creative commons

COMMONS DEED

저작자표시-비영리-동일조건변경허락 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.
- 이차적 저작물을 작성할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

동일조건변경허락. 귀하가 이 저작물을 개작, 변형 또는 가공했을 경우에는, 이 저작물과 동일한 이용허락조건하에서만 배포할 수 있습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

[UCI]I804:24011-200000233019

2006年 2 月
博士學位論文

안전한 유비쿼터스 서비스를 위한 MANET의 인증서 관리 및 보안 시스템에 관한 연구

朝鮮大學校 大學院

電算統計學科

李 大 榮

안전한 유비쿼터스 서비스를 위한 MANET의 인증서 관리 및 보안 시스템에 관한 연구

Security System and Certificate Management of MANET for Stable Ubiquitous Service

2006年　2月　　日

朝鮮大學校 大學院

電算統計學科

李　大　榮

안전한 유비쿼터스 서비스를 위한 MANET의 인증서 관리 및 보안 시스템에 관한 연구

指導敎授 裵 相 賢

이 論文을 博士學位申請 論文으로 提出함

2006年 2月 日

朝鮮大學校 大學院

電算統計學科

李 大 榮

李大榮의 博士學位論文을 認准함

委員長 朝鮮大學校 敎授 印

委　員 朝鮮大學校 敎授 印

委　員 木浦大學校 敎授 印

委　員 木浦大學校 敎授 印

委　員 朝鮮大學校 敎授 印

2005年　12月　　日

朝鮮大學校 大學院

목 차

목차 …… i

표 목차 …… iv

그림 목차 …… v

ABSTRACT …… viii

Ⅰ. 서론 …… 1

A. 연구배경 …… 1

B. 연구목적 …… 2

Ⅱ. *Mobile Ad-Hoc Network* …… 5

A. MANET …… 5

1. MANET의 특징 …… 5

2. Ad-Hoc 라우팅 프로토콜 …… 6

3. MANET의 형태 …… 8

4. MANET의 보안 메커니즘 …… 9

Ⅲ. *Cryptosystem* …… 13

A. Secret Sharing …… 13

B. Threshold Cryptography …… 14

1. Threshold Cryptography ……14

C. Share Refreshing ……17

D. Self-initializaion ……19

E. Cross Certification ……22

Ⅳ. 시스템의 구성 및 설계 ……25

A. 시스템의 구성 ……25

1. 시스템의 구성 ……25

2. 가정 및 접근 ……26

B. 시스템의 설계 ……27

1. 도메인 공개키와 비밀키 생성 및 구축 ……27

2. Secret Share의 분배 ……32

3. 인증서 생성 및 관리 ……33

4. 인증서 갱신 ……37

Ⅴ. 실험 및 평가 ……40

A. 실험환경 및 실험도구 ……40

1. 실험환경 ……40

2. 실험도구 NS2 ……41

B. 시스템 평가 ……44

1. 시스템 효율성 및 안정성 평가 ……44

2. 견고성 평가 ……54

Ⅴ. 결론 ……57

참고문헌 ……59

부록 Ⅰ ……63

부록 Ⅱ ……68

표 목차

【표 1】 Calculation time of certification by RSA Key bit (K=5, SPEC=20.5), SPEC=20.5≒PentiumIII/500 laptop ……45

【표 2】 Calculation time of certification by RSA Key bit (K=5, SPEC=12.1) ……46

【표 3】 Calculation time of certification by RSA Key bit (K=5, SPEC=1.37), SPEC=1.37≒SPARC station5/85 ……47

【표 4】 Certification performance in terms of system parameter K RSA key : 1024bit, time : sec ……48

【표 5】 Times of the partial certificate computation which using Lagrange interpolation operation, RSA key : 1024bit, time : sec ……49

【표 6】 Compare TCP with PCC, key : 1024bit, SPEC=20.5 ……50

【표 7】 Compare TCP with PCC, key : 1024bit, SPEC=12.1 ……51

【표 8】 Compare TCP with PCC, key : 1024bit, SPEC=1.37 ……52

그림 목차

【그림 1】 Ad-Hoc Network Routing Protocl ……7

【그림 2】 Single Mobile Ad-Hoc ……8

【그림 3】 Ad-Hoc Network와 기존 망의 연동 ……9

【그림 4】 service of key management ……15

【그림 5】 Threshold signature K/k ……16

【그림 6】 Illustrate share refreshing process ……19

【그림 7】 Localized Self-initialization service ……20

【그림 8】 Complete shuffling scheme use random nonces ……21

【그림 9】 Cross Certificaton between two CAs, CA-A and CA-B ……23

【그림 10】 System Accomplishment process ……26

【그림 11】 formation of cluster ……28

【그림 12】 creation and distribution of public key ……29

【그림 13】 An exchange of secret shares used Cross Certification Ⅰ ……30

【그림 14】 An exchange of secret shares used Cross Certification Ⅱ ……30

【그림 15】 transmission of secret share key ……31

【그림 16】 creation of partial certification key ……32

【그림 17】 Sharing secret share ……33

【그림 18】 Distribute Secret Share ……33

【그림 19】 Creation of Complete Certificate ……35

【그림 20】 Topology changes in ad hoc network ……37

【그림 21】 The joining of new node I ……39

【그림 22】 Architectural View of NS ……41

【그림 23】 Class hierarchy of NS2 ……42

【그림 24】 NS simulation script ……42

【그림 25】 Execution of simulation using NAM ……43

【그림 26】 Tracking of packet trace ……43

【그림 27】 Format of trace file ……44

【그림 28】 Calculation time of certification by RSA Key bit (K=5, SPEC=20.5) ……45

【그림 29】 Calculation time of certification by RSA Key bit (K=5, SPEC=12.1) ……46

【그림 30】 Calculation time of certification by RSA Key bit (K=5, SPEC=1.37) ……47

【그림 31】 Partial Certification performance in terms of system parameter K ……48

【그림 32】 Certification performance in terms of system Parameter K ……49

【그림 33】 Compare TPC with PCC, Key : 1024bit, SPEC=20.5 ……50

【그림 34】 Compare TPC with PCC, key : 1024bit, SPEC=12.1 ……51

【그림 35】 Compare TPC with PCC key : 1024bit, SPEC=1.37 ……52

【그림 36】 Format of trace file ……54

【그림 37】 Avg. rate of Key agreement (node K=5, 10, 20) ……55

【그림 38】 Avg. rate of Key agreement (node K=30, 40) ……55

ABSTRACT

Security System and Certificate Management of MANET for Stable Ubiquitous Service

BY Lee, Dae-Young
Advisor : Prof. Bae, Sang-Hyun.
Dept. of Computer Science and Statistics,
Graduate School of Chosun University

MANET(Mobile Ad-Hoc Network) consists of mobile nodes that transfer packet data using wireless interface. MANET is a network that can transmit and receive data through routing mechanism between nodes without infrastructure, rather than communicating under the environment equipped with infrastructures. Therefore, each node participating in network should perform as router and servers without help from base stations or AP and can transmit packets instead of other nodes and perform user applications. The most attractive thing of MANET is that users can have more freedom and flexibility in use of networks, completely out of central control. In the beginning, MANET was a military communication system and recently, it has been expanded to various areas such as WPAN(Wireless- Personal Area Network), Ubiquitous, Bluetooth, RFID(Radio Frequency IDentification), and sensor networks. Mark Weiser working for Palo Alto Research Center in Xerox of U.S.A. considers Ubiquitous part of our life, rather than serious technology. Through ubiquitous computing for which computers are invisibly

planted in daily life and things, information can be exchanged and we can use ubiquitous computers anytime and anywhere. That is, ubiquitous computing is our daily environment in which we can not recognize the existence of computers they are incorporated into our daily life though computers are installed in our physical space consisting of things and environments. For existing security, information digitalized in cyber space or physical space and stored in computers was a problem. In consideration of properties of ubiquitous networks, a problem that personal information may be exposed exists in ubiquitous network. For existing networks, the place attacked was confined to personal computers, but in ubiquitous networks, all the personal space may be revealed. Therefore, cyber terror in ubiquitous network includes terror given to physical space, things and physical bodies and extensive space protection is required beyond protection of personal, business and national information

In respect to security at MANET, a central technology in ubiquitous network, nodes move frequently or through mobile transfer between nodes, MANET is independently managed. So it is thought that management of its security is more difficult than that of existing networks.

Many routing protocols and security mechanisms of MANET were suggested, but it is judged that existing design methods can not meet the needs of MANET security completely, and a new concept about it should be prepared. At present, many institutions and organizations are studying on it. As MANET has dynamic composition in which nodes are frequently changing, when a node is damaged, reliability between nodes may be changed and they may join in other administrative domains quickly they can rely on or secede. However, MANET has been studied for military or natural disaster communication system because of its property that it can perform

communication without infrastructures. Such communication networks are appropriate because it is quickly-developed base structure and allows only optional and confined access.

Mobile networks can use various user certificate mechanisms such as EAP-TLS, LEAP and PEAP. As such user authorization mechanisms use public key infrastructure, PKI in mobile network should be prepared. The PKI mechanism is due to safety of PKI and through this mechanism, safety of electronic signature ca be secured. The most serious problem in using PKI in mobile personal network is that it can not have reliable certificate because of no trusted Certificate Authority. In general, key management on networks is performed through CA (Certification Authority) or Key Distribution Server. However, due to absence of trusted certificate authority on mobile networks, direct application of PKI has some problems and actually, certificate on mobile networks is device certificate, not user certificate. Also, for safe use of devices, symmetric key system in MAC is used according to security needs, but safe exchanges of keys are not mentioned.

This study suggests security requirements for MANET which can be a base of ubiquitous system and models that can prevent security threat through application of PKI without trusted certificate authority in MANET. It is also solve excessive loading found in centralized control model by dispersing CA for adjustment to dynamic changes of nodes of MANET quickly and suggests a system model which supports expansion so that existing nodes performing communication within clusters can provide active certificate service without being affected by input of new nodes. In addition, it is to evaluate its stability, effectiveness and strength through simulation of the suggested model.

Ⅰ. 서 론

A. 연구배경

MANET(Mobile Ad-Hoc Network)은 무선 인터페이스를 사용해 패킷 데이터를 전송하는 무선 노드로 구성된 네트워크이다. MANET은 기존의 네트워크와 같이 인프라가 구축된 상태에서 통신을 수행하는 것이 아닌 인프라가 존재하지 않은 상태에서 각 노드들 상호간의 라우팅으로 데이터 송・수신 등의 통신 기능을 수행할 수 있는 형태의 네트워크를 말한다[1]. 따라서 네트워크에 참여하는 각 노드들은 기지국이나 AP의 도움 없이 자신들이 라우터, 서버의 역할 등을 모두 수행하면서 다른 노드 대신 패킷을 전송할 수도 있고, 사용자 애플리케이션들을 실행할 수도 있어야 한다. MANET의 가장 매력적인 점은, 중앙통제로부터 완전히 독립해 사용자가 네트워크 사용에 더 많은 자유와 유연성을 얻게 된다는 것이다. 초기에 군사용 통신체계로 출발한 MANET은 최근에 WPAN(Wireless-Personal Area Network), Ubiquitous, Bluetooth, RFID(Radio Frequency IDentification, 전파식별), 센서 네트워크 등 더욱 다양한 분야로 응용이 확대되고 있다.

미국 Xerox사의 Palo Alto Research Center의 Mark Weiser는 Ubiquitous(유비쿼터스)를 심오한 기술로 보는 것이 아니라, 일상 생활속에 일부분으로 보고 있다[2]. 유비쿼터스 컴퓨팅은 일상 환경과 사물들에 컴퓨터가 보이지 않게 심어져 있어 서로간의 정보 교환이 가능하며, 언제, 어디서나, 존재하는 즉 편재되어 있는 컴퓨터를 편리하게 사용할 수 있는 컴퓨팅 환경이다. 즉, 유비쿼터스 컴퓨팅은 사물과 환경으로 구성되는 물리적인 공간에 컴퓨터가 스며들어가고 이러한 컴퓨터가 도처에 편재되어 일상생활에 통합되어 한 사람에 의하여 사용되는 수많은 컴퓨터로 컴퓨터의 존재가 인식되지 않는 말 그대로 일상 환경이 되는 것이다. 기존의 보안은 사이버 공간, 물리적인 공간이 디지털화 되어 컴퓨터에 저장된 정

보들이 문제가 되었지만, 유비쿼터스 네트워크 특성을 고려해 볼 때 유비쿼터스 네트워크에서는 개인의 모든 정보가 노출 될 수 있다는 security상의 문제점을 안고 있다. 또한 기존의 네트워크에서는 침입하는 장소가 개인의 컴퓨터로 한정되는 반면, 유비쿼터스 네트워크에서는 개인의 사적인 모든 공간이 노출되는 셈이 되는 것이다. 유비쿼터스 네트워크에서의 사이버 테러는 물리공간과 사물, 그리고 육체에 대한 테러를 포함하게 되며, 개인이나 기업과 국가의 정보보호를 뛰어 넘어 광범위한 공간 보호가 요구된다[3].

유비쿼터스의 핵심 기술인 MANET에서의 security는 각 노드들이 수시로 이동하거나, 모든 노드들 간의 무선 전송으로 다른 노드와는 독립적으로 운영되는 등 MANET의 특성 등으로 인하여 기존 네트워크보다 security 대책이 쉽지 않을 것으로 인식되고 있다[4][5][6].

많은 Routing Protocol 및 Security Mechanism이 제안되었지만 기존의 설계 방식으로는 완벽한 MANET의 보안 대책이 되기는 어려울 것으로 판단되며 새로운 개념의 검토가 이루어져야 할 것이며 현재, 많은 기관과 단체에서 연구 중이다[7][8][9][10].

B. 연구목적

MANET은 멤버(node)들이 수시로 변하는 동적인 구성을 가지고 있기 때문에 어떤 멤버가 훼손 되었을 경우, 멤버들의 신뢰관계도 변하게 되며, 신속하게 신뢰할 수 있는 다른 관리 도메인(Administrative Domain)에 가입하고 탈퇴하게 되는 다이나믹한 특성을 가지고 있다. 반면 MANET은 인프라의 구성없이도 통신을 할 수 있다는 특성에 힘입어 주로 상용 시스템이 아닌 군사용 통신체계로의 연구가 이루어져 왔다. 군사용 통신망은 신속히 전개되는 기반구조이며, 제한되고 선택적인 접근만을 허용하기 때문에 이러한 형태에 적합하다.

무선 네트워크에서는 EAP-TLS, LEAP, PEAP 등의 다양한 사용자 인증 메커

니즘의 사용이 가능하다. 이러한 사용자 인증 메커니즘은 모두 공개키 암호 기술을 이용하고 있어 무선 네트워크 환경에서의 PKI(공개키 기반 암호화 기법,Public Key Infrastructure) 구축이 요구된다. PKI 메커니즘은 그 근본이 되는 공개키 암호의 안전성에 기인하며 이를 통해 전자 서명의 안전성을 보장 받을 수 있게 된다. 무선화 개인 네트워크에서 공개키 기반 암호화 기법을 사용하려고 할 때 가장 문제가 되는 점은 신뢰된 인증기관(Trusted Certificate Authority)의 부재로 인한 신뢰된 인증서(Certificate)를 가질 수 없다는 점이다. 일반적으로 네트워크 상에서의 키 관리는 인증기관(CA, Certification Authority)이나 신뢰 할만한 키 분배 서버(Key Distribution Server)를 통하여 이루어진다. 하지만 무선화 네트워크상의 신뢰된 인증기관의 부재로 인해 공개키 기반 암호화 기법을 바로 적용 하는 데에는 문제가 있으며, 실제로 무선 네트워크 상에서의 인증은 사용자 인증이 아닌 디바이스 인증에 머물고 있는 실정이다. 또한 장치들의 안전한 사용을 위해 보안 요구에 따라 MAC 계층에서의 대칭키 암호 시스템을 채택하고 있지만, 키를 안전하게 교환하는 방법에 대해서는 언급하고 있지 않은 취약점을 안고 있다.

본 연구에서는 MANET 환경에서 발생할 수 있는 보안 요구사항을 도출하고, MANET에서 신뢰된 인증기관 없이 공개키 기반 암호화 기법을 사용하는 기법을 적용하여 보안 위협을 방지할 수 있는 모델을 제안 하도록 한다. 또한 MANET을 구성하는 멤버 또는 노드들의 동적인 변화에 즉각적으로 적응할 수 있도록 CA를 분산시켜 중앙 집중형의 관리 모델에서 발생되는 부하 집중 문제를 해결하고, 클러스터 내에 통신하고 있는 기존의 노드들이 새로운 노드의 유입에 영향을 받지 않고 능동적으로 인증 서비스를 수행할 수 있도록 높은 확장성을 지원하는데 목적이 있다.

본 논문의 구성은 다음과 같다.

먼저 2장에서는 MANET의 특징과 라우팅 프로토콜과 형태를, 그리고 기존의 보안 메커니즘에 대하여 기술하고 문제점을 도출해 본다. 3장에서는 본 논문에서 제안하고자 하는 시스템을 유도해 내기 위한 관련 기술 및 관련 연구에 대하여

기술한다. 4장에서는 제안한 시스템의 구성 및 설계를 기술한다. 5장에서는 본 논문에서 제안한 시스템을 시뮬레이션을 통해 안정성 및 효율성과 견고성을 평가해 보도록 한다.

Ⅱ. *Mobile Ad-Hoc Network*

A. MANET(Mobile Ad Hoc Network)

*1. MANET*의 특징

MANET은 기존의 네트워크와 같이 네트워크 인프라가 구축된 상태에서 통신을 수행하는 것이 아닌 인프라가 존재하지 않은 상태에서, 즉 신뢰된 인증기관(Trusted Certificate Authority), 중앙 관리(central administration) 없이 형성된 구조의 네트워크를 뜻한다. 이러한 MANET의 특징은 다음과 같이 요약된다.

우선 MANET의 가장 큰 특징은 네트워크에 참여하는 노드들이 가지는 이동성을 들 수 있다. 이러한 유형의 네트워크에 있는 노드들은 라우터나 호스트로 이용될 수 있기 때문에 각 노드들 상호간의 라우팅으로 데이터 송・수신 등의 통신 기능을 수행할 수 있고, 사용자 애플리케이션들을 실행할 수도 있어야 한다. 따라서 네트워크에 참여하는 각 노드들은 기지국이나 AP의 도움 없이 자신들이 라우터, 서버의 역할 등을 모두 수행할 수 있어야 한다[11].

두 번째, MANET에 참여하는 노드 일부나 전체가 무선이기 때문에 MANET은 다이나믹한 네트워크 토폴로지를 가지게 된다. 노드들은 네트워크에 상에서의 진입과 이탈이 자유롭고, 이로 인해 다양한 형태의 네트워크가 형성된다. 이러한 다양한 형태의 동적인 네트워크가 붕괴 되지 않고 연결이 유지되어 애플리케이션과 서비스를 차질 없이 제공할 수 있도록 상태를 유지하여야 한다[11][12].

세 번째로 MANET의 경우, 노드는 보안 및 라우팅 기능 지원을 백그라운드 네트워크에 의존할 수 없다. 그 대신, 이들 기능이 설계되어 분산된 조건 하에서도 효율적으로 운영될 수 있어야 한다. 따라서 하나의 노드가 많은 기능을 선점하고 있는 것이 아닌, 여러 노드들이 제공하는 서비스를 분배하고 협력하여 제공하는

구조를 지녀야 한다

마지막으로 MANET은 불안정한 링크 특성을 가지게 된다. 이것은 무선을 사용함으로써 생기는 문제로, 데이터에 대한 전송거리와 전송대역폭에 제약을 받고, 전파간섭/다중링크로 인한 보안 문제를 야기한다는 의미가 된다.

이러한 특징을 가지는 MANET의 경우 인프라의 구성없이도 통신을 할 수 있다는 특성에 힘입어 주로 상용 시스템이 아닌 군사용 통신체계로의 연구가 이루어져 왔다. 군사용 통신망은 신속히 전개되는 기반구조이며, 제한되고 선택적인 접근만을 허용하기 때문에 이러한 형태에 적합하다.

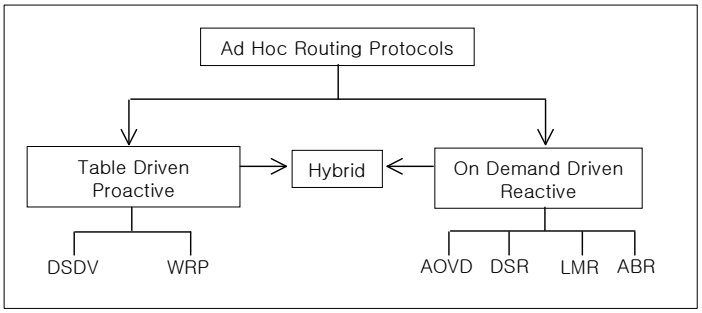
2. Ad Hoc 라우팅 프로토콜

Ad Hoc 네트워크에서 사용되는 라우팅 프로토콜은 미리 라우팅 정보를 수집해두어 사용하는 테이블 관리 방식(Table-Driven 혹은 Proactive)과 라우팅 정보를 필요한 시기에 수집하는 요구 기반 방식(On-Demand 혹은 Reactive), 마지막으로 테이블 관리 방식과 요구기반 방식을 혼합한 하이브리드 방식으로 크게 나눌 수 있다.

Reactive 방식은 연결 요청이 있을 때 해당 목적지로의 경로를 on-demand로 설정해주며, 연결 요청 시 마다 경로 발견을 하기 위한 flooding 오버헤드가 있으며 경로 발견을 위한 지연 시간이 커지는 문제가 있다[13]. 반면, Reactive 방식은 MANET 노드의 이동성을 즉시로 반영한다는 점에서 보다 정확한 경로 설정이 가능하다는 장점이 있다[14][15].

proactive 방식은 주기적으로 이동 노드들 간의 경로를 설정해준다. 이동 노드가 주기적으로 네트워크 상황을 다른 이동 노드들에게 flooding 해서 알려줌으로써 네트워크 정보를 최신으로 유지하는 방법을 사용하나, 이로 인한 flooding 오버헤드가 상당히 큰 단점이 있다. 따라서 proactive 방식을 사용하는 라우팅 프로토콜에서는 flooding 오버헤드를 줄이기 위한 방법들을 사용하고 있다. 반면

proactive 방식의 경우 연결 요청이 있을 때 이미 해당 경로 정보를 갖고 있으므로 연결 설정이 빨라지는 장점이 있다. 따라서 네트워크 규모가 작은 경우는 proactive 방식이 적합하고 네트워크 규모가 큰 경우는 reactive 방식이 적합하다고 할 수 있다. Hybrid 방식은 Table-Driven과 On-Demand 방식의 장점을 혼합한 방식으로 각 노드가 미리 정의된 범위의 구역을 유지하여 구역 안에서 Table-Driven 방식을 사용하고, 외부영역에서는 On-Demand 방식을 사용하여 경로를 설정하게 된다[16].



【그림 1】Ad Hoc Network Routing Protocol

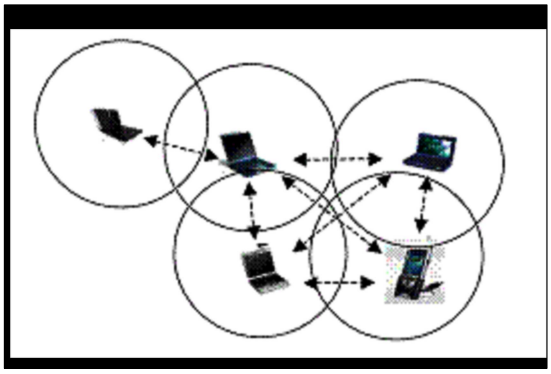
IETF는 유선 Infrastructure의 도움 없이 무선 노드들 간의 통신이 가능하게 하는 MANET과 관련된 표준화 작업을 하는 워킹 그룹으로 지금까지 MANET을 위한 유니캐스트 라우팅 프로토콜에 대한 표준화 작업을 추진해 오고 있다. 현재 AODV(Ad hoc On-Demand Distance Vector, RFC 3561) , OLSR(Optimized Link State Routing, RFC 3626), TBRPF(Topology Dissemination Based on Reverse-Path Forwarding), DSR(Dynamic Source Routing)과 관련된 문서를 experimental RFC(Request For Comments)로 만드는 노력을 기울여 왔으며, 이 프로토콜들은 experimental RFC로 채택되어 있는 상태이다[17].

MANET 워킹그룹의 목적은 동적, 정적 토폴로지 내에서 무선 라우팅 응용에 적합한 IP 라우팅 프로토콜 기능을 표준화하는 것으로, 여기에서의 문제는 무선 라우팅 인터페이스 특유의 특성과 노드 이동성으로 인해 라우팅이 어렵다는 것이다. MANET 워킹그룹에서는 MANET을 IP 인프라스트럭쳐의 끝단에서 라우팅 기술을 확장하는 용도로 사용하는 것을 주요 관심사항으로 하며, 또한 고정

노드와 이동 노드가 혼재하고 라우팅 기능이 없는 노드들도 존재하는 하이브리드 형태의 인프라스트럭처에 대해서도 지원할 예정이다. 또한 향후 reactive 방식과 proactive 방식을 통합하는 것과, generic flooding과 관련된 표준화를 추진할 것으로 예상된다. Generic flooding은 모든 MANET 노드들의 포워딩 프로토콜 빌딩 블록으로 고려되고 있으며, 지금까지 다수의 flooding 최적화 방법들이 제시되었으며, 이들 방법은 상당수의 중복 전송을 줄이는 것을 목표로 한다. 이렇게 제안된 generic flooding 방법은 임의의 라우팅 프로토콜에서 사용될 수 있고 또한 address auto configuration, multicast 등에서도 사용될 수 있을 것으로 예상된다.

*3. MANET*의 형태

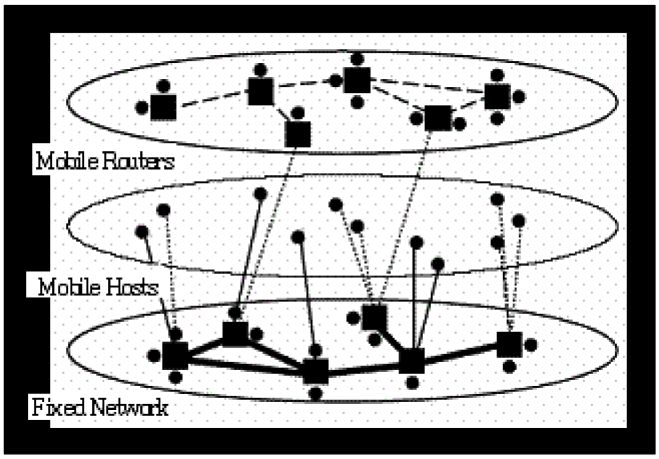
Ad Hoc 네트워크의 경우, 이동 단말들로만 구성된 단일 형태의 독립적인 망 구성이 가능할 뿐만 아니라 기존의 인터넷 망 등과의 연동 역시 가능하다.



【그림 2】Single Mobile Ad-Hoc

이러한 기존 망과의 연동을 통해서 Ad Hoc 네트워크는 임시적 망을 구축하여 상호간의 통신을 할 수 있다는 장점을 지니고 기존 망과의 연동을 통한 데이터의 송수신이 가능해지므로 기존 망에서 제공되는 서비스와 콘텐츠 서비스를 그대로

물려받을 수 있으며, 여러 곳에 분산된 장비를 이용하여 데이터를 가공/처리하여 정보의 가치를 높을 수 있다는 장점을 가지게 될 수 있는 것이다.



【그림 3】Ad Hoc 네트워크와 기존 망의 연동

4. MANET 보안 메커니즘

보안 문제는 유선 네트워크에서도 존재하고 있지만, MANET은 무선 인터페이스를 사용하기 때문에 유선 네트워크에 비해 훨씬 더 많은 위험에 노출되어 있다. 그러므로, 기본적인 Ad Hoc 네트워크의 보안 요구조건은 다른 통신 네트워크에서 요구되는 것과 동일하지만, MANET에서는 노드가 신뢰받은 인증기관을 통해 인증을 받는 형식이 아니기 때문에, 멀티홉 방식에 의해 라우팅을 수행할 경우 악의적인 중간 노드에 의해 데이터의 무결성 및 기밀성 문제가 발생할 수 있다. 특히, 매체를 신뢰할 수 없는 상황에서 암호를 사용하므로, 암호키에 크게 의존하게 된다. 또한, MANET 환경은 모든 노드들이 분산되어 있고, 어떠한 고정된 기반구조도 없으며, 모든 노드가 공평하게 역할을 나누어 갖는다는 특징을 갖는다. 한편으로, 보안 문제가 확실히 해결되다보면, 컴퓨팅 문제가 발생되어, 노드와 네트워크 전체에 심각한 부하를 주게 되므로, 이동 Ad Hoc 네트워크에

적합하게 구현된 알고리즘, 키 분배 및 인증 프로토콜의 개발이 현실적으로 가장 필요하다. 즉, 키 사이에 신뢰할 수 있는 관계를 형성하고, 이를 이동 Ad Hoc 네트워크 전반에 분배하는 것이 주요 관심 사항이라 할 수 있다.

Ad Hoc 환경에서 중요하게 고려되는 위협은 크게 외부 위협과 내부 위협으로 구분할 수 있다. 외부 위협은 다시 잘못된 라우팅 정보의 삽입을 통한 위협과 이전의 라우팅 정보를 재생하여 악용하는 위협, 라우팅 정보를 변형하여 네트워크에 위험을 가져오는 위협 등으로 분류할 수 있다. 외부 위협을 통해 공격자는 네트워크를 분할하거나 네트워크에 극심한 트래픽을 유발하여, 전체 네트워크 시스템에 장애를 일으키는 결과를 초래할 수 있다. 이러한 위협을 막는 것은 굉장히 어려운 일인데, 그 이유는 라우팅 정보를 통해 공격자에 의해 훼손된 노드를 찾는 것이 매우 어렵기 때문이다. 다시 말해서, 라우팅 정보의 한 부분이 유효하지 않다고 발견된 경우, 그 정보가 공격자와 타협한 노드에 의해서 만들어진 것인지 아니면, 토폴로지 변화의 결과로 유효하지 않게 되는 경우인지를 구분해내는 것이 매우 어렵다는 의미이다. 이러한 위협에 대처할 수 있는 효과적인 방법은 충분한 노드를 확보하여 공격자들을 우회할 수 있는 경로를 찾는 것이다. 두 번째는 내부 위협으로, 이는 외부 위협보다 훨씬 더 심각한 위협이며, 시스템 내의 훼손된 노드들에게서 발생하는데, 이 경우 어떤 노드들은 다른 노드들에게 잘못된 정보를 제공하여 시스템에 장애를 유발한다. 이를 막기 위한 방법으로써, 올바른 노드들이 충분히 존재하기만 하면, 라우팅 프로토콜은 훼손된 노드들 주변을 우회할 수 있는 경로를 찾을 수 있을 것이다. 따라서 Ad Hoc 보안에서 중요한 연구 과제는 먼저, Ad Hoc 네트워크에서는 무선 연결을 사용하기 때문에 공격자로 하여금 가장(Impersonation), 메시지 재연(Message Replay), 메시지 변형(Distortion) 등과 같은 수동적 공격을 가능하게 하므로, 이에 대한 연구가 필요하다. 또한, Ad Hoc 환경은 상대적으로 물리적 보호에 취약하다. 즉, 노드들은 항상 공격자가 존재하는 환경에서 움직이므로, 언제든지 공격자에 의해 중요한 정보가 훼손될 가능성을 무시하면 안된다. 한편, MANET에서는 네트워크 외부에서의 공격자뿐만 아니라, 네트워크 내부의 공격자와 타협한 노드들에게서 시작되는 공

격 또한 간과해서는 안된다. 그러므로, MANET에서 보다 높은 생존성을 보장하기 위해, 어떤 'Central Entities'도 없는 분산된 구조를 지녀야 한다. 그 이유는 중앙의 관리자(Authority)가 공격자와 타협하면 모든 정보가 노출되고, 다양한 공격이 가능하기 때문이다.

한편, Ad Hoc 네트워크는 토폴로지와 구성원이 수시로 변하는 동적인 구성을 갖고 있어서 어떤 노드가 훼손되었을 때, 노드들의 신뢰 관계도 변하게 되어 다른 무선 네트워크(예, Mobile IP)와는 다르게 Ad Hoc 노드들이 동적으로 관리 도메인(Administrative Domains)에 가입하고 탈퇴한다. 결론적으로, 정적 구성에서의 보안 해결책으로는 충분하지 못하며, 빈번한 변화에 즉각적으로 적응할 수 있는 시스템이 필요하다. 마지막으로, Ad Hoc 네트워크는 수백 내지 수천 개의 노드들로 구성될 수 있으므로, 보안 구조는 거대한 규모의 네트워크를 모두 포함할 수 있는 구조로 연구되어야 한다.

마지막으로 MANET의 보안 목표는 여러 가지가 존재하지만, 대체로 다섯 가지 정도로 말할 수 있다[18].

첫번째, 가용성(Availability) 측면인데, 이는 공격자에 의한 DOS 공격에도 불구하고, 네트워크 서비스의 붕괴가 초래하지 않아야함을 의미한다. 여기서 DOS 공격은 Ad Hoc 네트워크의 어떤 계층을 통해서도 도달할 수 있다고 보며, 물리 계층과 미디어 접근제어 계층에서 작은 Jamming을 통해 물리적인 채널을 통한 통신을 간섭할 수 있다고 가정하면, 네트워크 계층의 라우팅 프로토콜들을 붕괴시키고, 네트워크 연결을 끊을 수 있다. 상위 계층에서도 다양한 서비스들에 영향을 미칠 수 있으므로, 키 관리 서비스 측면에서 가용성은 중요한 목표가 된다.

두 번째, 노드들 간에 전달되는 데이터들은 인증되지 않은 노드들에게는 노출되지 않아야함을 보장하는 것으로 '신뢰성(Confidentiality)'이라 한다. 이는 특히 군사적인 부분에서 많이 요구된다.

세 번째, 전송된 메시지가 훼손되지 않았음을 보장하는 무결성(Integrity)이다.

네 번째, 인증(Authentication)으로서, 이는 노드로 하여금 통신에 관여하고 있는 상대 노드의 신원을 확신하게 하는 것을 의미한다.

마지막으로, 부인방지(Non-Repudiation)가 요구되는데, 이는 메시지를 보낸 곳에서 메시지를 보낸 사실을 부정하지 못하도록 하는 것이다.

이외에도 ordering, timeliness, isolation, lightweight computations, location privacy, Byzantine robustness, key management 등의 보안 목표가 있다.

Ⅲ. *Cryptosystem*

A. Secret Sharing

secret sharing은 그룹에 관계하는 참가자들에게 비밀키를 할당하고, 비밀키의 비율을 미리 정하여 알고 있는 참가자들로 접근을 한정하는 기술이다. secret sharing scheme은 1개의 dealer와 n개의 player로 구성되며, dealer는 player에게 명백한 조건을 충족시켰을 경우 비밀키를 할당하게 된다. dealer는 각 player에게 t 또는 t 이상의 player 그룹에게 share를 할당하고 함께 비밀키를 재건할 수 있는 조건을 충족 시킨다. 그러나 t 보다 작은 player 그룹은 비밀키를 재건할 수 없다. 이러한 Cryptosystem을 (t,n) Threshold 혹은 (n,t) Threshold scheme이라 한다. 이 Cryptosystem에서 송신자가 수신자에게 메시지를 전송하고자 하는 경우는 다음과 같다. 이 Cryptosystem에서는 하나의 공개키와 n개의 할당된 비밀키 쌍을 이용하는데, 그룹의 공개키를 가지고 원문을 암호화 하여 전송하면, 수신자는 각자 할당된 비밀키를 신뢰할 수 있는 노드(trusted node)에 안전한 채널을 통해 전송한다. 트러스트 내에서 t+1 이상의 비밀키가 모아지면 이를 통해 원문을 각 수신자에게 전송하게 된다. 다음은 simplest secret sharing이다.

Simplest sharing scheme is following
step 1. generate a random-bit string, R, the same length as the key K
step 2. XORs K with R to generate s. K⊕R=S
step 3. gives R to Alice and S to Bob
to reconstruct key, Alice and Bob have only on step to do:
step 4. Alice and Bob XOR their pieces together to reconstruct the key R⊕S=K

예를 들어 비밀“password″를 기존의 비밀키를 되찾기 위해 요구 되어지는 네 개의 비밀키 share ”pa------″, ″--ss----″, ″----wo--“, ″------rd″를 분배하는 딜러가 있다고 하자. share를 가지고 있지 않는 사람은 password가 8자의 글자로 구성되어 있다는 것을 알지 못 할뿐만 아니라, password에 대해서는 조금의 관심도 없다. 또한 26^8의 가능한 조합으로 password를 추측해야 할 것이다. 하지만 1개의 share를 알고 있다면, 6개의 글자 26^6의 조합만으로 password의 추측이 가능하게 된다. 만일 3개의 share를 알고 있다면 676번만으로 password의 추측이 가능하다. 이 시스템은 각 share의 비밀키의 내용 대한 중요한 정보를 제공하기 때문에 실제로는 secret sharing이 아니다.
몇몇의 secret share는 효율을 개선하기 위한 절대적인 안전을 단념하고, 다른 평범한 암호가 충분이 안전하다고 고려되고 충분한 안전을 유지하는 동안 이론적으로 안전한 정보라 할 수 있고, 증명 되어질 수 있다.

B. Threshold Cryptography

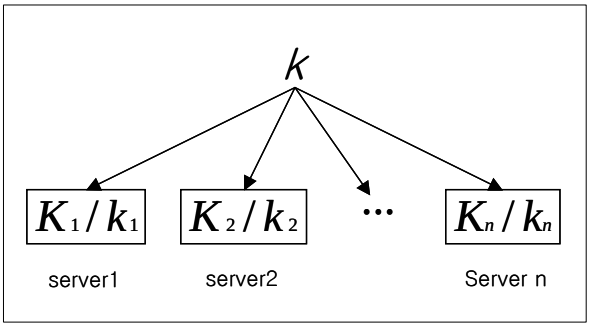
보안시스템에서 공개키 암호화 체계를 운용함에 있어 가장 중요한 문제는 각 사용자의 공개키를 다른 사람들이 신뢰할 수 있도록 보장하기 위한 공개키를 만드는 것이다. Mobile Ad Hoc 네트워크에서 공개키 기반 암호화 기법을 적용하고자 할 때 가장 문제가 되는 것은 신뢰된 인증기관이 없다는 것이다. 이를 위해 현재 공개키 측면에서 연구된 이론으로 Threshold Cryptography가 있고 이를 보다 안정하게 하기위해 Share Refreshing을 사용하고 있다[19][20][21].

1. Threshold Cryptography

Shamir가 비밀 공유기법[22]을 제안한 이후 Threshold Cryptosystem이 발전하

였다[23].

$An(n,t+1)$ Threshold Cryptography scheme은 n 노드들이 암호의 가동을 실행하기 위한 능력을 공유하는 것을(예를 들면 디지털 서명을 만든다) 허락하며, $t+1$ 노드들 이여야만이 연대적으로 가동을 실행할 수 있다. t 노드들로만 실행하는 것은 불가능하다. 키 관리 서비스의 n 서버들은 증명서들에 서명하는 능력을 공유한다. 관대한 서비스를 위해 t는 서버들과 타협하여 해결 짓고, $(n,t+1)$ Threshold Cryptography scheme을 사용하여, n 노드들에서 서비스를 위한 비밀키 k를 $s_1, s_2, ..., s_n$ 몫으로 나누고, 각각의 서버에 하나의 몫을 할당한다. 우리는 $s_1, s_2, ..., s_n$을 k의 sharing $(n,t+1)$이라 한다. 【그림 4】는 서비스가 형성되는 방법을 도시한다.



【그림 4】Service of key management

Service of key management

○ Key management service consists of n servers

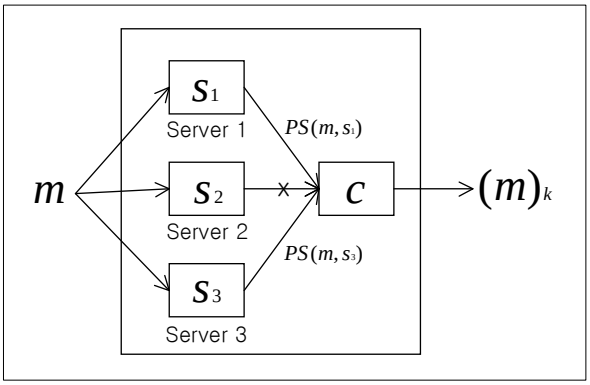
○ Service has a public/private key pair K/k

○ Public key K is known to all nodes in the network

○ private key k is divided into n shares $s_1, s_2, \cdots, s_n$

○ each server i has a public/private key pair K/k , and all nodes known to public keys

증명서에 사인하기 위한 서비스를 위해 각각의 서버는 비밀 키를 나눠 사용한 증명서를 위해 일부분의 서명을 형성하고, combiner에게 일부분의 서명을 제출한다. $t+1$개의 정확한 일부분의 서명들로, combiner는 증명서를 위한 서명을 계산할 수 있다. 그러나, 타협하여 해결 짓게 된 서버들(서버들 중 많아야 t개)은 많아야 t개의 일부분 서명들을 생성할 수 있기 때문에 그들 자신에 의해 바르게 서명 받았던 증명서들을 생성할 수 없다. 【그림 5】는 Threshold signature 기법을 이용하여 서버가 어떻게 서명을 생성하는가를 보여 준다.



【그림 5】Threshold signature K/k

Threshold signature scheme, 【figure 5】

- ○ Service of three server which has a public/private key pair K/k
- ○ Using [3,2]Threshold Cryptography scheme, each server I has private key share s_i
- ○ Server I can create partial certificate $PS(m, s_i)$, using private key share s
- ○ Server 1 and 3 creates partial certificate, and transmit certificate to combiner
- ○ c can create signature $(m)_k$ with server's private key k

Threshold cryptography를 적용할 때 우리는 타협하여 해결 지었던 서버들에 대하여 방어해야 한다. 예를 들어 타협하여 해결 짓게 되었던 서버는 부정확한 일부분의 서명들을 생성할 수 있었다. 이 일부분의 서명의 사용은 무효한 서명을 낳을 것이다. 다행히도 combiner는 공개 암호키를 서비스를 이용하여 계산된 서명의 정당성을 검사할 수 있었다. 검증에 실패하는 경우, combiner는 $t+1$ 일부분의 서명들의 다른 하나의 세트를 시험해 본다. 이 처리는 combiner $t+1$의 정확한 서명들의 일부분으로부터 완벽한 서명을 구성 할 때 까지 계속된다. 좀 더 효율적이고 강력한 combining 기술들이 제안되었다[24][25].

그 기술들은 일부분의 서명들 안에서 고유의 중복들을 개척하고(어떤 $t+1$의 정확한 일부분의 서명이라도 최종적인 서명의 모든 정보를 포함함을 주목해야 한다.), 부정확한 일부분의 서명들을 가리기 위해 에러 정정코드를 사용한다.

다음 [24]에서, 강력한 Threshold DSS(Digital Signature Standard) 기술이 제안되었다.

일부분의 서명으로부터 서명을 계산하는 처리는 본질적 보간법(補間法)이다. 보간법은 찾아낼 수 없거나 부정확한 일부분의 서명의 조그만 부분을 가지고 정확한 서명을 생산해 낼 수 있기 때문에 저자들은 Berlekamp, Welch의 디코더를 사용한다.

C. Share Refreshing

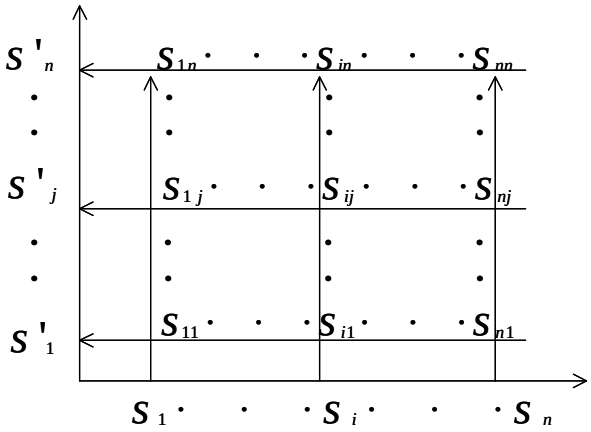
이동형 공격자는 일시적으로 한 서버를 공격하고, 바로 이어서 또 다른 희생자를 찾아 이동하는 특징을 가진 노드로써, 네트워크 내에 바이러스 같은 대상을 의미한다. 이러한 공격자는 오랜 기간동안 모든 서버들을 공격할 수가 있다. 비록 서버들이 공격자의 공격을 탐지하고 배제하였다 하더라도, 그 공격자는 공격되었던 서버들로부터 여전히 비밀키 t share 보다 더 많이 모을 수 있을 것이다 [26][27][28]. 공격자는 이러한 과정을 통해 얻은 비밀키로부터 유용한 인증서를

생성해 낼 수 있을 것이다. share refreshing은 이동형 공격자에 대응하기 위해 제안되었다[26][27][28][29][30].

Threshold cryptography scheme는 어떤 서버로부터 비밀키 서비스 제공 없이 협력적인 오래된 share로부터 새로운 share를 계산하는 것을 가능하게 하는 Share refreshing를 사용한다. 새로운 share는 비밀키 서비스를 위한 새로운 $(n,t+1)$ sharing을 구성한다. refreshing 후에, 서버들은 이전의 share를 제거하고, 부분 인증서를 생성해 내기 위해 새로운 shrare를 사용한다. 새로운 share들은 이전의 share들에 독립해 있기 때문에 공격자들은 오래된 share들과 비밀키 복구된 새로운 share들을 결합할 수가 없다.

이와 같이, 공격자들은 주기적인 refreshing 사이에서 $t+1$ 서버들을 공격하려고 시도할 것이다. Share refreshing은 homomorphic(이체동형,異體同形)한 성질에 의존한다. 만약 $s_1^1, s_2^1, \ldots, s_n^1$은 k의 $(n,t+1)$ sharing이고, $s_1^2, s_2^2, \ldots, s_n^2$는 k_2에 대한 $(n,t+1)$ sharing이면, $s_1^1+s_1^2, s_2^1+s_2^2, \ldots, s_n^1+s_n^2$은 k_2의 $(n,t+1)$ sharing이다. 만약 k_2가 0이면 k_1의 새로운 $(n,t+1)$ sharing을 얻을 수 있다.

주어진 n 서버는 서버 s_i의 비밀키 k의 $(n,t+1)$ sharing $s_1, s_2, \ldots, s_{n,}$을 생성한다. 모든 서버들이 정확하다고 가정하고 share refreshing 은 다음과 같은 결과를 갖는다. 먼저, 각각의 서버들은 무작위로 $s_{i1}, s_{i2}, \ldots, s_{in,}$을 생성해 낸다. 새로 생성된 share s_{ij}를 subshare라 하고, 모든 subshare s_{ij}는 안전한 경로를 통해 서버 j에 분산된다. 서버 j가 subshare $s_{1j}, s_{2j}, \ldots s_{nj}$를 취할 때 이전의 share $s_i = s_{i+}\sum_{i=1}^{n} s_{ij}$와 subshare를 통해서 새로운 share를 계산할 수 있게 된다. 다음 그림은 share refreshing 처리과정을 도시한 것이다.



【그림 6】Illustrate share refreshing process

Share Refreshing process

○ given an $(n, t+1)$ sharing $(s_1, s_2, \ldots, s_{n,})$ of a private key k with share s_i assigned to server i

○ to generated a new $(n, t+1)$ sharing $(s'_1, s'_2, \ldots, s'_{n,})$ of k

○ each server i generates subshares $s_{i1}, s_{i2}, \ldots, s_{in,}$

- $s_{i1}, s_{i2}, \ldots, s_{in,}$ constitute the i th column in the figure.

○ each subshare s_{ij} is then sent securely to server j

○ when server j gets all the subshares s_{1j}, s_{2j}, … s_{nj}

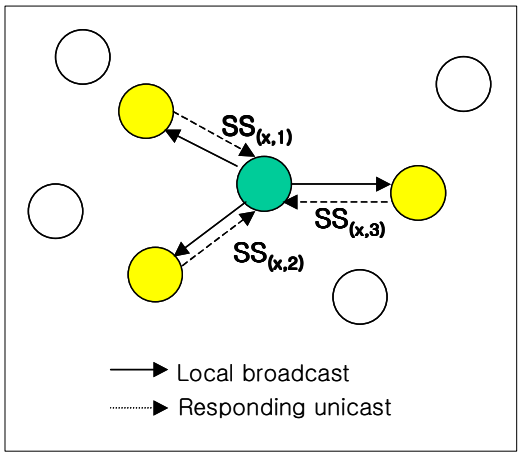
- s_{1j}, s_{2j}, … s_{nj}constitute the j th row in the figure

○ can generate new share s'_j from these subshares and its old share s_j

D. Self-initialization

Secret share를 분배해주는 역할을 분산시키기 위해 Localized self-initialization

알고리즘을 적용한다. Localized self-initialization 알고리즘은 신뢰된 딜러가 없을 때 Secret share를 가지고 있는 기존의 노드들이 새로운 노드를 위해 Secret share를 생성하면, 새로운 노드는 이를 전송받아 새로운 Secret share를 만들어 클러스터를 형성하는 알고리즘이다.



【그림 7】 Localized Self-initialization Service

$SS(x,i) = P_{v_{(x,i)}} \times L_{v_{(x,i)}}$

$l_{v_{(x,1)}}$, $SS_{(x,1)}$ is known. so, $P_{v(x,i)}$ is known

Localized self-initialization service

1. The uninitialized node v_x broadcasts the service request, along with local coalition information.
2. Each coalition member selects a random nonce for other members if its ID is lower than the other one. Each nonce is encrypted with the personal $\overline{PK}$ of the intended receiver. The requester acts as the router and accepts all shuffling packages from *K-1* members (The member with the highest ID needs not select nonces).

3. The requester routes encrypted nonces to intended receivers.
4. Each member decrypts all nonces, computes a shuffled partial secret share, and then sends it back to v_x.

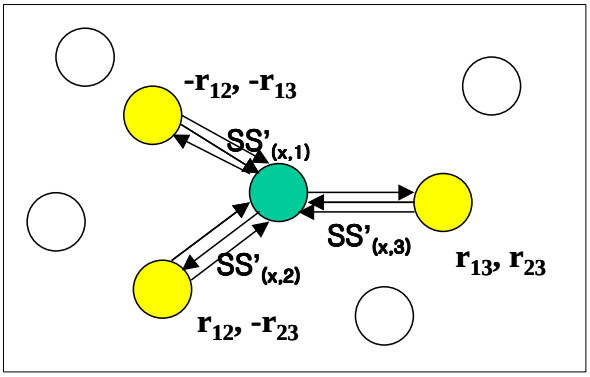
클러스터 내에 적어도 2개 이상의 타협되지 않은 노드가 있다면, Self initialization 프로토콜은 여전히 K-out-of-N 보안을 나타나게 할 수 있다. 암호 분석의 세부 내용은 [31]에서 찾아 볼 수 있다.

초기화 되지 않은 멤버 v_x(v_x는 secret share 소유자가 아니다)라 하고, 클러스터 내의 k share를 소유하고 있는 멤버들을 $\{v_{(x,1)}, v_{(x,2),}, \cdots, v_{(x,K)}\}$라 명명하면, 라크랑지 보간법에 의해 secret share P_{v_x}를 계산 할 수 있다.

$$P_{v_x} \equiv f(v_x) \equiv \sum_{j=1}^{K} P_{v_{(x,j)}} \cdot l_{v_{(x,j)}}(v_x) \equiv \sum_{j=1}^{K} SS_{(x,j)} \pmod{n}$$

각 멤버 $v_{(x,j)}$는 secret share $P_{v(x,j)}$으로부터 부분 secret share $SS_{(x,j)}$를 계산하여 부분 secret share를 요구하는 멤버에게 돌려준다.

요구자의 secret share는 k개의 부분 secret share의 합들로 구성하고, 라그랑지 계수가 공개적으로 알려지는 것처럼, 초기화 되지 않은 멤버 v_x는 알고 있는 부분 secret share $SS(x,j)$에 의해 $P_{v(c,j)}$를 유도해 낼 수 있다.



【그림 8】 Complete shuffling scheme, use random nonces

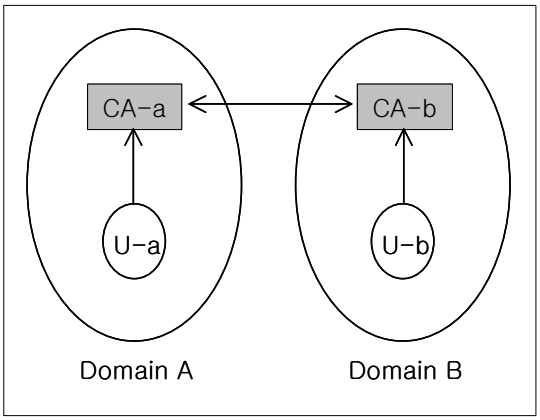
$$
\begin{aligned}
SS'_{(xi)} &= SS_{(xi)} + \sum nonces = SS'_{(x1)} + SS'_{(x2)} + SS'_{(x3)} \\
&= SS'_{(x,1)} + SS'_{(x,2)} + SS'_{(x,3)} \\
&= SS'_{(x,1)} - r_{12} - r_{13} + SS'_{(x,2)} - r_{12} - r_{13} + SS'_{(x,3)} - r_{12} - r_{13} \\
&= P_{vx}
\end{aligned}
$$

$v_{(x,j)}$를 소유한 멤버들만이 $P_{v(x,j)}$를 유지하기 위해, *complete shuffling scheme*를 사용한다. *complete shuffling scheme*은 클러스터내의 어떤 두 멤버사이에 무작위로 한번의 교환이 이루어지는데, 다른 한쪽에서 교환을 부정적으로 간주하는 동안 더 큰 ID를 가진 다른 한쪽은 교환을 명백한 수로 간주한다. 각 멤버는 전부 그러한 교환을 k-1 수행한다. 멤버 $v_{(x,j)}$는 부분 secret share를 되돌리기 전에 k-1 수행과 secret share $SS_{(x,j)}$를 합한 값, $SS'_{(x,j)} + \sum (nonces)$를 v_x에게 보낸다. 동일한 값 P_{v_x}를 포함하는 v_x를 검증하는 것이 쉽다.

E. Cross Certification

상호인증(Cross Certification)이란, 서로 다른 두 영역의 PKI를 가진 인증기관(CA, Certification Authority)이, 틀에 있는 요소들을 갖추며 상호연동을 하는 방법이다. 다음 【그림 9】는 두 개의 다른 영역의 인증기관이 서로 상호연동을 하는 그림이다. 상호인증은, 위의 그림과 같이 두 영역에서 대표되는 두 인증기관(CA) 사이에서 이루어진다. 계층적 구조를 가진 PKI에서는 대표되는 인증기관은 최상위인증기관(Root-CA)가 된다. 하지만, 꼭, 최상위인증기관(Root-CA)과 만이 가능한 것은 아니다. 인증기관(CA)들 사이에도 가능하다. 그림처럼, 인증기관 CA-A과 상대 인증기관 CA-B가 서로 교차인증을 하고 나면, 각 인증기관(CA)

에 달린 사용자 U-A와 U-B는 자연스럽게 연결이 된다.



【그림 9】 Cross Certification
between Two CAs, CA-A and CA-B

마치 두 영역이 하나의 연장된 영역으로 확장된 것처럼, 사용자 U-A에게 U-B는 그저 같은 영역의 사용자가 되는 것이다. 그 역인, 사용자 U-B에게 U-A역시 같은 영역의 사용자로 인식되게 된다. 다음은, 상호인증의 장점이다. 먼저, 장점을 이야기하면 두 가지다.

첫째는, 매끄러운 일 처리의 흐름이다. 두 영역이 만나는 영역의 연결이 매끄러워서, 마치 두 개의 PKI영역이 하나의 큰 PKI영역이 된 것처럼 일 처리의 흐름이 이루어진다.

둘째는, 사용자의 투명성인데, 사용자들에게 상호인증의 협정이 명확해 진다는 점이다.

그런 장점이 있는 반면에, 단점도 있다. 세 가지 정도로 볼 수 있겠는데, 첫째는, 절차의 복잡성이다. 세밀한 부분까지도 정책연결이 요구되어지는 성격상, 상호인증을 하는 절차가 무척 복잡하다.

둘째는, 확장성이 용이하지 않다. 항상 두 인증기관(CA)들 사이에서 연결이 되어져야 하므로, 새로운 PKI영역이 추가되려면, 추가되는 PKI영역의 CA와 기존의 다른 PKI영역의 연결이, 기존의 PKI영역간의 연결 할 때와 같은 절차로 반복되면서, 그 소요되는 연결의 개수가 기하급수적으로 늘어난다.

마지막으로, 앞의 두 가지 이유로 비추어 보면, 비즈니스의 시각에서 보면 무척 어렵고 비효율적인 방법이 된다는 것이 문제점이다. 이런 문제점들은 사실상, 전자상거래를 지원하려고 했던 기본적인 의도에 부합되지 않아서, 보다 나은 인증의 방법이 필요하게 되었다. 상호인정(Cross Recognition)이 그 대안이다.

Cross Certification mechanism ○ IF CM requires new Certificate · SK=<d,n> is divided · Divided SK is distribute to k CM · New Certificate is attained by the cooperation of the members · SK accomplish a Certificate Key role ○ CM create Partial Certificate

클러스터간 상호 인증 인증서는 해당 인접 클러스터를 탐색할 때 형성된다 인접 클러스터의 공개키가 클러스터 헤드에게 전해지면, 클러스터 헤드는 새로운 상호인증서를 생성하기 위하여, 인증서 발급과 동일한 방법으로 클러스터의 개인키를 분할하고 클러스터내의 k 노드로 분배된다. 클러스터의 개인키는 인증서 서명키로써의 역할을 하며 새로운 인증서의 발급은 k개 클러스터 멤버들의 부분 인증서 협력에 의해 이루어진다.

Ⅳ. 시스템의 구성 및 설계

A. 시스템의 구성

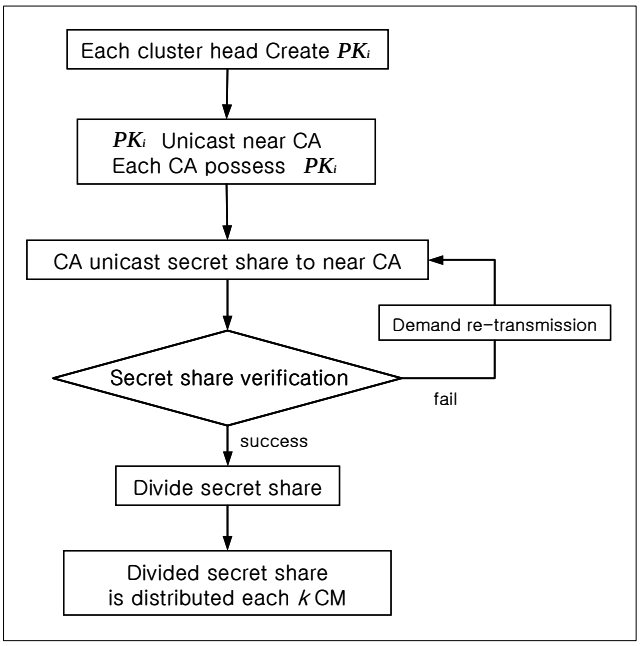
1. 시스템의 구성

MANET은 멤버(node)들이 수시로 변하는 동적인 구성을 가지고 있기 때문에 어떤 멤버가 훼손 되었을 경우, 멤버들의 신뢰관계도 변하게 되며, 신속하게 신뢰할 수 있는 다른 관리 도메인(Administrative Domain)에 가입하고 탈퇴하게 되는 다이나믹한 특성을 가지고 있다. 또한 MANET은 Ad-Hoc 네트워크의 한 부분으로 신뢰된 인증기관에서 발행하는 인증서(Certificate)를 가질 수 없기 때문에 공개키 기반의 암호화 기법을 바로 적용하는데 문제가 있다. 따라서 본 논문에서는 동적인 변화에 즉각적으로 적응할 수 있고, 신뢰된 인증기관 없이 공개키 기반 암호화 기법을 사용하는 있는 시스템을 구축, 제안하고자 한다.

본 시스템의 구성은 다음과 같다. 먼저 클러스터헤드와 클러스터 멤버간의 관계가 형성된 후 클러스터 헤드는 CA로서 도메인에 사용될 공개키와 인증서 서명을 위한 비밀키 쌍을 생성 및 구축한다. 클러스터 헤드는 생성한 공개키를 백본망을 통해 다른 클러스터 헤드에 유니캐스트함으로써 각 클러스터헤드는 도메인 공개키를 생성 보유하게 된다.

CA 역활을 담당하는 클러스터 헤드는 좌표평면상에 k개의 클러스터 멤버를 알고 있을 경우, 서로의 신뢰관계를 기반으로 교차인증(Cross-Certification)의 방법으로 secret share의 타당성을 검증한 후, 클러스터 백본망을 통해 다른 CA로 유니캐스트 한다. 이 때, 전송 받은 secret share의 값이 정당한 값인지 검증하고, 만일 검증이 실패하면 해당노드에게 실패한 secret share에 대한 재전송을 요청한다. 검증계수를 사용하여 성공적으로 전송 받은 난수에 대한 secret share의 검증이 완료되면, CA는 비밀키에 대한 부분키를 생성한다. 각 클러스터 멤버들은

부분 인증서를 생성하고, 인증서 발행을 위해 공개키에 대한 서명을 수행하고자 할 때 인접하는 멤버들에게 부분 인증서를 요청하게 된다. 부분 인증서는 CA역활을 하는 클러스터 헤드를 통해 해당 클러스터 멤버에게 전송되고 임계치 이상의 부분 인증서를 획득하게 되면 부분 인증서를 합하여 완전한 인증서를 생성할 수가 있다.



【그림 10】 System Accomplishment process

MANET의 특성상 멤버 노드들은 다른 관리 도메인에 가입하고 탈퇴하게 되는 다이나믹한 특성을 지니게 되는데, 새로운 노드가 클러스트 내에 진입 할 경우, 원활한 인증서 발급 서비스를 할 수 있도록 secret share를 분배해 줘야 한다. 제안하는 시스템에서는 Self-initialization 메커니즘을 적용하여 새로운 노드가 도메인 내의 한 클러스터 영역에 진입할 경우, 원활한 인증서 발급 서비스를 수행하기 위한 부분 서명키 SK를 분배해 주는 역할을 수행하게 된다.

2. 가정 및 접근 방법

본 시스템을 구성하기 위한 접근 방법은 다음과 같다.

첫째, 노드는 무작위로 선택되었던 목적지 노드와 session을 생성하고, 128byte로 고정된 패킷은 4pkts/sec의 일정한 비율로 전송된다.

둘째, 시그널 오버헤드를 줄이기 위해 CA 노드들 간의 플루딩 되는 메시지의 수를 감소시켜야 하므로, 키 일치를 위한 메시지를 부분 비밀키에 첨부하여 전달하고, 유니캐스트 전송 방식을 사용하여 전송 메시지의 수를 감소 시킨다.

셋째, CA 노드간 키 일치 확률을 높이기 위해 키 협의에 참여하는 노드들의 수를 가능한 제한 한다.

넷째, 키 일치 파라미터인 참여 노드수와 임계값을 모든 CA 노드들이 동일하게 정확히 파악할 수 있도록 협의에 참여하는 노드를 동적으로 파악할 수 있는 네트워크 구조를 기반 구조로 삼아야 한다.

본 시스템을 구성하기 위해 다음과 같은 가정을 설립한다.

첫째, 각각의 노드는 임의의 행선지를 향하여 0과 MAX의 일률적으로 배포되는 스피드로 움직인다.

둘째, 제안하는 시스템의 네트워크는 단일 도메인으로 구성하고 노드는 M×N의 지역 내에서 움직인다.

셋째, 클러스터를 구성하는 노드 간에 전송되는 메시지의 무결성과 기밀성은 보장된다.

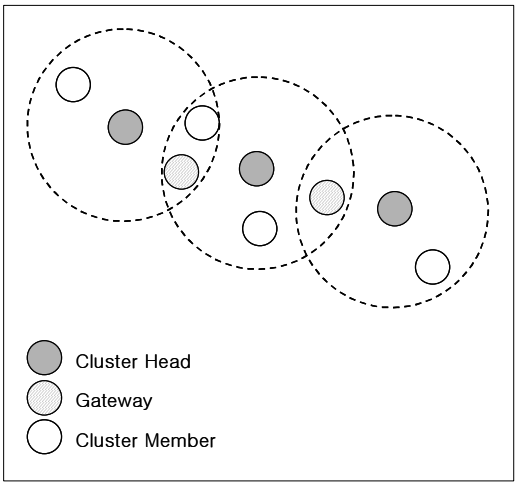
넷째, 라우팅 보안이나 외부로부터의 악의적인 공격의 가능성은 배제한다.

B. 시스템의 설계

1. 도메인 공개키와 비밀키 생성 및 구축

도메인 공개키와 인증서 서명을 위한 비밀키 생성 및 구축은 인증서 발급은 초기화 단계를 통해 클러스터 생성이 이루어지고 클러스터 헤드와 클러스터 멤버간

의 관계가 형성된 후 시작된다.



【그림 11】 Formation of Cluster

클러스터 헤드들이 CA로서 도메인에 사용될 공개키와 인증서 서명을 위한 비밀키 쌍을 생성 및 구축하는 과정이다. 헤드들은 각자 생성한 임의의 난수를 이용하여 도메인의 공개키를 협력적으로 생성하고 검증 가능한 부분 비밀키들을 분배한다. 그런데 클러스터 토폴로지가 변하게 되면 클러스터를 구성하는 멤버들의 역할도 변경되므로 클러스터 내의 키 일치 또한 대응되게 변화되어야 한다. 이는 키 일치를 위한 메시지 교환을 증가시키므로 그 연산 시간이 오래 걸릴 뿐만 아니라 네트워크에 큰 부담이 된다. 따라서 본 논문에서는 키 일치를 위한 메시지를 부분 비밀키에 첨부하여 전달하도록 한다. 다음은 도메인 공개키와 인증서 서명을 위한 비밀키 쌍을 생성 및 구축하는 과정이다.

Creation of public/private key pair

step 1. Each CA create PK_i with next calculation

- PK_i unicast near CA, each CA possess PK

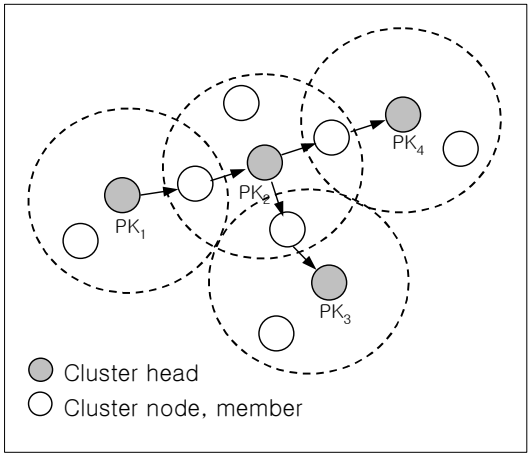
step 2. Create f(x) by (k-1) polynomial

step 3. CA unicast secret share to near CA

step 4. secret share verification

· secret share verification By the Cross-Certificate
· if fail to secret share verification
· demand re-transmission to failed secret share
step 5. if complete verification secret share
· create partial certificate

【그림 12】와 같이 CA역활을 하는 클러스터 헤드(이하 CA)는 임의의 난수 x_i를 발생시키고, 부분 공개키 값 PK_i를 생성하여 CA 백본망을 통해 다른 CA에게 유니캐스트 한다.



【그림 12】Creation and Distribution of public key

부분 공개키가 완료되면 CA들은 각각 4개의 부분 공개키 $PK_1, PK_2, \cdots, PK_4$를 획득하게 된다. 각 CA 노드는 다음의 연산에 의해 도메인 공개키를 생성하여 보유하게 된다.

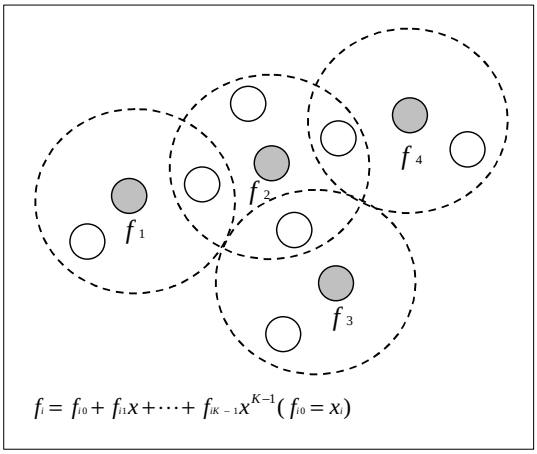
$$PK = \prod_{i=1}^{n} PK_i$$

CA 노드는 좌표평면상 k개의 노드를 알고 있을 경우 각각 생성한 난수 x_i를 y 절편으로 하는 $K-1$차 다항식 f_i를 생성한다. 서로의 신뢰관계를 기반으로 교차

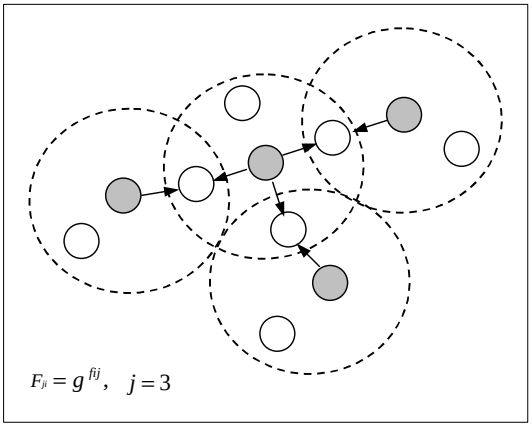
인증을 하는 방법으로 secret share의 타당성을 검증하기 위해 다항식 f_i의 계수들 f_{ij}에 대한 교차 정보 F_{ji}를 클러스터 백본망을 통해 다른 CA 노드들에게 유니케스트 한다.

$$f_i = f_{i0} + f_{i1}x^1 + f_{i2}x^2 \cdots + f_{iK-1}x^{K-1},\ (f_{i0} = x_i)$$

$$F_{ji} = g^{f_{ij}},\ (j = 0, \cdots, k-1)$$

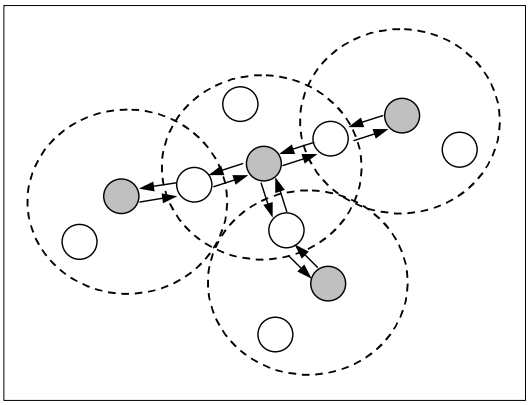


【그림 13】An exchange of secret shares used Cross Certification I



【그림 14】An exchange of secret shares used Cross Certification II

CA 노드는 자신이 생성한 임의의 수에 대한 secret share 값 $SK_{ij} = g^{f1j}$를 클러스터 백본망을 통해 다른 CA 노드들에게 유니캐스트 방식으로 전송한다.



【그림 15】Transmission of secret share key

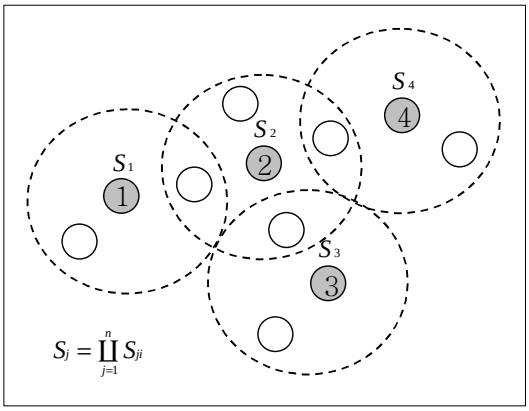
secret share의 전송이 완료되면 CA 노드는 전송 받은 secret share SK_{ij}가 정당한 값인지 검증하고, 만일 검증이 실패할 경우 해당 노드에게 실패한 secret share SK_{ij}에 대한 재전송을 요청한다.

검증 계수를 사용하여 성공적으로 전송 받은 난수에 대한 secret share의 검증이 모두 완료되면 CA노드는 다음 연산을 통해 secret share에 대한 부분키를 생성한다. .

$$SK = \prod_{j=1}^{n} SK_{ji}$$

다음 표는 각 CA가 secret share에 대한 부분키를 생성한 값이다.

S_1	S_{41}	S_{31}	S_{21}	S_{11}
S_2	S_{42}	S_{32}	S_{22}	S_{12}
S_3	S_{43}	S_{33}	S_{23}	S_{13}
S_4	S_{44}	S_{34}	S_{24}	S_{14}



【그림 16】 Creation of partial certification key

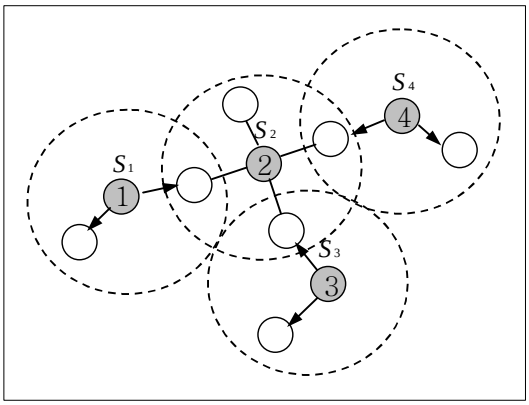
*2. Secret Share*의 분배

클러스터 기반의 인증서 발급은 초기화 단계를 통해 클러스터 생성이 이루어지고 클러스터 헤드와 클러스터 멤버간의 관계가 형성된 후 시작된다. 클러스터 헤드는 도메인 CA의 공개키와 비밀키 쌍을 생성하는데, 클러스터 내의 멤버들 중 대표로 참여하는 협력적인 분산 CA 역할을 한다. 클러스터 공개키는 앞 절의 과정을 통해 모든 클러스터 멤버들에게 알려진다.

각 클러스터 헤드는 도메인 CA의 공개키와 비밀키 쌍을 생성하는데, 클러스터 내의 멤버들 중 대표로 참여하여 협력적인 분산 CA 역할을 한다. 클러스터 공개키는 앞 절의 과정을 통해 모든 클러스터 멤버들에게 알려진다. 각 클러스터 헤드가 도메인 CA 공개키와 secret share를 구축하면 클러스터 내의 k 멤버들에게 secret share를 분배한다. 새로운 인증서는 멤버들 간의 협력에 의해 발급되어지며 비밀키는 인증키 역할을 한다. 이것은 분산된 CA 구조를 이루고 있으며, 신속하게 신뢰할 수 있는 다른 관리 도메인(Administrative Domain)에 가입하고 탈퇴하게 되는 다이나믹한 특성을 잘 적용한 것으로 이동성과 가용성이 좋아 키 일치 확률을 높일 수 있다.

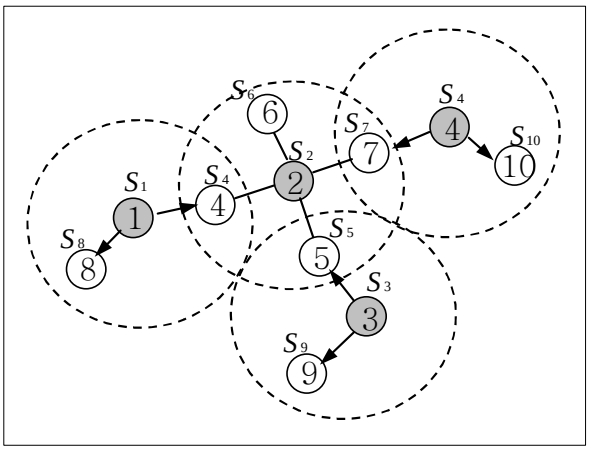
【그림 17】은 각 클러스터내의 모든 멤버들에게 secret share 분배하는 과정을

보여주고 있다.



【그림 17】 sharing secret share

다음 【그림 18】은 각 클러스터내의 모든 멤버들에게 secret share 분배가 모두 완료된 후, 분산 CA 구조를 나타내고 있는 것이다.



【그림 18】 Distributed secret share

3. 인증서 생성 및 관리

클러스터가 형성이 되면 인증기관 역할을 하기 위해 secret share를 가지고 있

는 N개의 노드 중에서 k개의 노드가 부분 전자 서명을 한 인증서 조각을 모으면 인증기관이 발행하는 인증서와 일치하는 인증서를 만들 수 있다.

본 논문에서는 Threshold Cryptography를 적용하여 CA는 생성된 부분 비밀 서명키를 클러스터 내의 k 노드들에게 분산시킴으로써 클러스터 내에 존재하는 노드가 중앙 집중적인 인증기관에 의존하지 않고 임계치 이상의 근접 노드와의 협력을 통해 인증서를 발급 받을 수 있도록 하였다. 클러스터를 구성하고 있는 각 멤버들은 Secret Share를 가지고 있으며, 인증서를 가지고 있지 않은 노드가 클러스터에 구성원으로 접속하고자 할 경우 인증서에 전자서명을 하기 위한 용도로 Secret Share를 사용 한다.

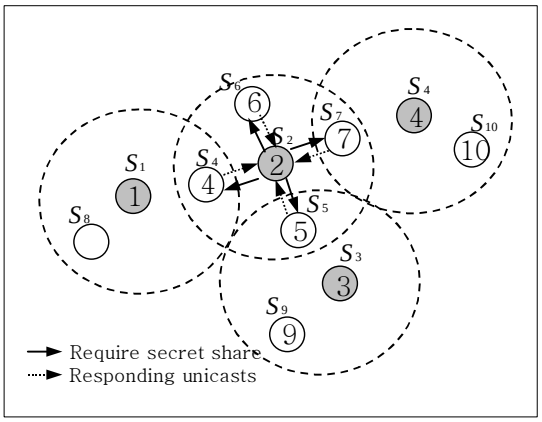
다음은 Threshold Cryptography를 적용한 인증서 생성 및 관리 과정 나타낸다.

Creation of certification and management
step1. PK=<e,n> informs to all CM(Cluster Member)
step2. IF CM requires new Certificate
- SK=<d,n> is divided
- Divided SK is distribute to k CM
- New Certificate is attained by the cooperation of the members
- SK accomplish a Certificate Key role

step3. CM create Partial Certificate
step4. Partial Certificate are transmitted to corresponding member
step5. to unite partial certificate, complete certificate is created

각 클러스터 멤버들은 부분 인증서를 생성한다. 인증서 발행을 위해 공개키에 대한 서명을 수행하고자 할 때, 인접하는 멤버들에게 부분 인증서를 요청한다. 부분 인증서는 클러스터 헤드를 통해 해당 노드로 전송되고, 임계치 이상의 부분 인증서를 획득하면, 부분인증서를 합하여 완전한 인증서를 생성할 수 있다. 가령 CA2가 인증서 발행을 위해 공개키에 대한 서명을 수행하고자 할 경우, CA2는 클러스터 내의 멤버 노드들에게 secret share를 요청하고, 임계치 이상의 secret

share가 획득되면 서명키를 생성하여 인증서를 발행 할 수 있다. 【그림 19】는 인증서 생성 방식의 예를 도시한 것이다.



[그림 19] Creation of complete certificate

형성된 클러스터 내에 개별적으로 존재하는 노드들의 Secret Share를 공유하는 방법은, 비밀 다항식(Secret Polynomial) $f(x)$에 의해 이루어진다. 즉, 개인키/공개키 쌍 SK=<d,n> / PK<e,n>을 공개하는데 좌표명면상의 k개의 노드들을 알고 있을 경우 Lagrange Interpolation(라그랑지 보간법)을 통해 K-1차 다항식을 다음 수식(1)과 같이 정의함으로써 완전한 비밀(Secret)이 복구 될 수 있다.

$$f(x) = d + f_1 \cdot x + f_2 \cdot x^2 + \cdots + f_{K-1} \cdot x^{K-1}, \qquad N(> K) \qquad (1)$$

단, 해당 집단 내의 노드들의 수가 k이하인 경우, 완전한 비밀에 대한 정보는 얻을 수 없다. 여기서 Shared Secret $f(0) = d$이고, 클러스터 C에 존재하는 노드 i가 가지는 Secret Share는 $P_i = f(C_i) \bmod n$이다.

인증서를 생성하기 위해서는 비밀키 SK=<d,n>를 사용하여 전자서명을 하여야 한다. 비밀키 SK=<d,n>는 k개의 Secret Share를 수집하여 d를 계산하면 쉽게 드러나게 된다.

$$d \equiv \sum_{i=1}^{K} (P_{C_i} \cdot l_{C_i}(0) \bmod n) \qquad (2)$$

$$\equiv \sum_{i=1}^{K} SK_i \, (\bmod n)$$

여기서, $l_{C_i}(0)$은 Lagrange Coefficient(라그랑지 계수)이며, 클러스터 내에 존재하는 k개 이상의 노드들이 협력에 의해 부분 서명한 인증서 조각을 사용하여 SK=<d,n>를 생성할 수 있다. Lagrange Interpolation에 의해 sk_i는 수식(2)의 다항식 d로부터 복구될 수 있으며, 수식(1)에 의해 다음 수식(3)과 같이 표현될 수 있다.

$$d \equiv \sum_{i=1}^{K} (P_{C_i} \cdot l_{C_i} \bmod n) \qquad (3)$$

$$\equiv t \times n + d$$

수식 (3)의 $(P_{C_i} \cdot l_{C_i} \bmod n)$는 n에 대한 모듈로 연산에 대해 d와 같은 값을 가지는 값이므로 k개의 인증서 조각을 곱하는 방법만으로는 완벽한 인증서를 생성하는데 한계가 있다. 따라서, K-bounded Coalition Offsetting 알고리즘을 적용하여 완전한 인증서를 생성할 수 있다[32].

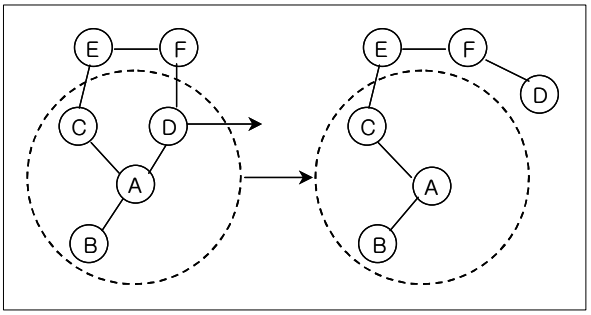
$$M^{t \times n + d} \equiv M^{t \times n} \times M^{d} \equiv = M^{d} (\bmod n)$$

여기서, M은 시스템의 공개키<e,n>이고, Threshold K는 각 집단 내에 존재하는 복수의 노드들을 의미하며, 반드시 큰 수일 필요는 없다.

새로운 인증서는 멤버들 간의 협력에 의해 발급되어지며 비밀키는 인증키 역할을 한다. 이것은 분산된 CA 구조를 이루고 있으며, 신속하게 신뢰할 수 있는 다른 관리 도메인(Administrative Domain)에 가입하고 탈퇴하게 되는 다이나믹한 특성을 잘 적용한 것으로 이동성과 가용성이 좋아 키 일치 확률을 높일 수 있다.

4. 인증서 갱신

MANET의 특성상 노드들은 다른 관리 도메인(Administrative Domain)에 가입하고 탈퇴하게 되는 다이나믹한 특성을 지니게 되는데, 새로운 노드가 도메인 내의 한 클러스터 영역에 진입할 경우, 원활한 인증서 발급 서비스를 수행할 수 있도록 Secret Share를 분배해 줘야 한다. 즉, 클러스터가 재 구축 되면, Secret share를 가지고 있는 멤버들이 새로운 멤버를 위해 secret share를 만들어 새로운 멤버에게 전송하면 그것을 사용하여 새로운 멤버의 secret share를 만들어 내는 것이다. 반대로 클러스터내의 멤버들의 수가 임계치 밑으로 감소되었을 경우, 임계치 이하의 부분키로는 서명키를 획득할 수 없기 때문에 그룹의 보안을 보장할 수 없게 된다. 따라서 클러스터 내의 멤버가 임계치 이상이 되도록 클러스트를 구축하고, CA를 선정한 후 공개키와 비밀 서명키 쌍을 재 구축해야한다.
(t,n)Threshold Cryptosystem에스는 주기적으로 키를 재 생성하는 경우 안전도가 더 증가하는 것으로 알려져 있다[33][34].
따라서 멤버들의 변화가 없더라도 클러스터 헤드는 주기적으로 공개키, 비밀키를 생성하여 메시지를 멤버들에게 전송하여 보안에 안전을 취할 수 있다.
그림 【그림 20】은 새로운 노드가 클러스터 내로 진입할 경우 변환된 Ad-hoc 네트워크의 토폴로지를 보여 주고 있다.



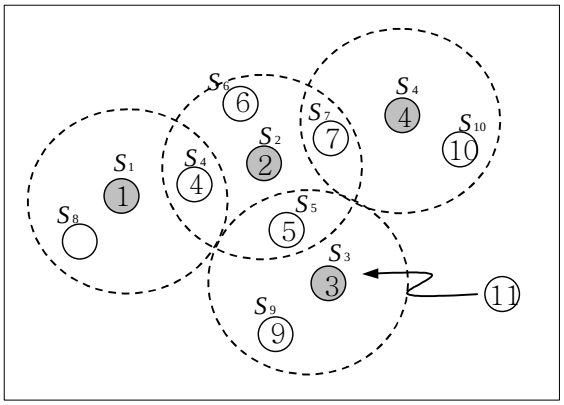
【그림 20】 Topology changes in ad hoc networks.

【그림 20】과 같이 새로운 노드가 도메인 내의 한 클러스터 영역에 진입할 경

우, 원활한 인증서 발급 서비스를 수행하기 위해 부분 서명키 SK를 분배해 주기 위해서 Self-initialization 메커니즘을 적용한다. 이는 기존의 통신하고 있는 노드들이 새로운 노드 유입에 영향 받지 않고 동일한 서명키로 인증 서비스를 수행하도록 할 수 있으며, 새로운 노드에 분산 CA 기능을 분배함으로써 높은 확장성을 지원할 수 있다.

클러스터 내에 새로운 노드가 진입하면, sceret share를 지닌 멤버노드는 CA에게 초기화되지 않은 노드를 위한 새로운 secret share를 생성할 것을 요청한다. CA는 새로운 노드를 멤버 노드로 등록하고 부분 비밀키를 생성하여 새로운 노드에게 분배함으로써 클러스터를 형성하게 된다. 다음은 Self-initialization를 적용하여 클러스터 내에 새로운 노드의 진입 시, 클러스터가 재형성 되는 과정을 나타낸 것이다.

1. The uninitialized node v_x broadcasts the service request, along with local coalition information.
2. Each coalition member selects a random nonce for other members if its ID is lower than the other one. Each nonce is encrypted with the personal $\overline{PK}$ of the intended receiver. The requester acts as the router and accepts all shuffling packages from k-1 members (The member with the highest ID needs not select nonces).
3. The requester routes encrypted nonces to intended receivers.
4. Each member decrypts all nonces, computes a shuffled partial secret secret share, and then sends it back to v_x.



【그림 21】 The joining of new node

【그림 21】과 같이 새로운 노드 No.11이 클러스터 영역에 진입할 경우, 새로운 노드는 헤드노드 CA3에게 클러스터 멤버 등록을 요청한다. 헤드 노드는 새로운 노드를 클러스터 멤버로 등록하고 부분 비밀키를 생성하여 새로운 노드에게 분배한다. 이와 같은 방법은 기존에 통신하고 있던 노드들이 새로운 노드의 진입에 영향을 받지 않고 동일한 서명키로 인증 서비스를 수행할 수 있도록 하는 결과를 얻을 수 있다.

V. 실험 및 평가

본 논문에서 MANET환경에서 공개키 기반 암호화 기법을 사용하려고 할 때 가장 문제가 되는 신뢰된 인증기관(Trusted Certificate Authority)의 부재로 인한 신뢰된 인증서(Certificate)를 가질 수 없다는 점을 해결하기 위하여 Threshold Cryptography mechanism을, 그리고, 공개키와 비밀키 생성하고 공유하기 위하여 Secret-Sharing, Share-refreshig, Cross-Certificate mechanism을, 분산된 CA 기능을 확장하기 위하여 Self-initialization mechanism 등을 적용하여 네크워크 효율성과 보안의 견고성을 개선하기 위한 시스템 모델을 제안하였다. 제안한 시스템을 검증하기 위한 시뮬레이션 도구로는 유닉스 및 리눅스 환경에서 최적으로 구동되고, TCL(Tool Command Language) 스크립트 언어에 객체(Object) 지향의 개념을 추가한 OTCL(Object TCL)과 C++로 구성된 NS2(Network Simulator 2)를 이용하였다. 시뮬레이션의 결과를 기존의 MANET 라우팅 프로토콜과 비교해 봄으로서 본 논문에서 제시하는 시스템 모델의 안전성 및 효율성과 견고성을 평가해 보도록 한다.

A. 실험 환경 및 실험 도구

1. 실험 환경

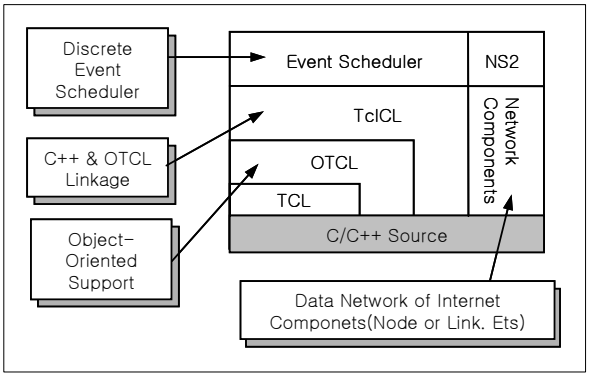
본 논문에서 제안한 시스템을 시뮬레이션 하기위하여 NS2(wireless extention)를 이용하며, IEEE 802.11 link layer와 TDMA(Time Division Multiple Access)를 따른다. 노드는 무작위로 선택되었던 목적지 노드와 session을 생성하고, 데이터의 크기는 일정한 비율로 전송된다. 각각의 노드는 임의의 행선지를 향하여 0

과 MAX의 일률적으로 배포되는 스피드로 움직인다. 제안하는 시스템의 네트워크는 단일 도메인으로 구성하고 노드는 M×N의 지역 내에서 움직이고, 클러스터를 구성하는 노드 간에 전송되는 메시지의 무결성과 기밀성은 보장됨을 가정한다. 라우팅 보안이나 외부로부터의 악의적인 공격의 가능성은 배제한다.

2. 실험 도구 NS2

(가) 소개

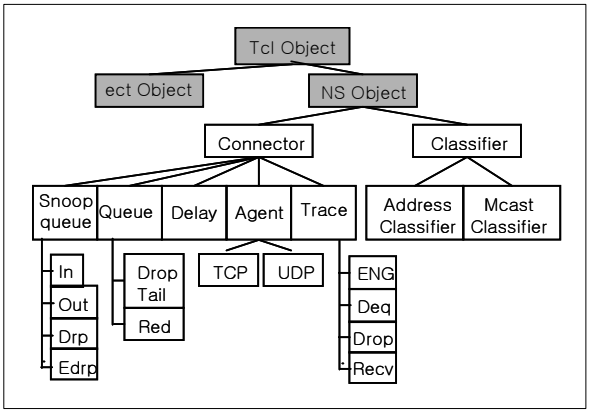
NS2 시뮬레이터는 TCP, UDP, FTP, HTTP 등과 같은 TCP/IP 프로토콜 패밀리와 라우팅 프로토콜, 그리고 멀티캐스팅 프로토콜, RTP, SRN 등과 같은 다양한 인터넷 프로토콜을 시뮬레이션 할 수 있다. 그리고 Ad-Hoc 네트워크, 이동통신망의 기지국(Base Station)모델, WLAN, Mobile-IP 관련 프로토콜, UMTS(Universal Mobile Telecommunication System), 위성 네트워크(Satellite Network) 등과 같은 무선 네트워크까지 지원 할 수 있는 매우 적용 범위가 넓은 네트워크 시뮬레이터이다.



【그림 22】 Architectural View of NS

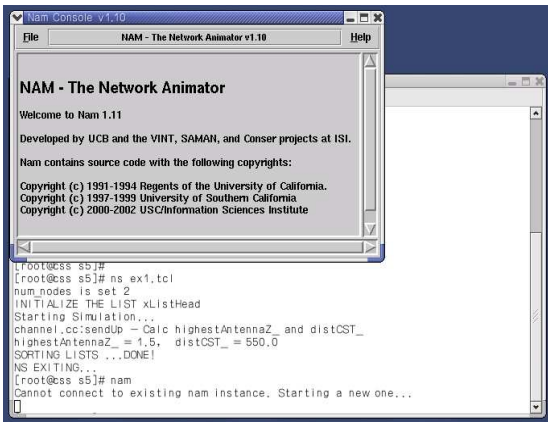
NS2 시뮬레이터에서 사용하는 프로그램 언어는 TCL(Tool Command Language) 스크립트 언어에 객체(Object)지향의 개념을 추가한 OTCL(Object

TCL)과 C++이다. 여기에서, C++로 기술된 부분들은 주로 소스 코드를 정의한다. 그리고 OTCL은 사용자 인터페이스와 매개 변수 등을 정의하였다. 그리고 Tclcl을 OTCL과 C++을 연동시키기 위해 사용하였다. 다음 그림은 NS2의 OTCL 클래스의 계층 구조를 나타내고 있으며, 이 클래스 계층 구조에 있는 NsObject 부분에 네트워크 컴포넌트가 포함되어 있다.

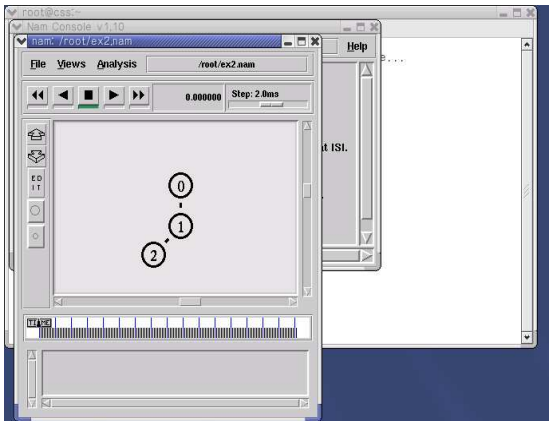


【그림 23】Class hierarchy of NS2

NS2는 애니메이션 툴이 제공되는데, 이 프로그램은 스크립트로 작성한 소스 코딩에 대한 시뮬레이션 실행, 결과 등을 디스플레이 장치에 직접 보여주는 기능을 제공한다. 다음 그림은 리눅스에서 스크립트 소스를 실행했을 때 디스플레이 화면에 나타나는 시뮬레이션 결과와 NAM 디스플레이 장치를 보여주고 있다.

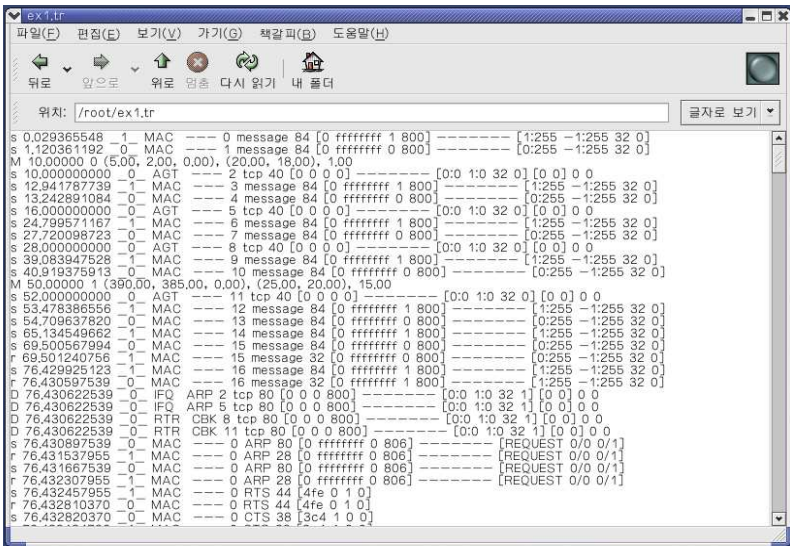


【그림 24】NS simulation script

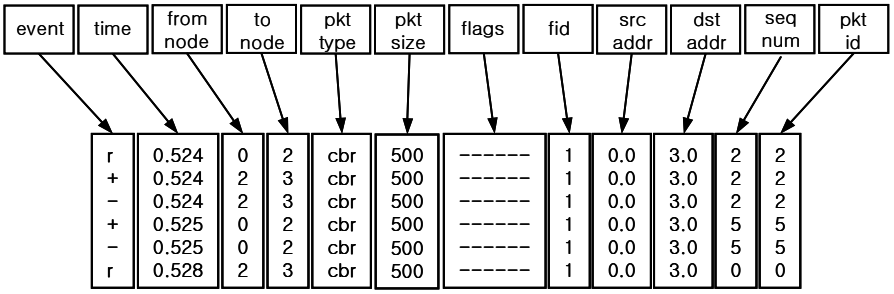


【그림 25】 Execution of simulation using NAM

NS 시뮬레이터를 이용하면 패킷 흐름을 눈으로 직접 볼 수 있는 NAM외에도, 시뮬레이션 결과를 쉽게 사용자에게 제공한다. 즉, 패킷 flow를 아주 짧은 시간 단위로 나누어 그때마다 패킷의 상태를 추적(tracing)할 수 있는 기능을 제공하여 사용자가 원하는 네트워크 성능 분석 자료를 제공하고 있다.



【그림 26】 Tracking of packet trace



【그림 27】 Format of trace file

본 논문에서는 제안한 시스템의 성능분석을 위하여 패킷 flow에 대한 추적결과를 근거로 데이터의 양과 시간을 계산, 다른 MANET의 라우팅 프로토콜들과 비교 분석하였다. 이를 통해 제안한 시스템의 네트워크 보안의 안전성 및 효율성과 견고성 등의 평가를 해 보도록 한다.

B. 시스템 평가

1. 시스템 효율성 및 안정성 평가

노드(K)의 변화에 따라 Partial Certificate(인증서 조각)을 생성하는데 요구되는 시간과 인증서 조각을 사용하여 완전 인증서를 생성하는데 요구되는 시간 및 데이터양을 측정하여 제안한 시스템의 안정성 및 효율성을 평가해 보도록 한다.

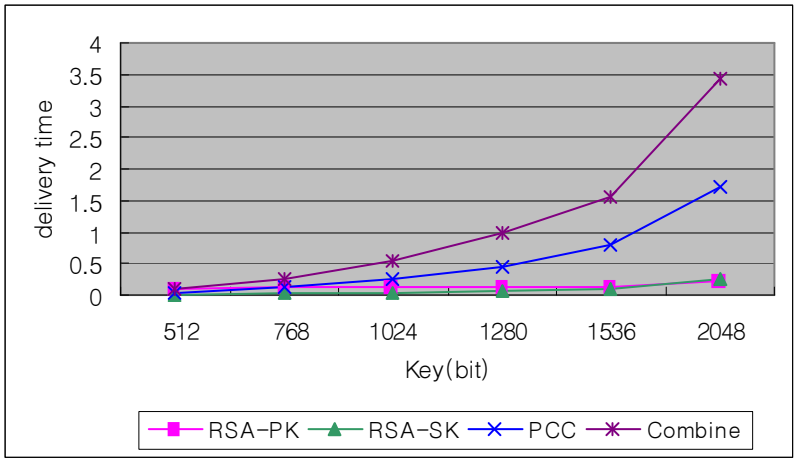
다음 [표 9][표 10][표 11]는 연산 장치의 성능에 따른 RSA 키 계산 시간을 나타낸 것이다[33]. PCC는 Secret Share를 사용하여 Partial Certificate를 계산하는 시간을 보여주고 있으며, Combine은 K Partial Certificates의 결합에 의하여 생성되는 인증서 생성 시간을 보여주고 있다. RSA-PK는 공개키 계산시간을, RSA-SK는 개인키 전자서명시간 나타낸다.

【표 1】는 K=5, SPEC=20.5인 경우, RSA 공개키 및 개인키 서명 시간과 Secret Share를 사용하여 Partial Certificate를 계산한 시간 PCC와 K Partial Certificates의 결합에 의해 생성된 인증서 생성 시간 Combine을 서로 비교한 표이다.

Key (bit)	RSA-PK (msec)	RSA-SK (sec)	PCC (sec)	Combine (sec)
512	0.093	0.0056	0.0466	0.0928
768	0.124	0.0173	0.1198	0.2416
1024	0.142	0.0386	0.2610	0.5280
1280	0.136	0.0669	0.4590	0.9742
1536	0.133	0.1089	0.7944	1.5598
2048	0.208	0.2462	1.7058	3.4410

【표 1】Calculation time of certification by RSA Key bit (K=5, SPEC=20.5), SPEC=20.5≒PentiumIII/500 laptop

【그림 28】은 【표 1】을 그래프로 도시화한 것이다.



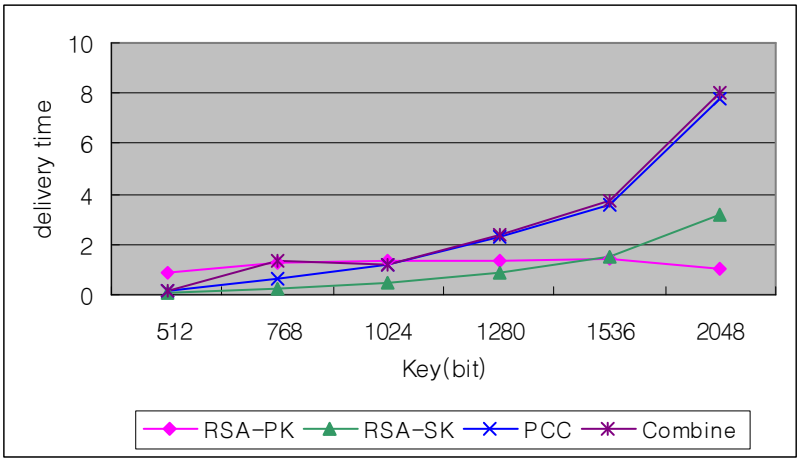
【그림 28】Calculation time of certification by RSA Key bit (K=5, SPEC=20.5)

【표 2】는 K=5, SPEC=12.1인 경우, RSA 공개키 및 개인키 서명 시간과 Secret Share를 사용하여 Partial Certificate를 계산한 시간 PCC와 K Partial Certificates의 결합에 의해 생성된 인증서 생성 시간 Combine을 서로 비교한 표이다.

key (bit)	RSA-PK (msec)	RSA-SK (sec)	PCC (sec)	Combine (sec)
512	0.884	0.0678	0.1835	0.1982
768	1.276	0.2165	0.5973	1.3430
1024	1.324	0.4672	1.1637	1.1978
1280	1.356	0.8734	2.2912	2.4109
1536	1.416	1.4863	3.5820	3.6952
2048	1.036	3.1883	7.7855	8.0324

【표 2】Calculation time of certification by RSA Key bit (K=5, SPEC=12.1)

【그림 29】은 【표2】를 그래프로 도시화한 것이다.



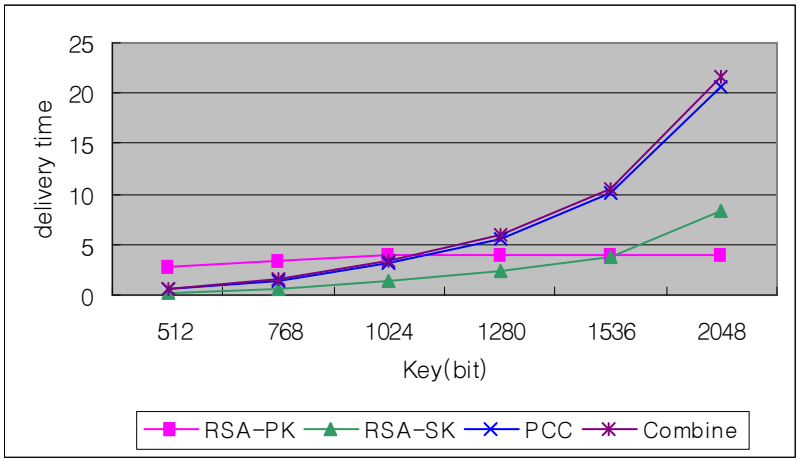
【그림 29】 Calculation time of certification by RSA Key bit (K=5, SPEC=12.1)

【표 3】는 K=5, SPEC=1.37인 경우, RSA 공개키 및 개인키 서명 시간과 Secret Share를 사용하여 Partial Certificate를 계산한 시간 PCC와 K Partial Certificates의 결합에 의해 생성된 인증서 생성 시간 Combine을 서로 비교한 표이다.

key (bit)	RSA-PK (msec)	RSA-SK (sec)	PCC (sec)	Combine (sec)
512	2.782	0.2347	0.5499	0.6144
768	3.382	0.6403	1.4818	1.6478
1024	4.036	1.2953	3.1738	3.3283
1280	4.065	2.4607	5.5492	5.9019
1536	3.941	3.8543	10.1253	10.4301
2048	3.954	8.3826	20.6606	21.7095

【표 3】 Calculation time of certification by RSA Key bit (K=5, SPEC=1.37), SPEC=1.37≒SPARC station5/85

【그림 30】은 【표3】을 그래프로 도시화한 것이다.



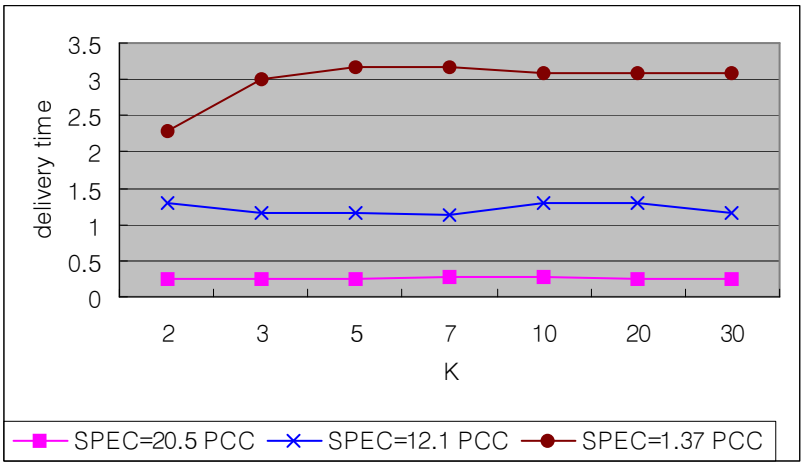
【그림 30】Calculation time of certification by RSA Key bit (K=5, SPEC=1.37)

【표 4】는 K 값의 변화에 따라 secret share를 사용하여 Partial certificate를 생성하는데 요구되는 시간과, K Partial Certificates의 결합에 의해 완전한 인증서를 생성하는데 요구되는 시간을 측정한 표이다.

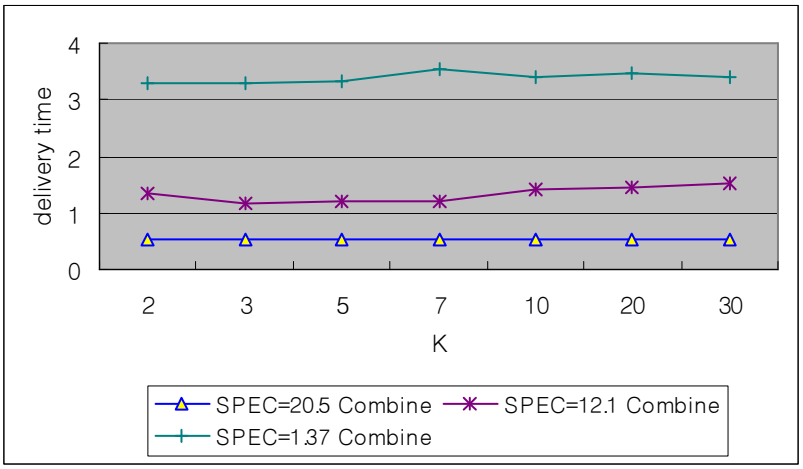
K	SPEC=20.5		SPEC=12.1		SPEC=1.37	
	PCC	Combine	PCC	Combine	PCC	Combine
2	0.260	0.526	1.293	1.334	2.299	3.304
3	0.261	0.528	1.149	1.171	2.998	3.293
5	0.261	0.528	1.164	1.198	3.174	3.328
7	0.263	0.531	1.140	1.207	3.163	3.530
10	0.262	0.537	1.309	1.410	3.099	3.394
20	0.261	0.532	1.308	1.464	3.078	3.458
30	0.261	0.537	1.160	1.510	3.082	3.410

【표 4】Certification performance in terms of system parameter K
RSA key : 1024bit, time : sec

【그림 31】 【그림 32】은 【표 4】를 그래프로 도시화 한 것이다.



【그림 31】Partial Certification performance
in terms of system parameter K



【그림 32】 Certification performance in terms of system parameter K

【표 5】는 제안한 시스템에서 인증서를 계산 하는 시간을 도시한 것이다.

TPC(가칭, Times of the Partial certificate Computation which using Lagrange interpolation operation)는 Lagrange interpolation을 이용하여 Partial Certification을 계산한 시간을 나타내고, Sum은 모든 K Partial Certificates를 합하여 비밀키를 획득, 인증서를 생성하는 계산시간이다.

Key	SPEC=20.5		SPEC=12.1		SPEC=1.37	
(bit)	TPC	Sum	TPC	Sum	TPC	Sum
512	0.407	0.301	1.568	0.361	3.556	1.121
768	0.462	0.361	2.423	0.512	4.982	1.211
1024	0.488	0.322	3.311	0.665	6.889	1.568
1280	0.551	0.426	4.511	0.864	8.563	1.795
1536	0.801	0.462	5.212	0.856	11.546	1.998
2048	1.461	0.488	6.245	0.756	19.871	2.265

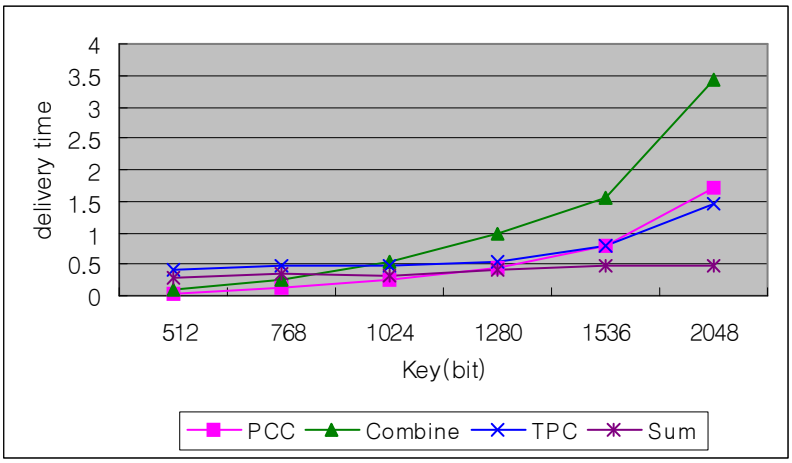
【표 5】 Times of the partial certificate computation which using Lagrange interpolation operation
RSA key : 1024bit, time : sec

【표 6】 【표 7】 【표 8】는 Key의 bit수가 증가에 따른, PCC, Combine과 제안한 시스템의 TPC, Sum를 각각 비교한 결과이다. 【표 6】는 Key bit가 1024bit이고, SPEC=20.5인 경우 TCP와 PCC를 비교한 결과이다.

key (bit)	PCC	Combine	TPC	Sum
512	0.0466	0.0928	0.407	0.301
768	0.1198	0.2416	0.462	0.361
1024	0.2610	0.5280	0.488	0.322
1280	0.4590	0.9742	0.551	0.426
1536	0.7944	1.5598	0.801	0.462
2048	1.7058	3.4410	1.461	0.488

【표 6】Compare TPC with PCC
key : 1024bit, SPEC=20.5

【그림 33】는 Key bit가 1024bit이고, SPEC=20.5인 경우 Key의 bit수가 증가에 따른, PCC, Combine과 제안한 시스템의 TPC, Sum를 각각 비교한 결과를 그래프로 도시한 것이다.



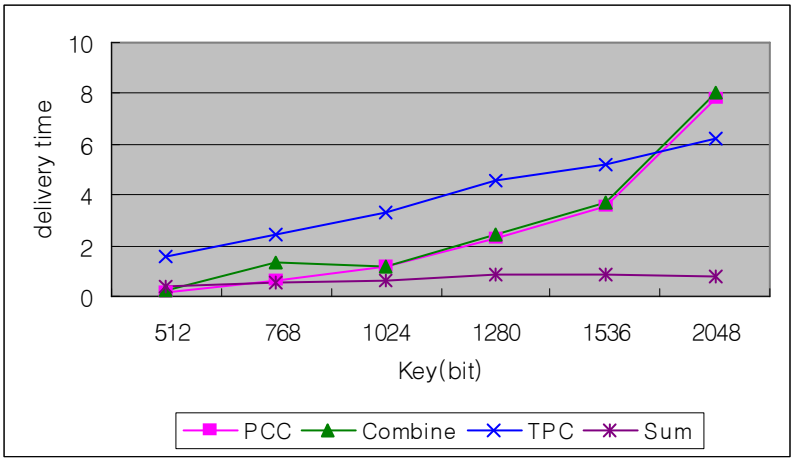
【그림 33】Compare TPC with PCC
key : 1024bit, SPEC=20.5

【표 7】는 Key bit가 1024bit이고, SPEC=12.1인 경우 TPC와 PCC를 비교한 결과이다.

key (bit)	PCC	Combine	TPC	Sum
512	0.1835	0.1982	1.568	0.361
768	0.5973	1.3430	2.423	0.512
1024	1.1637	1.1978	3.311	0.665
1280	2.2912	2.4109	4.511	0.864
1536	3.5820	3.6952	5.212	0.856
2048	7.7855	8.0324	6.245	0.756

【표 7】Compare TPC with PCC
key : 1024bit, SPEC=12.1

【그림 34】는 Key bit가 1024bit이고, SPEC=12.1인 경우 Key의 bit수가 증가에 따른, PCC, Combine과 제안한 시스템의 TPC, Sum를 각각 비교한 결과를 그래프로 도시한 것이다.



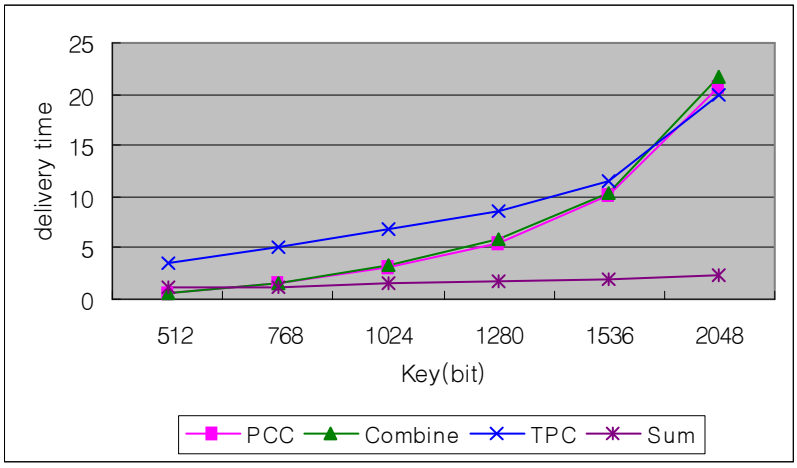
【그림 34】Compare TPC with PCC
key : 1024bit, SPEC=12.1

【표 8】는 Key bit가 1024bit이고, SPEC=1.37인 경우 TCP와 PCC를 비교한 결과이다.

key (bit)	PCC	Combine	TPC	Sum
512	0.5499	0.6144	3.556	1.121
768	1.4818	1.6478	4.982	1.211
1024	3.1738	3.3283	6.889	1.568
1280	5.5492	5.9019	8.563	1.795
1536	10.1253	10.4301	11.546	1.998
2048	20.6606	21.7095	19.871	2.265

【표 8】Compare TPC with PCC
key : 1024bit, SPEC=1.37

【그림 35】는 Key bit가 1024bit이고, SPEC=1.37인 경우 Key의 bit수가 증가에 따른, PCC, Combine과 제안한 시스템의 TPC, Sum를 각각 비교한 결과를 그래프로 도시한 것이다.



【그림 35】Compare TPC with PCC
key : 1024bit, SPEC=1.37

【표 6】【표 7】【표 8】과 【그림 33】【그림 34】【그림35】의 결과를 분석해보면, Key의 bit가 증가함에 따라, PCC, Combine이 점차적으로 증가되는 것을 볼 수 있다. 이러한 결과는 packet data length가 증가함에 따라 overhead 발생률도 증가하게 되므로, 네크워크의 효율성을 떨어뜨리게 된다. 이러한 문제점은 Ad-Hoc 네크워크의 안정성 및 효율성을 저해하는 요인이 된다. 또한 불안정한 네트워크로 인하여 전달되는 메시지가 중간에 도청 또는 변조될 위험이 있으며, 통신 중 메시지가 도중에 유실될 가능성이 높게 되어 네트워크의 안정성을 저해하는 요인이 된다.

반면, 제안한 시스템에서는 Key의 bit가 증가하더라도 TPC, Sum의 값은 bit가 늘어남에 따라 다소 증가함을 보여주고 있으나 기존의 PCC, Combine에 보다 현저하게 줄어들었음을 확인 할 수 있다. 특히, 모든 부분 인증서를 합하여 비밀키를 획득하여, 인증서를 생성하는 시간 Sum은 Packet data length가 늘어난다 하더라도 일정한 비율의 시간을 갖는 것을 확인할 수 있다.

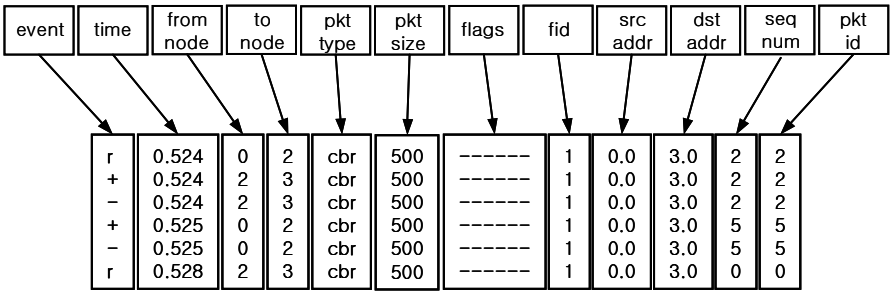
따라서, 본 논문에서 제안한 시스템에서는 bit(packet data length)가 늘어난다 하더라고 일정한 delivery time을 유지함으로 네트워크의 안정성 및 효율성을 보장 받을 수 있다.

또한, K 값의 변화에 따라 인증서 조각을 생성하는데 요구되는 시간과 완전한 인증서를 생성하는데 요구되는 시간을 측정한 결과, Partial Certificate들은 연합 멤버들에 의해 일정하게 계산되기 때문에 매개 변수 K가 시스템 효율에 영향을 미치지 않은 것을 발견할 수 있으며, 특히 Partial Certificate를 요구하는 측면에서 Certificate를 생성하는 모든 작동이 이루어지므로 오버헤드 계산에 관해서 적절한 결과를 보여주고 있다. 이 기법은 Lagrange interpolation과 역수(逆數)를 적용한 self-initialization과 proactive 갱신기법을 이용한 것으로, 1/1000sec의 스케일에서 부분 비밀키 생성 계산 시간 적게 소요 되고, 오버헤드를 줄일 수 있음을 알 수 있다.

2. 견고성 평가

이 절에서는 CA 비밀키를 구축하기 위해 전송되는 패킷 및 부분 키 메시지의 키 일치율과 노드 수의 증가에 따른 라우팅의 오버헤드를 비교해 봄으로써 견고성을 평가해 보도록 한다.

【그림 36】은 시뮬레이션을 통한 결과를 도시한 trace format이다. 먼저, 맨 앞에 위치하고 있는 데이터는 event를 나타내며, 각각은 다름과 같다.



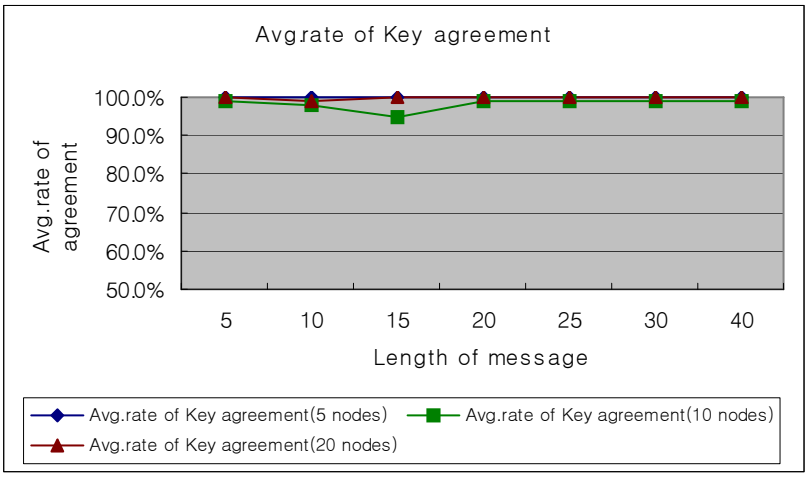
【그림 36】Format of trace file

+ : 큐에 들어오는 패킷
- : 큐에서 나가는 패킷
s : 패킷전송(sending)
r : 패킷수신(receiving)
D : 패킷 폐기(dropping)
f : 패킷 전달(forwarding), 송신 노드가 아닌 경우를 나타낸다.

먼저, 노드 수의 증가와 각 노드 간 송・수신 되는 length of message(packet) 비율을 비교해 봄으로써 키 일치성을 평가해 보았다. 노드의 증가에 따른 시뮬레이션의 결과는 본 논문의 부록에 수록하였다.

【그림 37】은 노드 K가 5, 10, 20인 경우의 키 일치율을 도시한 것이다. Avg. rate of Key agreement는 전송하는 패킷과 수신되는 패킷의 데이터의 비율을 나

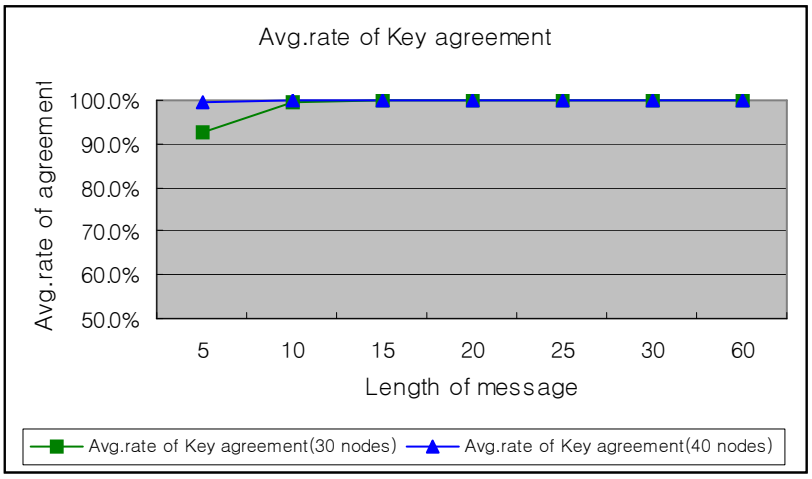
타내고, length of message는 패킷의 크기를 나타낸다.



【그림 37】 Avg. rate of Key agreement
(node K=5, 10, 20)

각 노드들 간의 송·수신되는 packet은 packet length가 커 지더라도 키 일치율이 100%에 근사하게 높게 나타나고 있으며, 노드의 수가 5, 10, 20으로 증가하더라도 키 일치율에는 영향을 미치지 않고 일정하게 유지됨을 알 수 있다.

【그림 38】은 노드 K가 30, 40인 경우의 키 일치율을 도시한 것이다.



【그림 38】Avg. rate of Key agreement
(node K=30, 40)

클러스터내의 노드가 30, 40 증가하더라도 100%에 근사한 키 일치율을 보여주고 있으며, packet length가 증가하더라도 키 일치율에는 영향을 미치지 않음을 알 수 있다. 이는 CA 비밀키를 구축하기 위해 전송되는 패킷 및 부분 키 메시지의 키를 완벽하게 생성 할 수 있음을 보여주고 있다. 또한 클러스터 내의 노드 수가 증가하더라도 노드 간 packet을 일정하게 송・수신 할 수 있게 됨으로 오버헤드를 줄일 수 있어 견고성을 보장 받을 수 있다.

Ⅵ. 결론

MANET은 기존의 네트워크와 같이 인프라가 구축된 상태에서 통신을 수행하는 것이 아닌 인프라가 존재하지 않은 상태에서 각 노드들 상호간의 라우팅으로 데이터 송·수신 등의 통신 기능을 수행할 수 있는 형태의 네트워크를 말한다. 따라서 네트워크에 참여하는 각 노드들은 기지국이나 AP의 도움 없이 자신들이 라우터, 서버의 역할 등을 모두 수행하면서 다른 노드 대신 패킷을 전송할 수도 있고, 사용자 애플리케이션들을 실행할 수도 있어야 한다. MANET에서 공개키 기반 암호화 기법을 사용하려고 할 때 가장 문제가 되는 신뢰된 인증기관(Trusted Certificate Authority)의 부재로 인한 신뢰된 인증서(Certificate)를 가질 수 없다는 점이다. 따라서 본 논문에서는 이러한 문제점을 해결하기 위하여 Threshold Cryptography를 적용하였으며, Secret-Share를 통한 공개키와 비밀키 생성 및 공유 메커니즘, 분배된 CA 기능을 확장하기 위하여 Self-initialization 메커니즘, 그리고 견고성의 개선하기 위하여 마스터 키 구축 메커니즘으로 구성된 모델을 제안하였다. K Partial Certificates의 결합에 의해 생성되는 인증서의 생성시간은 Secret Share를 사용하여 Partial certificate를 계산하는 시간의 경우 Key의 bit값이 증가함에 따라 인증서 생성 시간이 증대된다. 이는 packet data length가 증가함에 따라 overhead도 발생률도 증가하게 되므로 네트워크의 효율성을 떨어트리게 된다. 이러한 문제점은 MANET의 안정성 및 효율성을 저해하는 요인이 된다. 불안정한 네트워크는 통신 중 인증을 위한 메시지가 중간에 도청 또는 변조될 위험에 노출될 수가 있으며, 메시지가 유실 될 경우 안정된 네트워크를 보장 받을 수 없게 된다. 반면, 본 논문에서 제안하는 시스템에서는 K Partial Certificate를 결합하여 완정한 인증서를 생성하는 시간이, bit가 늘어남에 따라 다소 증가함을 보여주고 있으나 기존의 시스템보다 현저하게 줄어들었음을 확인 할 수 있었다. 특히, 모든 Partial Certificate를 합하여 비밀키를 획득, 인증서를 생성하는 시간은 Packet data length가 늘어난다 하더라도 일정한 비율의

delivery time 유지함으로 네트워크의 안정성 및 효율성을 보장 받을 수 있음을 확인 할 수 있었다.

또한, K 값의 변화에 따라 인증서 조각을 생성하는데 요구되는 시간과 완전한 인증서를 생성하는데 요구되는 시간을 측정한 결과, Partial Certificate들은 연합 멤버들에 의해 일정하게 계산되기 때문에 매개 변수 K가 시스템 효율에 영향을 미치지 않은 것을 발견할 수 있으며, 특히 Partial Certificate를 요구하는 측면에서 Certificate를 생성하는 모든 작동이 이루어지므로 오버헤드 계산에 관해서 적절한 결과를 보여주었다. 아울러, 본 논문의 연구 결과는 CA로부터 부분 서명키를 사전에 분배 받거나 중앙 집중적인 인증기관에 의한 문제를 완전 해결하였으며, 이는 중앙 CA에 의존하지 않는 MANET에 적합한 모델로 사료된다. 또한 새로 노드가 도메인 내의 한 클러스터 영역에 진입했을 경우에도, 원활한 인증서 발급 서비스를 수행하여 통신하고 있는 기존의 노드들이 새로운 노드의 유입에 영향을 받지 않고 능동적으로 인증 서비스를 수행할 수 있도록 높은 확장성을 지원할 수 있었다.

[참고문헌]

[1] Charles E. Perkins, "Ad Hoc Networking," *Addison Wesley*, 2001.

[2] Mark Weiser. "The Computer for the 21st Century," *Scientific American. 265*:pp.94-104. 1991

[3] 박춘식. "유비쿼터스 네트워크와 시큐리티 고찰," 『정보보호학회지』, *제14권 제1호*.pp14. 2004.02.

[4] J.Kong. p.Zerfos. H.Luo. S.Lu and L.Zhang. "Providing robust and ubiquitous security support for mobile Ad-Hoc networks". *IEEE ICNP.2001.*

[5] Y. Zhang and W. Lee. "intrusion detection in wireless ad-hoc network." *ACM Mobicom*. Aug. 2000

[6] L.Zhou and Z. Hass. "Securing ad hoc networks." *IEEE Network*. pp. 24-30. Nov/Dec. 1999

[7] D. Johnson, D, Maltz, Y-C. Hu, and J. Jetcheva. The dynamic source routing protocol for mobile ad hoc network. *IEEE* Internet Draft, March 2001. draft-ietf-manet=dsr-05.txt(work in progress)

[8] E.M. Belding-Royer and C,-K. Tho. A review of corrent routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications Magazine*, Pages 46-55, April 1999.

[9] C.E Perkins and E. M. Royer. "Ad hoc on-demand distance vector routing". In *IEEE* Workshop on Mobile Computing Systems and Applications, pages 90-100, Feb. 1999.

[10] S. Murthy and J.J. Garcia-Lunca-Aceves. "An efficient routing protocol for wireless networks". *ACM Mobile Networks and Applications Journal*, pages 183-197, Oct. 1996

[11] E. M Royer, C-K Toh "A review of Current Routing Protocols for Ad Hoc Mobile Wireless Network", *IEEE Personal Communication*, pp.46-55, April, 1999.

[12] David B. Johnson. "Routing in ad hoc networks of mobile hosts", In Proceedings of the *IEEE* Workshop on Mobile Computing Systems and Applications, pp.158-163, December, 1994.

[13] J. Broch, D. A. Maltz, D. B Johnson, Y-C. Hu, and J.Jetcheva. "A performance comparison of multi-hop wireless ad hoc network routing protocols", *In Proceedings of the 4th International Conference on Mobile Computing and Networking*, pp.85-97, October, 1998.

[14] Sander van Valkenburg, Asko Vilavaara and Ramjee Prasad, "The Implementation of a Mobile Ad-Hoc Networking Routing Protocol", *Proc. of WPMC'99*, pp.324-330, September, 1999.

[15] W.J Choi and Tekinary, "An Efficient Table Driven Routing Algorithm for Wireless ad Hoc Networks", *IEEE VTC'01*, Vol. *17*, pp.1395-1414, August, 1999..

[16] Zygmunt. J. Hass & Marc.r.Pearlman "The Zone Routing Protocol(ZRP) for Ad-Hoc Networks" Internet-Draft, draft-ietf-manet -zone-zrp-04.txt, July, 2002

[17] [http://www.ietf.org], [http://www.itfind.or.kr]

[18] Lidong Zhou, Zygmunt J. Haas, “Securing Ad Hoc Networks,” *IEEE Network* 1999.

[19] Y. Desmedt, Y. Frankel, "Threshold Cryptosystems", Proceedings on Advances in cryptology. *Lecture Notes in Computer Science*, pp.307-315. 1989.

[20] Y. Desmedt and Y. Frankel, "Threshold Cryptosystems", *Advances in Cryptology-Crypto* '89, (*Lecture Notes in Computer Science* **435**), G. Brassard, Ed., Springer Verlag, May 1990, pp. 307-315

[21] Y. Desmedt, "Threshold Cryptography," *Euro. Trans. Telecommunication*, vol. 5, no. 4, July-Aug. 1994, pp449-457

[22] Adi, Shamir, "Hoe to share a secret", *Communications of the ACM*, Vol22, pp612-313.

[23] Douglas, R. Stinson, Cryptography-Theory and Practice, *CRC Press*, 1995.

[24] R. Gennaro et al., "Robust Threshold DSS Signatures", *Advances in Cryptography-Proc. Eurocrypt '96* (*Lecture Notes in Computer Science* ***1070***), Springer Verlag, May 1996, pp. 354-171

[25] R. Gennaro et al., "Robust and Efficient Sharing of RSA functions", *Advances in Cryptography - Crypto '96 (Lecture Notes in Computer Science* ***1109)***, Springer Verlag, Aug 1996, pp. 157-172

[26] S.Jarecki, "proactive Secret Sharing and Public Key Cryptosystems," *Master's Thesis, Dept. Elec. Eng. and Comp. Sci.,* MIT, Sept. 1995

[27] A. Herzberg et al., "Proactive Secret Sharing or: How to Cope with Perpetual Leakage," *Advancer in Cryptology-Crypto '95 (Lecture Notes in Computer Science* ***963)***, D. Coppersmith, Ed., Aug. 1995, Springer-Verlag, pp. 457-469.

[28] A. Herzber et al., "Proactive Public-Key and Signature Schemes," *Proc. 4th Annual Cont. Comp. Commun. Security*, 1997, pp. 100-110.

[29] Y. Frankel et al., "Proactive RSA," *Advances in Cryptology - Crypto '97 (lecture Note in Computer Science* ***1294)***, Springer-Verlag, U.S.A., Aug. 1997, pp. 440-454

[30] Y. Frankel et al., "Optimal Resilience Proactive Public-Key Cryptosystem," *Proc, 38th Symp. Foundations of Comp. Sci.*, 1997.

[31] H. Luo and S. Lu. "Ubiquitous and Robust Authentication Services for Ad Hoc Wireless Network," *Technical Report TR-200030, Dept. of Computer Science*, UCLA, 2000.

[32] J. Kong. P.Zerfos, H.Luo. S. Lu, L.Zhang. "Providing Robust and

Ubiquitous Security Support for Mobile Ad-Hoc Networks", *IEEE Computer Society, Proceedings of the Ninth International Conference on Network Protocols (ICNP'01)*, PP. 251. 2001.

[33] A. Herzberg, M. jakobsson, S. Jarecki, H. Krawczyk, and M.Yung, "Proactive public-key and signature schemes.", In proceedings of the Fourth Annual Conference on computer Communications Security", *ACM*, 1997. pp. 100-120.

[34] A. Herzberg, S. Jarecki, H. Krawczyk, and M.Yung, "Proactive secret sharing or : How to cope with perpetual leakage", *Advances in Cryptolgy-Crypto '95*, Santa Barbara, Califonia, U.S.A, Aug. 1995, pp.457-469.

부록 I

```
s 0.475907155 _4_ MAC  --- 0 message 84 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
s 0.475907155 _4_ MAC  --- 0 message 84 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
r 0.476579519 _2_ MAC  --- 0 message 32 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
s 0.626786489 _2_ MAC  --- 1 message 84 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
r 0.627458853 _4_ MAC  --- 1 message 32 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
r 0.627459287 _3_ MAC  --- 1 message 32 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
s 0.886998331 _0_ MAC  --- 2 message 84 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.887671034 _1_ MAC  --- 2 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.887671163 _3_ MAC  --- 2 message 32 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
s 1.601893317 _1_ MAC  --- 3 message 84 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
r 1.602566020 _0_ MAC  --- 3 message 32 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
s 1.638268262 _3_ MAC  --- 4 message 84 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 1.638941060 _2_ MAC  --- 4 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 1.638941094 _0_ MAC  --- 4 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
M 10.00000 1 (126.00, 353.00, 0.00), (20.00, 218.00), 31.00
M 10.00000 2 (525.00, 110.00, 0.00), (320.00, 18.00), 71.00
M 10.00000 3 (559.00, 347.00, 0.00), (150.00, 318.00), 101.00
s 12.479211113 _2_ MAC  --- 5 message 84 [0 ffffffff
r 12.479883552 _4_ MAC  --- 5 message 32
```

```
                                        [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
                                      44 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
s 39.987299080 _0_ MAC  --- 35 message 120 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
r 39.988259763 _3_ MAC  --- 35 message 68 [0 ffffffff 0 800] ------- [0:255 -1:255 32 0]
s 41.000000000 _3_ AGT  --- 36 tcp 40 [0 0 0 0] ------- [3:1 4:0 32 0] [0 0] 0 0
s 41.783556992 _3_ MAC  --- 37 message 108 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 41.784421539 _1_ MAC  --- 37 message 56 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 41.784421675 _0_ MAC  --- 37 message 56 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
s 42.568552506 _1_ MAC  --- 38 message 108 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
r 42.569417052 _3_ MAC  --- 38 message 56 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
s 43.000000000 _1_ AGT  --- 39 tcp 40 [0 0 0 0] ------- [1:1 2:0 32 0] [0 0] 0 0
s 43.000315000 _1_ MAC  --- 0 RTS 44 [5fe 3 1 0]
r 43.000667547 _3_ MAC  --- 0 RTS 44 [5fe 3 1 0]
s 43.000677547 _3_ MAC  --- 0 CTS 38 [4c4 1 0 0]
r 43.000982093 _1_ MAC  --- 0 CTS 38 [4c4 1 0 0]
s 43.000992093 _1_ MAC  --- 39 tcp 112 [13a 3 1 800] ------- [1:1 2:0 32 3] [0 0] 0 0
r 43.001888640 _3_ MAC  --- 39 tcp 60 [13a 3 1 800] ------- [1:1 2:0 32 3] [0 0] 1 0
s 43.001898640 _3_ MAC  --- 0 ACK 38 [0 1 0 0]
D 43.001913640 _3_ IFQ  ARP 29 tcp 60 [13a 3 3 800] ------- [1:1 2:0 31 2] [0 0] 1 0
r 43.002203187 _1_ MAC  --- 0 ACK 38 [0 1 0 0]
s 43.002772640 _3_ MAC  --- 0 ARP 80 [0 ffffffff 3 806] ------- [REQUEST 3/3 0/2]
r 43.003413187 _1_ MAC  --- 0 ARP 28 [0 ffffffff 3 806] ------- [REQUEST 3/3 0/2]
r 43.003413323 _0_ MAC  --- 0 ARP 28 [0 ffffffff 3 806] ------- [REQUEST 3/3 0/2]
s 45.636953668 _3_ MAC  --- 40 message 84 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 45.637626215 _1_ MAC  --- 40 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 45.637626352 _0_ MAC  --- 40 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
s 45.889288973 _1_ MAC  --- 41 message 96 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
r 45.890057520 _3_ MAC  --- 41 message 44 [0 ffffffff 1 800] ------- [1:255 -1:255 32 0]
s 46.633072813 _3_ MAC  --- 42 message 96 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 46.633841360 _1_ MAC  --- 42 message 44 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 46.633841496 _0_ MAC  --- 42 message 44 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
s 46.644361359 _2_ MAC  --- 43 message 96 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
r 46.645129934 _4_ MAC  --- 43 message 44 [0 ffffffff 2 800] ------- [2:255 -1:255 32 0]
s 48.000000000 _2_ AGT  --- 44 tcp 40 [0 0 0 0] ------- [2:1 3:0 32 0] [0 0] 0 0
s 48.795212144 _4_ MAC  --- 45 message 108 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
r 48.796076719 _2_ MAC  --- 45 message 56 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
M 50.00000 0 (323.00, 428.00, 0.00), (125.00, 20.00), 15.00
D 50.000000000 _0_ IFQ  END 33 tcp 60 [0 0 0 800] ------- [0:0 1:0 32 1] [0 0] 0 0
D 50.000000000 _2_ IFQ  END 32 tcp 60 [0 0 2 800] ------- [2:1 3:0 32 3] [0 0] 0 0
D 50.000000000 _3_ IFQ  END 39 tcp 60 [13a 3 3 800] ------- [1:1 2:0 31 2] [0 0] 1 0
```

【그림 39】 node 5의 trace

```
s 0.086805504 _4_ MAC  --- 0 message 84 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
r 0.087477868 _2_ MAC  --- 0 message 32 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
r 0.087477890 _9_ MAC  --- 0 message 32 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
r 0.087478054 _7_ MAC  --- 0 message 32 [0 ffffffff 4 800] ------- [4:255 -1:255 32 0]
s 0.232218354 _6_ MAC  --- 1 message 84 [0 ffffffff 6 800] ------- [6:255 -1:255 32 0]
r 0.232890486 _1_ MAC  --- 1 message 32 [0 ffffffff 6 800] ------- [6:255 -1:255 32 0]
r 0.232890954 _0_ MAC  --- 1 message 32 [0 ffffffff 6 800] ------- [6:255 -1:255 32 0]
s 0.337526486 _3_ MAC  --- 2 message 84 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 0.338199060 _5_ MAC  --- 2 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 0.338199168 _7_ MAC  --- 2 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 0.338199284 _2_ MAC  --- 2 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
r 0.338199318 _0_ MAC  --- 2 message 32 [0 ffffffff 3 800] ------- [3:255 -1:255 32 0]
s 0.358556573 _9_ MAC  --- 3 message 84 [0 ffffffff 9 800] ------- [9:255 -1:255 32 0]
r 0.359228959 _4_ MAC  --- 3 message 32 [0 ffffffff 9 800] ------- [9:255 -1:255 32 0]
r 0.359229057 _7_ MAC  --- 3 message 32 [0 ffffffff 9 800] ------- [9:255 -1:255 32 0]
r 0.359229094 _2_ MAC  --- 3 message 32 [0 ffffffff 9 800] ------- [9:255 -1:255 32 0]
r 0.359229111 _8_ MAC  --- 3 message 32 [0 ffffffff 9 800]
s 0.622488306 _1_ MAC  --- 4 message 84 [0 ffffffff 1 800]
r 0.623160439 _6_ MAC  --- 4 message 32
```

```
                                    [13a 4 2 800] ------ [5:0 4:1 31 4] [3 0] 1 0
r 49.973130557 _4_ MAC  --- 1873 ack 60 [13a 4 2 800] ------ [5:0 4:1 31 4] [3 0] 2 0
s 49.973140557 _4_ MAC  --- 0 ACK 38 [0 2 0 0]
r 49.973155557 _4_ AGT  --- 1873 ack 60 [13a 4 2 800] ------ [5:0 4:1 31 4] [3 0] 2 0
s 49.973155557 _4_ AGT  --- 1881 tcp 1040 [0 0 0 0] ------ [4:1 5:0 32 0] [5 0] 0 0
s 49.973155557 _4_ AGT  --- 1882 tcp 1040 [0 0 0 0] ------ [4:1 5:0 32 0] [6 0] 0 0
r 49.973445132 _2_ MAC  --- 0 ACK 38 [0 2 0 0]
s 49.973554557 _4_ MAC  --- 0 RTS 44 [253e 9 4 0]
r 49.973906943 _9_ MAC  --- 0 RTS 44 [253e 9 4 0]
s 49.973916943 _9_ MAC  --- 0 CTS 38 [2404 4 0 0]
r 49.974221329 _4_ MAC  --- 0 CTS 38 [2404 4 0 0]
s 49.974231329 _4_ MAC  --- 1879 tcp 1112 [13a 9 4 800] ------ [4:1 5:0 32 9] [4 0] 0 0
r 49.983127715 _9_ MAC  --- 1879 tcp 1060 [13a 9 4 800] ------- [4:1 5:0 32 9] [4 0] 1
0s 49.983137715 _9_ MAC  --- 0 ACK 38 [0 4 0 0]
r 49.983442101 _4_ MAC  --- 0 ACK 38 [0 4 0 0]
s 49.983631715 _9_ MAC  --- 0 RTS 44 [253e 8 9 0]
r 49.983984252 _8_ MAC  --- 0 RTS 44 [253e 8 9 0]
s 49.983994252 _8_ MAC  --- 0 CTS 38 [2404 9 0 0]
r 49.984298790 _9_ MAC  --- 0 CTS 38 [2404 9 0 0]
s 49.984308790 _9_ MAC  --- 1879 tcp 1112 [13a 8 9 800] ------ [4:1 5:0 31 8] [4 0] 1 0
r 49.993205327 _8_ MAC  --- 1879 tcp 1060 [13a 8 9 800] ------ [4:1 5:0 31 8] [4 0] 2 0
s 49.993215327 _8_ MAC  --- 0 ACK 38 [0 9 0 0]
r 49.993519865 _9_ MAC  --- 0 ACK 38 [0 9 0 0]
s 49.993719679 _5_ MAC  --- 0 RTS 44 [253e 1 5 0]
r 49.994072212 _1_ MAC  --- 0 RTS 44 [253e 1 5 0]
s 49.994082212 _1_ MAC  --- 0 CTS 38 [2404 5 0 0]
r 49.994386746 _5_ MAC  --- 0 CTS 38 [2404 5 0 0]
s 49.994396746 _5_ MAC  --- 1875 tcp 1112 [13a 1 5 800] ------ [5:1 6:0 32 1] [5 0] 0 0
M 50.00000 0 (323.00, 428.00, 0.00), (125.00, 20.00), 15.00
M 50.00000 8 (289.00, 176.00, 0.00), (220.00, 48.00), 201.00
s 50.000000000 _6_ AGT  --- 1883 tcp 40 [0 0 0 0] ------ [6:1 7:0 32 0] [0 0] 0 0
D 50.000000000 _0_ IFQ  END 1237 tcp 60 [13a 0 0 800] ------ [4:1 5:0 28 5] [0 0] 4 0
D 50.000000000 _0_ IFQ  END 45 tcp 60 [0 0 0 800] ------ [0:0 1:0 32 1] [0 0] 0 0
D 50.000000000 _2_ IFQ  END 898 tcp 60 [0 0 2 800] ------ [2:1 3:0 32 3] [0 0] 0 0
D 50.000000000 _3_ IFQ  END 813 tcp 60 [0 0 3 800] ------ [3:1 4:0 32 7] [0 0] 0 0
D 50.000000000 _4_ IFQ  END 1882 tcp 1060 [0 9 4 800] ------ [4:1 5:0 32 9] [6 0] 0 0
D 50.000000000 _5_ IFQ  END 1876 tcp 1060 [0 1 5 800] ------ [5:1 6:0 32 1] [6 0] 0 0
D 50.000000000 _5_ IFQ  END 1877 ack 60 [13a 3 5 800] ------ [4:0 3:1 30 3] [159 0] 2 0
D 50.000000000 _5_ IFQ  END 1878 ack 60 [13a 3 5 800] ------ [4:0 3:1 30 3] [159 0] 2 0
D 50.000000000 _8_ IFQ  END 1498 ack 60 [13a 8 8 800] ------- [2:0 1:1 31 1] [0 0] 1 0
```

【그림 40】 node 10의 trace

```
s 0.197427145 _0_ MAC  --- 0 message 84 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099293 _19_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099365 _13_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099680 _10_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099744 _6_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099847 _1_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099923 _14_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099952 _11_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099972 _5_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
r 0.198099976 _3_ MAC  --- 0 message 32 [0 fffffff 0 800] ------- [0:255 -1:255 32 0]
s 0.219395970 _17_ MAC  --- 1 message 84 [0 fffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.220068124 _2_ MAC  --- 1 message 32 [0 fffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.220068191 _4_ MAC  --- 1 message 32 [0 fffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.220068316 _7_ MAC  --- 1 message 32 [0 fffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.220068367 _9_ MAC  --- 1 message 32 [0 fffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.220068503 _11_ MAC  --- 1 message 32 [0 fffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.220068751 _8_ MAC  --- 1 message 32 [0 fffffff 11 800] -------
s 0.376316519 _19_ MAC  --- 2 message 84 [0 fffffff 13 800]
r 0.376988667 _0_ MAC  --- 2 message 32 [0
```

```
                                        ------ [0:0 1:0 32 3] [392 0] 0 0
                                   800] ------ [0:0 1:0 32 3] [393 0] 0 0
                                   800] ------ [0:0 1:0 32 3] [394 0] 0 0
D 50.000000000 _0_ IFQ  END 1422 tcp 1060 [0 3 0 800] ------ [0:0 1:0 32 3] [395 0] 0 0
D 50.000000000 _0_ IFQ  END 1444 tcp 1060 [0 3 0 800] ------ [0:0 1:0 32 3] [396 0] 0 0
D 50.000000000 _0_ IFQ  END 1448 tcp 1060 [0 3 0 800] ------ [0:0 1:0 32 3] [397 0] 0 0
D 50.000000000 _0_ IFQ  END 144 tcp 60 [0 0 0 800] ------ [0:0 1:0 32 1] [0 0] 0 0
D 50.000000000 _2_ IFQ  END 1431 tcp 1060 [0 5 2 800] ------ [2:1 3:0 32 5] [4 0] 0 0
D 50.000000000 _2_ IFQ  END 188 tcp 60 [0 0 2 800] ------ [2:1 3:0 32 3] [0 0] 0 0
D 50.000000000 _3_ IFQ  END 497 tcp 60 [0 0 3 800] ------ [3:1 4:0 32 11] [0 0] 0 0
D 50.000000000 _4_ IFQ  END 1441 tcp 1060 [0 d 4 800] ------ [4:1 5:0 32 13] [18 0] 0 0
D 50.000000000 _5_ IFQ  END 1424 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [175 0] 0 0
D 50.000000000 _5_ IFQ  END 1427 ack 60 [0 2 5 800] ------ [5:0 4:1 32 2] [14 0] 0 0
D 50.000000000 _5_ IFQ  END 1438 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [176 0] 0 0
D 50.000000000 _5_ IFQ  END 1440 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [177 0] 0 0
D 50.000000000 _5_ IFQ  END 1445 ack 60 [0 2 5 800] ------ [5:0 4:1 32 2] [15 0] 0 0
D 50.000000000 _5_ IFQ  END 1446 ack 60 [0 2 5 800] ------ [5:0 4:1 32 2] [16 0] 0 0
D 50.000000000 _5_ IFQ  END 1447 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [178 0] 0 0
D 50.000000000 _6_ IFQ  END 1399 ack 60 [13a 0 6 800] ------ [1:0 0:0 31 0] [378 0] 1 0
D 50.000000000 _6_ IFQ  END 1402 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [160 0] 0 0
D 50.000000000 _6_ IFQ  END 1404 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [161 0] 0 0
D 50.000000000 _6_ IFQ  END 1408 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [162 0] 0 0
D 50.000000000 _6_ IFQ  END 1412 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [163 0] 0 0
D 50.000000000 _6_ IFQ  END 1414 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [164 0] 0 0
D 50.000000000 _6_ IFQ  END 1416 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [165 0] 0 0
D 50.000000000 _6_ IFQ  END 1406 ack 60 [13a 0 6 800] ------ [1:0 0:0 31 0] [379 0] 1 0
D 50.000000000 _6_ IFQ  END 1420 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [166 0] 0 0
D 50.000000000 _6_ IFQ  END 1409 ack 60 [13a 0 6 800] ------ [1:0 0:0 31 0] [380 0] 1 0
D 50.000000000 _6_ IFQ  END 1425 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [167 0] 0 0
D 50.000000000 _6_ IFQ  END 1418 ack 60 [13a 0 6 800] ------ [1:0 0:0 31 0] [381 0] 1 0
D 50.000000000 _6_ IFQ  END 1428 ack 60 [13a 0 6 800] ------ [1:0 0:0 31 0] [382 0] 1 0
D 50.000000000 _6_ IFQ  END 1430 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [168 0] 0 0
D 50.000000000 _6_ IFQ  END 1435 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [169 0] 0 0
D 50.000000000 _6_ IFQ  END 1436 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [170 0] 0 0
D 50.000000000 _6_ IFQ  END 1429 ack 60 [13a 0 6 800] ------ [1:0 0:0 31 0] [383 0] 1 0
D 50.000000000 _6_ IFQ  END 1437 ack 60 [13a 0 6 800] ------ [1:0 0:0 31 0] [384 0] 1 0
D 50.000000000 _6_ IFQ  END 1439 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [171 0] 0 0
D 50.000000000 _6_ IFQ  END 1442 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [172 0] 0 0
D 50.000000000 _6_ IFQ  END 1443 ack 60 [0 5 6 800] ------ [6:0 5:1 32 5] [173 0] 0 0
D 50.000000000 _6_ IFQ  END 1449 tcp 60 [0 0 6 800] ------ [6:1 7:0 32 10] [0 0] 0 0
D 50.000000000 _11_ IFQ  END 668 tcp 60 [13a b b 800] ------ [1:1 2:0 31 2] [0 0] 1 0
```

【그림 41】node 20의 trace

```
s 0.029821547 _15_ MAC  --- 0 message 84 [0 ffffffff f 800] ------- [15:255 -1:255 32 0]
r 0.030493702 _5_ MAC  --- 0 message 32 [0 ffffffff f 800] ------- [15:255 -1:255 32 0]
r 0.030493718 _22_ MAC  --- 0 message 32 [0 ffffffff f 800] ------- [15:255 -1:255 32 0]
r 0.030493804 _20_ MAC  --- 0 message 32 [0 ffffffff f 800] ------- [15:255 -1:255 32 0]
r 0.030493987 _21_ MAC  --- 0 message 32 [0 ffffffff f 800] ------- [15:255 -1:255 32 0]
r 0.030494245 _19_ MAC  --- 0 message 32 [0 ffffffff f 800] ------- [15:255 -1:255 32 0]
r 0.030494268 _3_ MAC  --- 0 message 32 [0 ffffffff f 800] ------- [15:255 -1:255 32 0]
s 0.062259317 _27_ MAC  --- 1 message 84 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931478 _28_ MAC  --- 1 message 32 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931588 _9_ MAC  --- 1 message 32 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931613 _8_ MAC  --- 1 message 32 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931718 _24_ MAC  --- 1 message 32 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931773 _11_ MAC  --- 1 message 32 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931845 _12_ MAC  --- 1 message 32 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931857 _7_ MAC  --- 1 message 32 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931928 _17_ MAC  --- 1 message 32 [0 ffffffff 1b 800] ------- [27:255 -1:255 32 0]
r 0.062931966 _10_ MAC  --- 1 message 32 [0 ffffffff 1b 800] -------
r 0.062931967 _4_ MAC  --- 1 message 32 [0 ffffffff 1b 800]
r 0.062931974 _14_ MAC  --- 1 message 32
                                                    --- [4:1 5:0 32 20] [29 0] 0 0
                                                  ----- [4:1 5:0 32 20] [30 0] 0 0
                                          [0 6 5 800] ----- [5:1 6:0 32 6] [111 0] 0 0
D 50.000000000 _5_ IFQ  END 2456 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [112 0] 0 0
D 50.000000000 _5_ IFQ  END 2462 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [113 0] 0 0
D 50.000000000 _5_ IFQ  END 2436 tcp 1060 [13a 3 5 800] ------- [2:1 3:0 31 3] [181 0]
1 0D 50.000000000 _5_ IFQ  END 2465 tcp 1060 [0 6 5 800] ---- [5:1 6:0 32 6] [114 0] 0 0
D 50.000000000 _5_ IFQ  END 2467 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [115 0] 0 0
D 50.000000000 _5_ IFQ  END 2470 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [116 0] 0 0
D 50.000000000 _5_ IFQ  END 2476 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [117 0] 0 0
D 50.000000000 _5_ IFQ  END 2440 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [182 0] 1 0
D 50.000000000 _5_ IFQ  END 2480 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [118 0] 0 0
D 50.000000000 _5_ IFQ  END 2483 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [119 0] 0 0
D 50.000000000 _5_ IFQ  END 2484 tcp 1060 [0 6 5 800] ------- [5:1 6:0 32 6] [120 0] 0
0D 50.000000000 _5_ IFQ  END 2489 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [121 0] 0 0
D 50.000000000 _5_ IFQ  END 2446 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [183 0] 1 0
D 50.000000000 _5_ IFQ  END 2449 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [184 0] 1 0
D 50.000000000 _5_ IFQ  END 2499 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [122 0] 0 0
D 50.000000000 _5_ IFQ  END 2501 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [123 0] 0 0
D 50.000000000 _5_ IFQ  END 2472 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [185 0] 1 0
D 50.000000000 _5_ IFQ  END 2485 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [186 0] 1 0
D 50.000000000 _5_ IFQ  END 2486 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [187 0] 1 0
D 50.000000000 _5_ IFQ  END 2493 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [188 0] 1 0
D 50.000000000 _5_ IFQ  END 2509 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [124 0] 0 0
D 50.000000000 _5_ IFQ  END 2505 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [189 0] 1 0
D 50.000000000 _5_ IFQ  END 2513 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [125 0] 0 0
D 50.000000000 _5_ IFQ  END 2506 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [190 0] 1 0
D 50.000000000 _5_ IFQ  END 2517 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [126 0] 0 0
D 50.000000000 _5_ IFQ  END 2511 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [191 0] 1 0
D 50.000000000 _5_ IFQ  END 2514 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [192 0] 1 0
D 50.000000000 _5_ IFQ  END 2515 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [193 0] 1 0
D 50.000000000 _5_ IFQ  END 2525 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [127 0] 0 0
D 50.000000000 _5_ IFQ  END 2529 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [128 0] 0 0
D 50.000000000 _5_ IFQ  END 2532 tcp 1060 [0 6 5 800] ----- [5:1 6:0 32 6] [129 0] 0 0
D 50.000000000 _5_ IFQ  END 2519 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [194 0] 1 0
D 50.000000000 _5_ IFQ  END 2419 ack 60 [13a 5 5 800] ----- [2:0 1:1 31 16] [7 0] 1 0
D 50.000000000 _11_ IFQ  END 700 tcp 60 [13a b b 800] ----- [1:1 2:0 31 2] [0 0] 1 0
D 50.000000000 _16_ IFQ  END 0 ARP 28 [0 5 10 806] ------- [REPLY 16/16 5/5]
D 50.000000000 _16_ IFQ  END 0 ARP 28 [0 5 10 806] ------- [REPLY 16/16 5/5]
D 50.000000000 _23_ IFQ  END 2246 tcp 1060 [13a 1 17 800] --- [0:0 1:0 31 1] [690 0] 1 0
D 50.000000000 _23_ IFQ  END 2259 tcp 1060 [13a 1 17 800] --- [0:0 1:0 31 1] [692 0] 1 0
```

【그림 42】 node 30의 trace

```
s 0.029170082 _5_ MAC  --- 0 message 84 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842177 _22_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842226 _20_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842236 _15_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842464 _32_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842626 _21_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842648 _33_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842655 _3_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842775 _19_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842898 _13_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
r 0.029842909 _0_ MAC  --- 0 message 32 [0 ffffffff 5 800] ------- [5:255 -1:255 32 0]
s 0.063645768 _17_ MAC  --- 1 message 84 [0 ffffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.064317922 _2_ MAC  --- 1 message 32 [0 ffffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.064317989 _4_ MAC  --- 1 message 32 [0 ffffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.064318066 _39_ MAC  --- 1 message 32 [0 ffffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.064318114 _7_ MAC  --- 1 message 32 [0 ffffffff 11 800] ------- [17:255 -1:255 32 0]
r 0.064318165 _9_ MAC  --- 1 message 32 [0 ffffffff 11 800] -------
r 0.064318200 _24_ MAC  --- 1 message 32 [0 ffffffff
r 0.064318298 _28_ MAC  --- 1 message 32
```

```
                                        [0 6 5 800] ------ [5:1 6:0 32 6] [85 0] 0 0
D 50.000000000 _5_ IFQ  END 1723 tcp 1060 [13a 2 5 800] ------ [1:1 2:0 31 2] [21 0] 1 0
D 50.000000000 _5_ IFQ  END 1728 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [86 0] 0 0
D 50.000000000 _5_ IFQ  END 1733 ack 60 [0 2 5 800] ------ [5:0 4:1 32 2] [188 0] 0 0
D 50.000000000 _5_ IFQ  END 1724 tcp 1060 [13a 2 5 800] ------ [1:1 2:0 31 2] [22 0] 1 0
D 50.000000000 _5_ IFQ  END 1736 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [87 0] 0 0
D 50.000000000 _5_ IFQ  END 1725 tcp 1060 [13a 2 5 800] ------ [1:1 2:0 31 2] [23 0] 1 0
D 50.000000000 _5_ IFQ  END 1738 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [88 0] 0 0
D 50.000000000 _5_ IFQ  END 1747 ack 60 [0 2 5 800] ------ [5:0 4:1 32 2] [189 0] 0 0
D 50.000000000 _5_ IFQ  END 1730 tcp 1060 [13a 2 5 800] ------ [1:1 2:0 31 2] [24 0] 1 0
D 50.000000000 _5_ IFQ  END 1748 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [90 0] 0 0
D 50.000000000 _5_ IFQ  END 1749 ack 60 [0 2 5 800] ------ [5:0 4:1 32 2] [190 0] 0 0
D 50.000000000 _5_ IFQ  END 1750 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [91 0] 0 0
D 50.000000000 _5_ IFQ  END 1731 tcp 1060 [13a 2 5 800] ------ [1:1 2:0 31 2] [25 0] 1 0
D 50.000000000 _5_ IFQ  END 1753 ack 60 [0 2 5 800] ------ [5:0 4:1 32 2] [191 0] 0 0
D 50.000000000 _5_ IFQ  END 1754 ack 60 [13a 2 5 800] ------ [3:0 2:1 31 2] [287 0] 1 0
D 50.000000000 _5_ IFQ  END 1758 ack 60 [13a 2 5 800] ------ [3:0 2:1 31 2] [288 0] 1 0
D 50.000000000 _5_ IFQ  END 1761 ack 60 [0 2 5 800] ------ [5:0 4:1 32 2] [192 0] 0 0
D 50.000000000 _5_ IFQ  END 1767 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [93 0] 0 0
D 50.000000000 _5_ IFQ  END 1764 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [280 0] 1 0
D 50.000000000 _5_ IFQ  END 1765 tcp 1060 [13a 3 5 800] ------ [2:1 3:0 31 3] [281 0] 1
0D 50.000000000 _5_ IFQ  END 1769 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [94 0] 0 0
D 50.000000000 _5_ IFQ  END 1771 ack 60 [0 2 5 800] ------- [5:0 4:1 32 2] [194 0] 0 0
D 50.000000000 _5_ IFQ  END 1777 ack 60 [13a 2 5 800] ------ [3:0 2:1 31 2] [290 0] 1 0
D 50.000000000 _5_ IFQ  END 1778 tcp 1060 [0 6 5 800] ------- [5:1 6:0 32 6] [96 0] 0 0
D 50.000000000 _5_ IFQ  END 1779 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [282 0] 1 0
D 50.000000000 _5_ IFQ  END 1782 ack 60 [13a 2 5 800] ------ [3:0 2:1 31 2] [291 0] 1 0
D 50.000000000 _5_ IFQ  END 1781 tcp 1060 [13a 2 5 800] ------ [1:1 2:0 31 2] [26 0] 1 0
D 50.000000000 _5_ IFQ  END 1785 tcp 1060 [0 6 5 800] ------- [5:1 6:0 32 6] [97 0] 0 0
D 50.000000000 _5_ IFQ  END 1780 tcp 1060 [13a 3 5 800] ----- [2:1 3:0 31 3] [283 0] 1 0
D 50.000000000 _5_ IFQ  END 1788 tcp 1060 [0 6 5 800] ------ [5:1 6:0 32 6] [98 0] 0 0
D 50.000000000 _5_ IFQ  END 1787 ack 60 [13a 2 5 800] ------ [3:0 2:1 31 2] [292 0] 1 0
D 50.000000000 _5_ IFQ  END 1791 tcp 1060 [0 6 5 800] ------- [5:1 6:0 32 6] [99 0] 0 0
D 50.000000000 _5_ IFQ  END 1794 ack 60 [0 2 5 800] ------- [5:0 4:1 32 2] [197 0] 0 0
D 50.000000000 _6_ IFQ  END 1796 tcp 60 [0 0 6 800] ------- [6:1 7:0 32 10] [0 0] 0 0
D 50.000000000 _11_ IFQ  END 367 tcp 60 [13a b b 800] ------- [1:1 2:0 31 2] [0 0] 1 0
D 50.000000000 _28_ IFQ  END 365 tcp 60 [13a 1c 1c 800] ------- [2:1 3:0 31 3] [0 0] 1
0D 50.000000000 _30_ IFQ  END 979 ack 60 [13a 1e 1e 800] ----- [4:0 3:1 31 3] [0 0] 1 0
```

【그림 43】 node 40의 trace

부록 Ⅱ

1. tcl 파일의 구성(node 10인 경우)

```
##############################################################################
# Copyright (c) 2005 Artificial Intelligece Lab. of Chosun University.      #
# All rights reserved.                                                       #
##############################################################################

#===================================================================
# Define options / 10 mobilenodes
#===================================================================
set opt(chan)           Channel/WirelessChannel ;# channel type
set opt(prop)           Propagation/TwoRayGround ;# radio-propagation model
set opt(netif)          Phy/WirelessPhy ;# network interface type
set opt(mac)            Mac/802_11 ;# MAC type
set opt(ifq)            Queue/DropTail/PriQueue ;# interface queue type
set opt(ll)             LL ;# link layer type
set opt(ant)            Antenna/OmniAntenna ;# antenna model
set opt(ifqlen)         50 ;# max packet in ifq
set opt(adhocRouting) DSDV ;# routing protocol

set opt(x)              600 ;# x coordinate of topology
set opt(y)              600 ;# y coordinate of topology
set opt(seed)           0.0 ;# seed for random number gen.
set opt(stop)           50 ;# time to stop simulation

set opt(ftp-start)      20.0 ;# ftp 1 traffic start time
set opt(nn)             10;# number of mobilenodes

# create simulator instance
set ns_ [new Simulator]

# $ns_ use-newtrace
set tracefd [open no10.tr w]
set namtrace [open no10.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)

# Create topography object
set topo [new Topography]
```

```
# define topology(
$topo load_flatgrid $opt(x) $opt(y)

# create God
create-god [expr $opt(nn)]

# create channel(
set chan1 [new $opt(chan)]

# ==============================================================
# IEEE 802.11 parameter setting
# ==============================================================
Phy/WirelessPhy set CPThresh_ 10.0 ;# capture effect thresh
Phy/WirelessPhy set CSThresh_ 1.559e-11     ;# carrier sense : -78 dBm
Phy/WirelessPhy set RXThresh_ 3.652e-10     ;# Receiver sensitivity : -64
dBm
Phy/WirelessPhy set Pt_ 0.28183815   ;# Tx power 24.5 dBm
Phy/WirelessPhy set freq_ 2450e+6    ;# 2.45GHz
Phy/WirelessPhy set L_ 1.0

# configure for mobile node
$ns_ node-config -adhocRouting $opt(adhocRouting) \
        -llType $opt(ll) \
        -macType $opt(mac) \
        -ifqType $opt(ifq) \
        -ifqLen $opt(ifqlen) \
        -antType $opt(ant) \
        -propType $opt(prop) \
        -phyType $opt(netif) \
        -channel $chan1 \
        -topoInstance $topo\
        -agentTrace ON \
        -routerTrace OFF \
        -macTrace ON \
        -movementTrace OFF \
        -wiredRouting OFF \

#create node
for {set i 0} {$i < $opt(nn)} {incr i} {
        set node_([expr $i]) [$ns_ node]
        $node_([expr $i]) random-motion 0
}
```

```
set rng_x [new RNG]
set rng_y [new RNG]
$rng_x seed 7
$rng_y seed 200

for {set i 0} {$i < $opt(nn)} {incr i} {
	$node_($i) set X_ [$rng_x integer $opt(x)]
	$node_($i) set Y_ [$rng_y integer $opt(y)]
	$node_($i) set Z_ 0.0
	$node_($i) namattach $namtrace		;# enable namtrace
	$ns_ initial_node_pos $node_($i) 10 ;#node size in nam
	# print the position.
	puts "x($i)=[$node_($i) set X_]"
	puts "y($i)=[$node_($i) set Y_]"

}

# Node_(1) starts to move towards node_(0) at 15m/s
$ns_ at 50.0 "$node_(0) setdest 125.0 20.0 15.0"
$ns_ at 10.0 "$node_(1) setdest 20.0 218.0 31.0"
$ns_ at 10.0 "$node_(2) setdest 320.0 18.0 71.0"
$ns_ at 10.0 "$node_(3) setdest 150.0 318.0 101.0"
$ns_ at 30.0 "$node_(5) setdest 180.0 218.0 121.0"
$ns_ at 50.0 "$node_(8) setdest 220.0 48.0 201.0"
$ns_ at 80.0 "$node_(9) setdest 320.0 158.0 211.0"


########### setup traffics #############

for {set i 0} {$i < [expr $opt(nn) - 1]} {incr i} {
	set tcp_($i) [new Agent/TCP]
	$tcp_($i) set class_ 2
	set sink_($i) [new Agent/TCPSink]
	$ns_ attach-agent $node_($i) $tcp_($i)
	$ns_ attach-agent $node_([expr $i + 1]) $sink_($i)
	$ns_ connect $tcp_($i) $sink_($i)
	set ftp_($i) [new Application/FTP]
	$ftp_($i) attach-agent $tcp_($i)
	$ns_ at [expr $opt(ftp-start) + $i*5] "$ftp_($i) start"
}
```

```
# Tell all nodes when the simulation ends
for {set i 0 } {$i < $opt(nn) } {incr i} {
$ns_ at $opt(stop).0 "$node_($i) reset";
}
$ns_ at $opt(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop).0001 "stop"
proc stop {} {
global ns_ tracefd namtrace
$ns_ flush-trace
close $tracefd
close $namtrace
}

puts "Starting Simulation..."
puts "$opt(nn) nodes are created"
$ns_ run
```