번호 정의
2) ip_rte_v4.h 헤더 파일: ip interface에서 선택 가능한 라우팅 프로토콜의 유형 정의
3) ip_higher_layer_proto_reg_sup.h 헤더 파일: IP 패킷의 헤더에 지정되는 프로토콜 번호 정의

표 2. 라우팅 프로토콜의 인식을 위한 부분
Table 2. Modified files and its positions to recognize routing protocols

| Files | Positions |
|---|---|
| ip_cmn_rte_table.h | enum IpT_Rte_Prot_Type<br>#define IPC_DYN_RTE_*xxxx* |
| ip_rte_v4.h | enum Ipt_Rte_Protocol<br>#define IPC_RTE_PROTO_*xxxx* |
| ip_higher_layer_proto_reg_sup.h | enum IpT_Protocol_Type |
| ip_cmn_rte_table.ex.c | char* IpC_Dyn_Rte_Prot_Names |
| routing_adding_framework.ex.c | char* arr_raf_protocol_name<br>IpT_Rte_Protocol arr_raf_protocol<br>int arr_raf_protocol_flag<br>char* arr_raf_protocol_str<br>int arr_raf_protocol_type<br>IpT_Rte_Prot_Type<br>arr_mmcn_ipc_rte_prot_type |

*. Italic체로 표시된 *xxxx*는 새로운 라우팅 프로토콜의 이름이다.

가지고 있다. ip_cmn_rte_table.ex.c에는 라우팅 프로토콜의 정보를 표시 (출력)하기 위한 문자열 배열을 정의하고 있으며, routing_adding_framework.ex.c은 라우팅 추가 프레임워크의 적응적 라이브러리로서 프로토콜을 인식하여 필요한 정보를 설정해 주기 위한 프로토콜 정보를 정의한다.

이러한 라우팅 프로토콜의 정보는 IP 계층의 초기화와 패킷의 처리 시에 IP 계층이 어떤 동작을 취해야 하는지를 결정할 수 있도록 한다. 이는 노드에 설정된 라우팅 프로토콜의 종류에 따라 각각 운용방식이 다르고 IP 계층과 상호작용을 통해 설정해야 할 값들이 있기 때문으로, 프로토콜의 종류에 등록된 프로토콜만이 정확한 동작을 보장할 수 있다.

### 3.2 프로토콜의 인식을 위한 API

새로운 라우팅 프로토콜을 위하여 정의된 식별자는 라우팅 추가 프레임워크를 거쳐 IP 계층 프로세스 (ip_dispatch.pr.m)와 라우팅 지원 라이브러리 (ip_rte_support.ex.c)에서 라우팅 프로토콜을 인식하는데 활용된다. 그림 5는 ip_dispatch 프로세스에서 라우팅 프로토콜을 인식하여 활성화하는 부분의 예이다. 그림 5의 상단의 프로토콜을 식별하기 위한 함수 (ip_interface_routing_protocols_contains)에서 (a) 라우팅 프로토콜의 목록과 비교하는 것으로 AODV 등의 기존 프로토콜의 인식 후 노드, 라우팅 프로토콜, interface의 초기화를 수행한다. 각각의 라우팅 프로토콜 별로 내부에서 수정되어야 하는 식별자가 상이하므로, 해당 부분은 API로 구성하여 간단하게 처리할 수 있다.

그림 5에서 **raf**_ip_interface_routing_protocols_contains 함수는 프로토콜 식별을 위한 라우팅 추가 프레임워크의 API이다. 그림 5의 (b)와 같이 프로토콜 별로 상이한 식별자를 반환하여 초기화를 수행할 수 있게 하였고, 그림 5의 (c)는 공통적인 부분으로 프로토콜 인식 후 처리하도록 하였다. ip_dispatch 프로세스 모델과 ip_rte_support 외부 소스 파일에는 이와 유사하게 프로토콜 별로 인식하여 비슷한 유형의 작업을 수행하는 부분이 많이 존재하며, 이를 그림 5와 같은 방식으로 인식하도록 하였다.

### 3.3 패킷 전달시 처리방법 결정

AODV 등의 Distance Vector 기반의 hop-by-bop (nexthop) 방식을 사용하는 프로토콜은 IP 계층에서 cmn_route_table을 활용하여 패킷을 전달한다. 반면, DSR 등의 Source Routing 방식은 nexthop의 결정을 라우팅 프로토콜에서 패킷에 들어있는 경로정보를 바

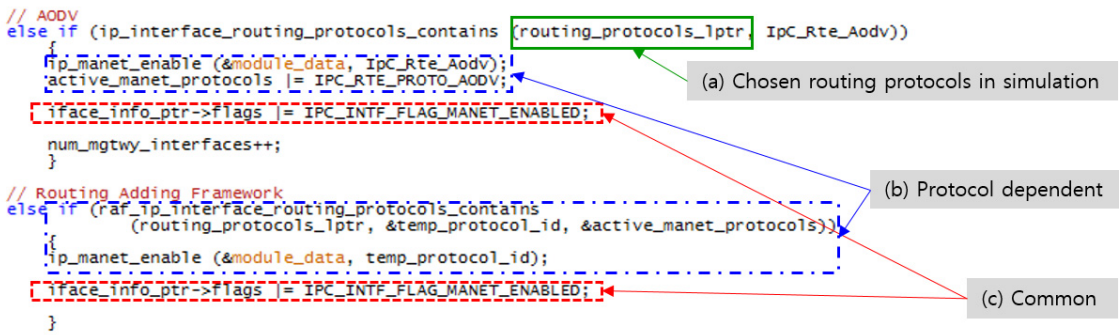



그림 5. ip_dispatch 프로세스에서 새로운 라우팅 프로토콜을 인식하는 부분
Fig. 5. Recognition of new routing protocols in ip_dispatch process model

탕으로 처리해 주어야 한다. 또한 현재의 IP 계층의 cmn_route_table은 Multipath 라우팅의 특징 및 이에 관련한 경로 선택 방법 (확률적 경로 선택 등)을 지원하지 않기 때문에 모든 패킷의 전달을 라우팅 프로토콜에서 수행하여야 한다. 즉, Distance Vector 방식은 경로가 라우팅 테이블에 없는 경우만 라우팅 프로토콜로 패킷이 전달되는 반면, Source Routing 방식은 모든 패킷이 라우팅 프로토콜로 전달되어 처리되어야 한다.

이를 위하여, 모든 데이터 패킷을 라우팅 프로토콜에서 처리하는 설정을 추가하였다. ip_rte_support.h 헤더에 정의된 "MANET 라우팅 프로토콜에서 사용하는 정보"를 담고 있는 IpT_Manet_Info 구조체에 all_packet_handled_by_routing 변수를 정의하였고, 라우팅 프로토콜의 초기화 시에 해당 변수를 활성화한다. 그림 6은 ip_rte_support 헤더와 외부 소스 파일에 반영된 부분이다. 그림 6의 (b)에서는 ip_rte_support 외부 소스 파일의 ip_rte_packet_arrival 함수에서, 기존 DV 방식 중 경로가 없는 경우 및 DSR과 같은 Source Routing 프로토콜의 경우 패킷을 MANET 라우팅 프로토콜로 전

```
typedef struct IpT_Manet_Info
    {
    Prohandle                   mgr_prohandle;
    IpT_Rte_Protocol            rte_protocol;
    AodvT_Route_Table*          aodv_route_table_ptr;

    // for dealing with all packets by routing protocol
    Boolean all_packet_handled_by_routing;

    Boolean                     manet_gateway_status;
    } IpT_Manet_Info;
```

(a) Definition of IpT_Manet_Info structure in ip_rte_support.h

```
if ((packet_from_lower_layer == OPC_FALSE) &&
    ((iprmd_ptr->manet_info_ptr->rte_protocol
                                == IpC_Rte_Dsr) ||
    (iprmd_ptr->manet_info_ptr->rte_protocol
                                == IpC_Rte_Grp)))
    {
    intf_ici_fdstruct_ptr->manet_redirect
                                = OPC_TRUE;
    }
// for dealing with all packets by routing protocol
else if
    (iprmd_ptr->manet_info_ptr->all_packet_handled_by_routing
                                        == OPC_TRUE)
    {
    intf_ici_fdstruct_ptr->manet_redirect
                                = OPC_TRUE;
    }
else if ((iprmd_ptr->manet_info_ptr->rte_protocol
                                == IpC_Rte_Tora) ||
        (iprmd_ptr->manet_info_ptr->rte_protocol
                                == IpC_Rte_Aodv) ||
        raf_ip_interface_routing_protocols_contains_unit
            (iprmd_ptr->manet_info_ptr->rte_protocol))
    {
    if (got_datagram_info == OPC_COMPCODE_FAILURE)
        {
        intf_ici_fdstruct_ptr->manet_redirect
                                = OPC_TRUE;
        }
    }
```

Source Routing

All packets

No route of DV Routing

(b) Redirection condition of ip_rte_packet_arrival function in ip_rte_support.ex.c

그림 6. MANET 라우팅 프로토콜의 유형별 패킷 처리 조건
Fig. 6. Redirection conditions to MANET routing protocols

달[4]하도록 되어있고, 이에 추가하여 해당 변수가 활성화된 라우팅 프로토콜의 경우도 MANET 라우팅 프로토콜에서 패킷을 처리하도록 하였다.

### 3.4 프로토콜의 등록

이후의 프로토콜 등록 및 사용은 [4]에 서술된 방법과 같다. 라우팅 프로토콜의 사용을 위하여 manet_mgr 프로세스 모델에 등록하는 절차가 필요하다. manet_mgr 프로세스 모델의 설정값으로 지정되어 있는 *AD-HOC Routing Parameters* 그룹에 라우팅 프로토콜을 선택할 수 있도록 프로토콜의 이름을 등록하고, 라우팅 프로토콜의 성능 분석에 사용할 파라미터를 등록한다. manet_mgr 프로세스 모델에서는 설정 속성에서 선택된 라우팅 프로토콜을 활성화 할 수 있도록 manet_mgr_routing_protocol_determine 함수와 manet_mgr_routing_process_create 함수를 새 라우팅 프로토콜에 맞게 수정하고, 새 라우팅 프로토콜을 manet_mgr 프로세스의 자식 프로세스로 추가한다.

### 3.5 라우팅 추가 프레임워크의 개선효과

지금까지 살펴본 라우팅 추가 프레임워크를 활용하여 구현한 라우팅 프로토콜을 추가하는 경우, 매번 수정해 주어야 하는 많은 부분을 간소화 할 수 있다. 그림 7은 라우팅 프로토콜의 추가를 위해 수정해야 하는 부분에 대하여 기존의 "매번 전체 소스를 수정하는 방식 (Brute Force)"과 제안하는 라우팅 추가 프레임

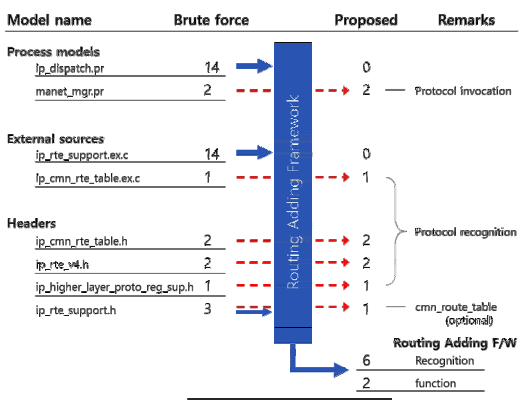



그림 7. 라우팅 추가 프레임워크의 개선효과
Fig. 7. The improvement effect by Routing Adding Framework

4) 패킷을 MANET 라우팅 프로토콜로 전달: manet_redirect 변수를 OPC_TRUE로 설정하면 이후에 패킷이 설정된 MANET 프로토콜로 전달된다.

워크를 사용하는 방식의 개선효과를 보여준다. 라우팅 추가 프레임워크를 이용할 경우 ip_dispatch 프로세스 모델과 ip_rte_support.ex.c 외부 소스 파일은 더 이상 수정하지 않아도 되며, 다만 라우팅 추가 프레임워크에서 새로운 프로토콜을 인식하기 위한 부분을 한 번에 수정하는 것으로 실수의 여지를 최소화할 수 있다.

## Ⅳ. 제안하는 프레임워크를 사용한 AntHocNet 라우팅 프로토콜 구현

### 4.1 AntHocNet 라우팅 프로토콜의 개요

본 장에서는 제안한 라우팅 추가 프레임워크를 이용하여 집단생태특성 기반의 Hybrid 라우팅 프로토콜인 AntHocNet 라우팅 프로토콜 (AntHocNet) [10]을 구현하여 추가한 사례를 제시한다. AntHocNet에서 활용하는 집단생태기반 알고리즘인 Ant Colony Optimization (ACO) [15]은 개미 집단의 특성을 활용하여 조합 최적화의 문제를 해결하는 방법으로, 네트워크의 라우팅 문제를 해결하는데 도입되어 많은 연구가 파생되었다. 개미 집단은 지나다닌 경로에 페로몬을 남겨두고 다음 개미가 그 페로몬을 따라 먹이를 찾아 갈 수 있도록 한다. 먹이를 찾아가는 과정에서 다양한 경로가 만들어 질 수 있지만 더 많은 개미가 다니는 경로를 최단, 최적의 경로로 판단할 수 있다. 그림 8은 ACO의 기본이 되는 개미 집단에서 장애물이 생겼을 경우의 최적 경로를 재탐색하는 과정을 보여주는 것으로, A는 경로가 생성된 상태를 보여주고, B는 중간에 장애물이 생긴 상황을 보여준다. 개미들은 C와 같이 장애물을 돌아 두 개의 경로를 생성하지만, 결국 D와 같이 개미가 더 많이 다닐 수 있는 경로(짧은 경로)가 최적의 경로로 선정된 것을 확인할 수 있다.

AntHocNet은 개미 집단의 역할 분담 및 페로몬의 확산을 응용한 Multipath 라우팅 프로토콜로, 목적지로의 경로가 없는 경우 Reactive Ant를 이용하여 경로를 탐색하고, 경로가 생성된 이후 Hello 메시지를 이용한 경로정보에 대한 페로몬의 확산을 수행한다. 이렇게 확산된 페로몬은 목적지로 갈 수 있다는 미확인 정보에 해당하고 Proactive Ant를 이용하여 목적지까지의 새로운 경로를 탐색한다. AntHocNet에서 제어 패킷 (Ant 패킷) 및 데이터 패킷의 전달은 페로몬의 양에 기반한 확률로 결정된다. 이러한 특징을 갖는 AntHocNet을 추가하기 위하여, 모든 패킷을 라우팅 프로토콜에서 관리 및 전달하도록 하였다.

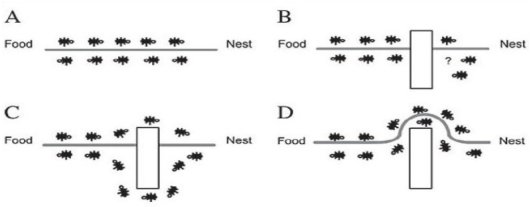



그림 8. 개미의 최적경로 탐색 과정[16]
Fig. 8. Ant finding shortest path[16]

### 4.2 제안 프레임워크 기반 AntHocNet 구현

우선, 라우팅 추가 프레임워크를 이용하여 AntHocNet을 추가하기 위하여, 그림 9와 같이 AntHocNet을 인식하기 위한 식별자를 등록하였다. 그림 9의 결과는 표 2의 수정 위치에 대응하는 부분으로, 열거자를 이용하여 식별자를 구성한다. 열거자에

```
typedef enum
    {
    IpC_Dyn_Rte_Invalid     = -1,
    IpC_Dyn_Rte_Directly_Connected = 0,
          :
    IpC_Dyn_Rte_Ospfv3      = 21,
    // for new protocol, AntHocNet
    IpC_Dyn_Rte_AntHocNet   = 22,
    IpC_Dyn_Rte_Number       /* KEEP LAST, auto increment */
    } IpT_Rte_Prot_Type;

// definition of new protocols from in struct IpT_Rte_Prot_Ty
#define IPC_DYN_RTE_AntHocNet       IpC_Dyn_Rte_AntHocNet
```

(a) ip_cmn_rte_table.h

```
typedef enum
    {
    IpC_Rte_Default = -99,
          :
    IpC_Rte_Ospf3,
    IpC_Rte_Static,
    // AMN, NSH, for new protocol
    IpC_Rte_AntHocNet
    } IpT_Rte_Protocol;

// new protocols
#define IPC_RTE_PROTO_AntHocNet     (1<<12)
```

(b) ip_rte_v4.h

```
typedef enum IpT_Protocol_Type
    {
    /* Background traffic tracer packets can */
    /* be encapsulated in ip_dgram packets.  */
    IpC_Protocol_Basetraf               = -2,
    IpC_Protocol_Unspec                 = -1,
          :
    IpC_Protocol_Tora                   = 254,
    IpC_Protocol_Ip_Mip                 = 300,
    // for new protocol
    IpC_Protocol_AntHocNet              = 301
    } IpT_Protocol_Type;
```

(c) ip_higher_layer_proto_reg_sup.h

```
const char* IpC_Dyn_Rte_Prot_Names[IPC_DYN_RTE_NUM] =
    {"Direct", "OSPF", "RIP", "IGRP", "BGP", "EIGRP", "IS-IS
    "Static", "EXT_EIGRP", "IBGP", "Default", "RIPng", "TORA
    "AODV", "OLSR", "Mobile IP", "LDP", "Custom", "Local", "
    // new protocol
    "AntHocNet"
    };
```

(d) ip_cmn_rte_table.ex.c

그림 9. AntHocNet의 인식을 위한 식별자의 추가
Fig. 9. Adding the identifier for recognizing AntHocNet

배정되는 프로토콜의 번호는 앞의 것 이후의 것으로 순서대로 사용하며, ip_cmn_rte_table.h 헤더의 IpC_Dyn_Rte_Number 는 라우팅 프로토콜의 전체 수를 자동으로 알기 위한 것으로 ip_cmn_rte_table 외부 소스 파일 등에서 활용된다. #define 상수는 새로 추가된 식별자를 반영한다.

그림 10은 라우팅 추가 프레임워크에 해당하는 routing_adding_framework.ex.c 외부 소스 파일에 AntHocNet의 인식을 위하여 식별자를 추가한 결과를 보여준다. CNT_RAF_PROTOCOLS 상수 정의는 새로 추가된 라우팅 프로토콜의 수이며, 인식을 위한 식별자는 그림 9에서 이미 정의된 식별자를 추가한다. 이렇게 추가된 라우팅 추가 프레임워크의 식별자는 API를 통해 프로토콜을 인식하는데 활용된다. 라우팅 추가 프레임워크에 추가해 주어야 하는 위치 중, 함수 내부의 수정에 대한 부분은 공용 라우팅 테이블을 사용하는 것과 그 프로토콜 아이디를 얻어내는 것에 대한 부분으로 AntHocNet에서는 자체 테이블을 사용하기 때문에 별도로 추가하지 않아도 무방하다.

그림 11은 구현한 anthocnet_rte를 manet_mgr 프로세스에 추가한 결과를 보여준다. 이렇게 추가한 AntHocNet 프로세스를 선택할 수 있도록 manet_mgr 프로세스의 모델 속성 편집창 *AD-HOC Routing Parameters* 그룹의 AD-HOC Routing Protocol에서 속성 값을 추가하면, 기본적으로 프로토콜을 추가하기 위한 작업을 마쳤다고 볼 수 있으며, 라우팅 프로토콜 선택 시 정상적으로 구동되어 동작함을 알 수 있다.

그림 12는 구현한 AntHocNet의 프로세스 모델인

```
// number of protocols newly added
#define CNT_RAF_PROTOCOLS 1

// protocol name that exists as a process
char*    arr_raf_protocol_name    [CNT_RAF_PROTOCOLS] =
                                  {"anthocnet_rte"};

// protocols which in the ip_rte_v4.h,
// in struct IpT_Rte_Protocol
IpT_Rte_Protocol
        arr_raf_protocol          [CNT_RAF_PROTOCOLS] =
                                  {IpC_Rte_AntHocNet};

int      arr_raf_protocol_flag    [CNT_RAF_PROTOCOLS] =
                                  {IPC_RTE_PROTO_AntHocNet};

char*    arr_raf_protocol_str     [CNT_RAF_PROTOCOLS] =
                                  {"AntHocNet"};

// ip_higher_layer_proto_reg_sup.h,
// in struct IpT_Protocol_Type
int      arr_raf_protocol_type    [CNT_RAF_PROTOCOLS] =
                                  {IpC_Protocol_AntHocNet};

// ip_cmn_rte_table.h,
// in struct IpT_Rte_Prot_Type
IpT_Rte_Prot_Type
        arr_raf_ipc_rte_prot_type [CNT_RAF_PROTOCOLS] =
                                  {IpC_Dyn_Rte_AntHocNet};
```

(e) routing_adding_framework.ex.c

그림 10. AntHocNet의 인식을 위하여 라우팅 추가 프레임워크에 식별자 추가
Fig. 10. Adding the identifier for AntHocNet in routing adding framework

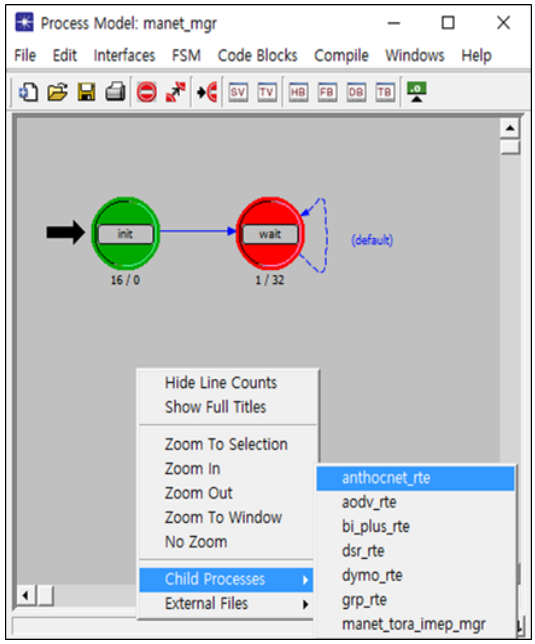
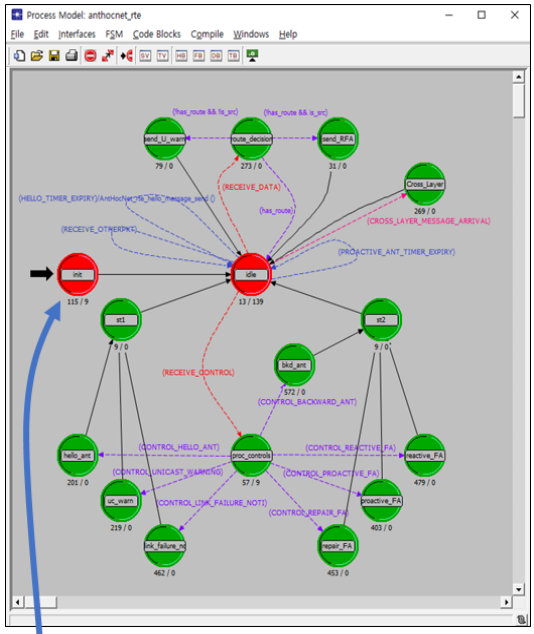



그림 11. manet_mgr 프로세스에 AntHocNet을 자식 프로세스로 추가
Fig. 11. Adding the AntHocNet as a child process of manet_mgr process




```
/* Register the AntHocNet Routing process as a higher layer in IP   */
higher_layer_proto_id = IpC_Protocol_AntHocNet;
Ip_Higher_Layer_Protocol_Register ("AntHocNet", &higher_layer_proto_id);

module_data_ptr->manet_info_ptr->all_packet_handled_by_routing = OPC_TRUE;
```

그림 12. AntHocNet 프로세스 모델과 초기화과정
Fig. 12. Process model for AntHocNet and its initialization

anthocnet_rte를 보여주며, 프로세스 모델의 초기화 과정에서 라우팅 프로토콜로서 IP 계층의 상위계층으로 등록하여 IP 계층까지 전달된 패킷이 AntHocNet 프로세스까지 전달될 수 있도록 한다. 또한, AntHocNet 프로토콜은 확률적 패킷 전달을 활용하는 Multipath 라우팅 프로토콜로서 모든 패킷을 라우팅 프로토콜에서 처리해 주어야 하며, 이를 위하여 초기화 단계에서 all_packet_handled_by_routing 변수를 활성화 하여 적절하게 동작할 수 있도록 하였다.

### 4.3 AntHocNet 구현 결과

본 연구에서 구현한 AntHocNet 라우팅 프로토콜이 적절히 구현되어 라우팅 추가 프레임워크로 추가되어 원활히 동작함을 보이기 위하여, AntHocNet[10]에서 제시한 성능 및 실험환경 (표 3)을 기준으로 실험을 수행하였다. 실험환경의 유사함을 보이기 위하여, AntHocNet뿐만 아니라, AODV 및 OLSR 프로토콜도 같이 비교하였으며, 각 라우팅 프로토콜에 대한 성능은 다음과 같다.

그림 13은 [10]에서 제시한 최대 이동속도에 따른 각 라우팅 프로토콜의 패킷 전달률을 보여준다. 그림 14는 Riverbed 시뮬레이터로 수행한 결과로, AODV 및 OLSR을 그림 13의 결과와 비교하였을 때 유사한

표 3. AntHocNet 구현 검증을 위한 시뮬레이션 파라미터
Table 3. Simulation parameters for validating the implemented AntHocNet

| Parameters | | Value |
|---|---|---|
| # of nodes | | 100 |
| Mobility domain | | 2400m * 800m |
| Simulation time | | 900 seconds |
| # of iterations | | 20 times |
| Mobility | Model | Random Waypoint |
| | Speed | Uniform (min=0, max_speed) |
| | max_speed | 1, 2, 5, 10, 20, 30 |
| | Pause time | 30 seconds |
| Traffic | # of sessions | 20 nodes |
| | Dist. model | Constant (CBR) |
| | Properties | 4 packets/sec, 64 bytes/packet |
| Routing Protocol | | AntHocNet, AODV, OLSR |
| MAC Protocol | | CSMA/CA (IEEE 802.11 DCF, WLAN) |
| Physical Setting | Data rate | 2 Mbps |
| | Tx range | 250m |

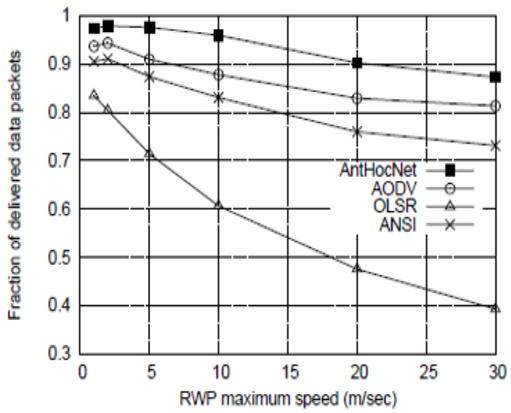



그림 13. AntHocNer 프로토콜의 PDR 성능 기준[10]
Fig. 13. PDR criteria of AntHocNet[10]

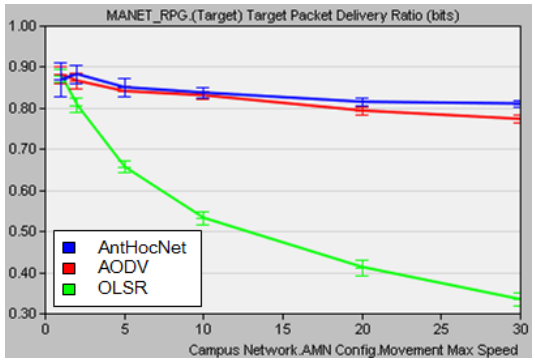



그림 14. AntHocNet 프로토콜의 구현 결과 (PDR)
Fig. 14. The PDR obtained by the implemented AntHocNet

양상을 보이는 것을 확인할 수 있다. AntHocNet의 경우 속도가 빨라질수록 성능이 떨어지는 결과를 보이고 있으나, [10]에서 제시한 결과의 실험환경 중 명시되지 않은 부분 및 다른 시뮬레이터로 실험한 결과 차이로 볼 수 있고, 라우팅 추가 프레임워크가 타당하게 설계 및 적용되었다고 판단할 수 있다.

## V. 결 론

본 논문에서는 Riverbed Modeler의 라우팅 프로토콜의 구조에 대하여 라우팅 프로토콜을 추가하기 힘든 원인을 살펴보고, 효율적이고 쉽게 라우팅 프로토콜을 추가할 수 있도록 지원하는 라우팅 추가 프레임워크를 제안하였다. 라우팅 추가 프레임워크는 IP 계층의 자식 프로세스의 형태로 라우팅 프로토콜을 구성하는 환경에서 사용가능하며, Distance Vector 방식과 Source Routing 방식 모두를 지원할 수 있도록 구

성하였다.

제안한 라우팅 추가 프레임워크를 활용하여 구현한 라우팅 프로토콜을 추가하는 경우, 매번 수정해 주어야 하는 많은 부분을 절반이하로 간소화 할 수 있었으며 여러 군데에 분산되어 있는 프로토콜 인식 부분을 파일 하나로 집중함으로써 효율성을 높이고 최소의 수정으로 라우팅 프로토콜을 쉽게 추가할 수 있다. 또한 본 논문에서는 제안한 라우팅 추가 프레임워크를 이용하여, Hybrid 라우팅 프로토콜인 AntHocNet을 구현하여 추가한 결과를 보였으며, AntHocNet 라우팅에서 제시하는 성능결과와 유사한 성능을 얻어 라우팅 추가 프레임워크가 타당하게 설계 및 반영되었음을 확인하였다.

## References


[1] K. Kim, S. H. Nam, C. W. Lee, B.-h. Roh, B. S. Roh, and M. H. Han, "Routing protocol exchanging framework for implementing new routing protocols with ease in riverbed modeler," in *Proc. KICS Winter Conf. 2016*, vol. 59, pp. 300-301, Jan. 2016.

[2] Riverbed, Riverbed Modeler, Retrieved Apr. 28. 2016. from http://kr.riverbed.com/products/performance-management-control/network-performance-management/kr-network-simulation.html

[3] 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.

[4] 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.

[5] M. Abolhasan, T. Wysocki, and E. Dutkiewicz, "A review of routing protocols for mobile ad hoc networks," *Ad Hoc Networks*, vol. 2, no. 1, pp. 1-22, 2004.

[6] T. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, *Optimized link state routing protocol*, RFC 3626, IETF, Oct. 2003.

[7] C. Perkins, E. Belding-Royer, and S. Das, *Ad hoc on-demand distance vector (AODV) routing*, RFC 3561, IETF, Jul. 2003.

[8] D. Johnson, Y. Hu, and D. Maltz, *The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4*, RFC 4728, IETF, Feb. 2007.

[9] Z. J. Haas, M. R. Pearlman, and P. Samar, *The zone routing protocol (ZRP) for ad hoc networks*, Internet Draft, IETF, Jul. 2002.

[10] F. Ducatelle, *Adaptive routing in ad hoc wireless multi-hop networks,* Ph.D. Dissertation, Università della Svizzera Italiana, Istituto Dalle Molle di Studi sull´Intelligenza Artificiale, 2007.

[11] M. Seo, J. Kim, H. Cho, S. Jung, J. Park, and T. Lee, "A study on Cross-Layer network synchronization architecture for TDMA-Based mobile Ad-Hoc networks," *J. KICS*, vol. 37, no. 8, pp. 647-656, Aug. 2012.

[12] S.-H. Lee, J.-H. Kim, K.-D. Moon, K. Lee, and J. H. Park, "Performance analysis on integrated ship area network," *J. KICS*, vol. 38, no. 3, pp. 247-253, Mar. 2013.

[13] Y. Lee and J. Kim, "Performance enhancement of AODV routing protocol using interrupt message in MANET," *J. KICS*, vol. 38, no. 10, pp. 785-800, Oct. 2013.

[14] H.-H. Choi, B. S. Roh, H. S. Choi, and J.-R. Lee, "Bio-inspired routing protocol for mobile ad hoc networks," *J. KICS*, vol. 40, no. 11, pp. 2205-2217, Nov. 2015.

[15] M. Dorigo and T. StÄutzle, *Ant Colony Optimization,* MIT Press, Cambridge, MA, 2004.

[16] D. Dhull and S. Kamra, "Application of ant colony optimization for multicasting in MANET," *Int. J. Scientific & Eng. Res.,* vol. 4, no. 1, pp 1-6, Jan. 2013.

**김 광 수** (Kwangsoo Kim)

2009년 2월 : 아주대학교 정보 및 컴퓨터공학부 (공학사)
2009월 3월~현재 : 아주대학교 대학원 컴퓨터공학과 석박사 통합과정
<관심분야> 국방전술통신망, 집단생태기반 자율기동망, 에드혹 라우팅, Modeling & Simulation, IoT 플랫폼 SW 및 서비스

**이 철 웅** (Cheol-Woong Lee)

2015년 2월 : 아주대학교 정보 컴퓨터공학과 (공학사)
2015년 3월~현재 : 아주대학교 대학원 컴퓨터공학과 석사과정
<관심분야> 네트워크 M&S, 군전술통신, 생체모방 네트워크

**신 승 훈** (Seung-hun Shin)

2000년 2월 : 아주대학교 정보 컴퓨터공학부 (공학사)
2002년 2월 : 아주대학교 정보 통신공학과 (공학석사)
2011년 2월 : 아주대학교 정보 통신공학과 (공학박사)
2011년 3월~2016년 2월 : 아주대학교 소프트웨어융합학과 강의전담교수
2016년 3월~현재 : 아주대학교 다산학부대학 조교수
<관심분야> 소프트웨어 테스트, IoT, 멀티미디어 데이터 전송

**노 병 희** (Byeong-hee Roh)

1987년 2월 : 한양대학교 전자공학과(공학사)
1989년 2월 : 한국과학기술원 전기 및 전자공학과(공학석사)
1998년 2월 : 한국과학기술원 전기 및 전자공학과(공학박사)
1989년 3월~1992년 2월 : 한국통신 통신망연구소
1998년 2월~2000년 3월 : 삼성전자
2000년 3월~현재 : 아주대학교 소프트웨어학과/대학원 컴퓨터공학과 교수
<관심분야> 이동 멀티미디어 통신 네트워킹, 트래픽 제어, IoT 플랫폼SW 및 서비스, 미래 인터넷 기술, 국방전술통신 네트워크, 네트워크 보안

**노 봉 수** (Bongsoo Roh)

2004년 2월 : 한양대학교 전자전기공학부 (공학사)
2006년 2월 : POSTECH 컴퓨터공학과 (공학석사)
2006년 4월~현재 : 국방과학연구소 연구원
<관심분야> 차세대 이동통신 시스템, 에드혹 라우팅, 저전력 통신 프로토콜, 매체접속제어, 분산자원관리, 생체모방알고리즘

**한 명 훈** (Myoung-hun Han)

2007년 2월 : 중앙대학교 컴퓨터공학과 (공학사)
2009년 8월 : 중앙대학교 컴퓨터공학과 (공학석사)
2013년 2월 : 중앙대학교 컴퓨터공학과 박사 수료
2014년 10월~현재 : 국방과학연구소 연구원
<관심분야> 이동통신, 에드혹 네트워크, 생체모방통신 등