석사 학위논문
Master's Thesis

S-RAY : Yen의 알고리즘을 응용한 네트워크 보안 라우팅

S-RAY : Secure Routing Algorithm
based on Yen's algorithm for a Network

정 제 성 (鄭 濟 成 Jeong, Je-Seong)
정보보호대학원
Graduate School of Information Security

KAIST

2016

S-RAY : Yen의 알고리즘을 응용한 네트워크 보안 라우팅

S-RAY : Secure Routing Algorithm
based on Yen's algorithm for a Network

S-RAY : Yen의 알고리즘을 응용한 네트워크 보안 라우팅

Major Advisor : Professor Kim, Kwangjo
Co - Advisor : Professor Shin, Seungwon

by

Jeong, Je-Seong
Graduate School of Information Security
KAIST

A thesis submitted to the faculty of KAIST in partial fulfillment of the requirements for the degree of Master of Science in Engineering in the Graduate School of Information Security. The study was conducted in accordance with Code of Research Ethics[1].

2015. 12. 4.
Approved by
Professor Kim, Kwangjo
[Major Advisor]

2015. 12. 4.
Approved by
Professor Shin, Seungwon
[Co－Advisor]

1) Declaration of Ethical Conduct in Research: I, as a graduate student of KAIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

S-RAY : Yen의 알고리즘을 응용한 네트워크 보안 라우팅

정 제 성

위 논문은 한국과학기술원 석사학위논문으로
학위논문심사위원회에서 심사 통과하였음.

2015 년 12 월 4 일

심사위원장 김 광 조 (인)

심사위원 김 명 철 (인)

심사위원 신 승 원 (인)

MIS
20143894

정 제 성. Jeong, Je-Seong. Secure Routing Algorithm based on Yen's algorithm. Yen's algorithm을 응용한 네트워크 라우팅. Graduate School of Information Security. 2016. 36p. Advisor Prof. Kim, Kwangjo, Co-Advisor Prof. Shin, Seungwon

ABSTRACT

Due to the rapid development of modern network, a variety of information are flourished throughout the Internet. However, the development of the network has caused some side-effects like DoS (Denial of Service) and MITM (Man In the Middle) attacks. While Random Route is typically used to prevent such attacks in wireless networks, the static route, which cannot prevent DoS or MITM, is popularly used in wired networks because of the difficulty to find disjoint route in wired networks. RRM (Random Route Mutation) has rarely discussed to apply Random Route to wired networks, also those ideas are not exercised in the conventional networks due to their inefficiency. In this paper, we propose S-RAY (Secure Routing Algorithm based on Yen's algorithm for a Network), which is an efficient and secure routing solution in both wired and wireless networks. S-RAY consists of three novel algorithms according to QoSS (Quality of Security Service), and the solution will be effective even in the government and military networks that require the dynamic level of security.

목 차

표 목 차

그 림 목 차

제 1 장 Introduction

1.1 연구 배경

1973년 최초 인터넷이 사용 된 이후로 현재까지 인터넷은 급속도록 발전 하였다. 현대 사회에서 우리는 인터넷을 통해 각종 정보를 얻을 수가 있으며, 다양한 업무를 수행하고 있다. 현재, 대한민국 인터넷 사용자는 2014년 기준 41,118천명으로 인구대비 83.6%가 사용을 하고 있다[1]. 그리고 이 수치는 스마트폰의 보급률이 증가함에 따라 더욱 증가 할 것으로 예상한다. 이와 같은 인터넷 사용이 가능 한 것은 네트워크 기술의 발전과 기간망이 확충되었기 때문이다. 과거에는 유선만 사용을 했다면 현재는 무선이 추가 되었으며, 과거에는 주요 도시 위주의 사용이었다면 현재는 전국 어디에서나 인터넷 사용이 가능해졌다. 하지만 이렇게 네트워크 망이 확충됨에 따라서 다양한 취약점이 발생했다. 이에 대한 피해 사례로는 2013년 3.20 사이버 테러 및 6.25 사이버 공격이 있었으며, 2009년 7 · 7 DDoS 공격과 2011년 3 · 4 DDoS 공격이 있었다. 이와 같은 피해가 발생하는 이유로는 첫째, 현대 대부분의 네트워크는 정보를 보낼 시 고정된 경로를 사용하기 때문이다. 둘째, 네트워크 규모가 커지는 것에 비해서 비용 상의 문제로 정보보호 장비를 모든 곳에 설치하는 것이 제한되기 때문이다. 이러한 이유로 공격자가 네트워크 경로에 대한 정보를 쉽게 얻거나 예측을 할 수가 있어 목표를 정하고 사이버 공격을 하는데 이점을 제공한다. 따라서, 무선 네트워크에서는 이를 방어하기 위해 데이터를 주고 받을 시 다양한 임의 경로로 데이터를 전송하여 재밍(Jamming)과 블랙홀 공격(Blackhole attack)을 막는 방법을 제안하였다[2][3]. 하지만, 이를 유선 네트워크에 적용하는 데는 다음과 같은 문제점이 있다. 첫째, Satisfiability Modulo Theories (SMT)를 적용하여 이미 사용이 된 노드나 링크를 다시 사용을 하면 안 되는 것, 둘째, 서비스 품질 측면에서 사용자가 불편이 없이 인터넷을 사용 할 수 있도록 하는 것이다. 따라서, 아직까지는 유선 네트워크에서는 고정 경로를 사용함으로써 서비스 거부 공격(Denial of Service, DoS)[4] 또는 중간자 공격(Man In The Middle, MITM)[5], 블랙홀 공격(Blackhole attack)[6]에 취약하므로 현실에 적용 가능하도록 효율성을 향상 시킨 임의 경로 선택 방식이 필요하다.

1.2 연구 내용 및 논문 구성

본 논문에서는 기존 임의 경로 선택 방식[7] 보다 효율성을 향상시킨 Yen 알고리즘(Yen's algorithm)을[8] 응용한 네트워크 보안 라우팅을 제안한다. 이는 Yen 알고리즘을 이용하여 경로를 구한 후 사용자 계층을 구분하여 이동 경로를 할당 한 후에 각 계층별로

임의 경로를 선택하여 데이터를 전송하는 방법이다. 이는 기존 네트워크에서 추가 비용 부담이 없이 적용 할 수 있는 방법으로 보안성과 효율성을 만족시키며 특히, 라우팅 정보 프로토콜(Random Information Protocol, RIP)[9] 또는 최단 경로 우선 프로토콜(Open Shortest Path First, OSPF)[10]을 사용하는 정부, 군, 경찰 또는 기업 내 인트라넷에 적용 시 효과를 볼 것으로 판단한다.

본 논문의 2장에서는 기존 네트워크 라우팅과 새로 적용을 하려는 보안 네트워크 라우팅에 대한 배경지식, 3장에서는 관련연구, 4장에서는 Yen의 알고리즘을 응용한 네트워크 보안 라우팅(Secure Routing Algorithm based on Yen's algorithm for a Network, S-RAY), 5장에서는 시뮬레이션, 6장에서는 평가, 7장에서는 활용방안, 마지막으로 8장에서는 결론을 제안한다.

제 2 장 배경 지식

2.1 기존 라우팅 방식

이번 장에서는 인트라넷에서 가장 많이 사용하는 IP 라우팅 프로토콜인 라우팅 정보 프로토콜(Routing Information Protocol, RIP)과 최단 경로 우선 프로토콜(Open Shortest Path First, OSPF)을 설명하겠다[6].

2.1.1. 라우팅 정보 프로토콜(Routing Information Protocol, RIP)

RIP는 중소 규모의 인터네트워크에서 라우팅 정보를 교환하기 위해 사용된다. 또한, 사용자 데이터그램 프로토콜(User Datagram Protocol, UDP) 상에서 동작을 하며, 경유 할 가능성이 있는 라우터를 홉 수로 수치화는 거리 벡터 알고리즘(Distance Vector Algorithm, DVA)을 사용한다[11]. 출발점부터 도착점까지 홉 카운트 제한을 두어 라우팅 루프를 방지하며, 이때 최대 숫자는 15이다. 이는 RIP가 지원 할 수 있는 네트워크 사이즈를 제한하며, 16 이상의 네트워크에서는 연결이 되지 않는다. 라우팅 테이블은 30초마다 업데이트 되며 방식은 다음과 같다. 처음에는 각 라우터의 라우팅 테이블에 실제로 연결 된 네트워크 만이 포함된다. RIP 라우터는 연결 할 수 있는 네트워크의 다른 로컬 라우터에 알릴 라우팅 테이블 항목이 들어 있는 알림을 주기적으로 보낸다. 이때, RIP 버전 1은 이 알림을 위해 인터넷 프로토콜(Internet Protocol, IP) 브로드캐스트 패킷을 사용한다. RIP 버전 2는 이 알림을 위해 멀티캐스트 또는 브로드캐스트 패킷을 사용한다[12].

RIP의 최대 장점은 구성과 배포가 간단하다는 것이며 단점은 대규모 인터네트워크로는 확장을 할 수 없다는 점이다.

2.1.2. 최단 경로 우선 프로토콜(Open Shortest Path First, OSPF)

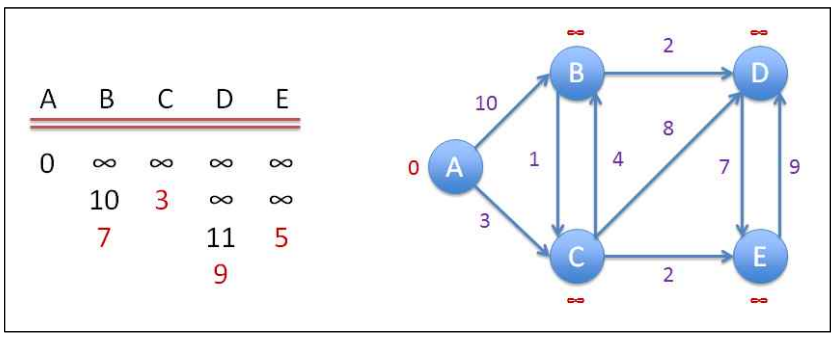
OSPF는 인터넷 프로토콜의 라우팅 프로토콜의 한 종류로서 RIP 보다 규모가 큰 대규모 자율 네트워크에서 가장 널리 쓰이는 내부 게이트웨이 프로토콜(Interior Gateway Protocol, IGP)이다[13]. TCP/IP 네트워크에서 라우팅 정보 프로토콜(Routing Information Protocol, RIP)의 단점을 개선하여 제어기능이 향상되었으며, 관리 정보 트래픽도 줄일 수 있다. OSPF는 라우팅 테이블 전체를 30초 마다 라우팅 테이블을 인접 호스트에 보내는 RIP와는 달리 변경이 발생 했을 때만, 변경 된 부분 만을 멀티캐스트 하며, 정보를 받는 즉시 라우팅 테이블은 다시 계산된다. OSPF는 SPF(Shortest Path Firtst) 알고리즘을 사용하

여 네트워크에서 가장 비용이 적게 드는 최단 경로를 계산한다[14].

OSPF의 장점은 네트워크 오버헤드가 아주 적어 효율적이라는 것이며 단점은 구성과 관리가 RIP에 비해 더 복잡하다.

2.2 다익스트라 알고리즘(Dijkstra's algorithm)

다익스트라 알고리즘은 어떤 변도 음수 값을 갖지 않는 유향 그래프에서 주어진 출발점과 도착점 사이의 최단 경로를 푸는 알고리즘이다[15]. 원리는 각각의 꼭짓점 v 에 대해 출발점 s 에서 각각의 v 까지의 최단 거리 $d[v]$를 저장하면서 작동한다. 시작 시 출발점은 $d[s]=0$이고, 다른 모든 꼭짓점 v 는 $d[v]=\infty$ 로 놓아 출발점 s 에서는 최단 경로를 모른다는 것을 표시한다. 이후, 출발점 s 와 인접한 꼭짓점들(u) 까지의 경로($d[u]$)를 구한 후 이를 저장한다. 이후 변 경감(edge relaxation)이라고 불리는 기본 연산을 바탕으로 u에서 인접한 v 까지 길이 $w(u, v)$를 측정하여 기존 s 부터의 경로 값 $d[v]$에 합($d[u]$ + $w(u, v)$)을 하여 최단 경로 값을 저장한다. 이와 같이 모든 점에 대해서 거리를 구한 후 최소 값을 저장한다. 만약 경로가 존재하지 않으면 거리는 무한대로 남는다. [그림 2.1] 다익스트라 알고리즘의 예이다. 출발점을 A라 하고 나머지 꼭짓점을 목적지로 하여 거리 값을 구한다. 최초에는 A는 0 나머지는 ∞로 저장한다. 이후 A와 인접한 B, C 값 10, 3을 입력 후에 다음으로 B와 인접한 C, D와 C와 인접한 B, D, E와의 경로를 구한 후에 A에서 B, C로 간 최소 값과 합을 구한 후 비교 후에 최소 값을 저장한다. 이와 같은 방식으로 지속 진행하면 결과는 B는 7, C는 3, D는 9, E는 5가 저장된다[16].



[그림 2.1] 다익스트라 알고리즘(Dijkstra's algorithm) 예[16]

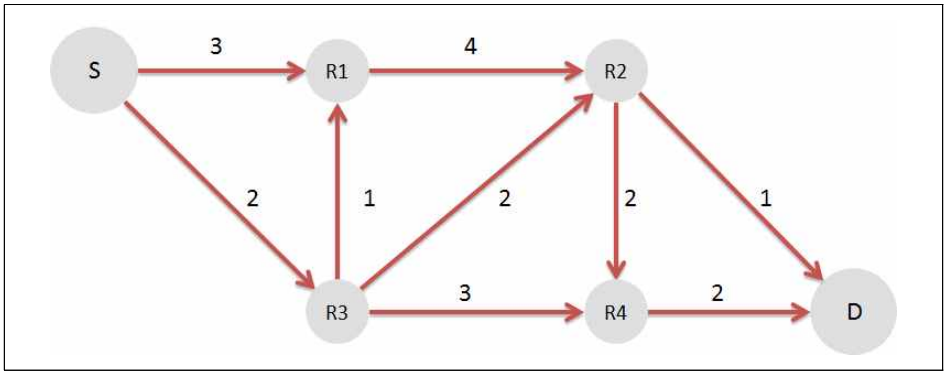
2.3 Yen 알고리즘(Yen's algorithm)

Yen 알고리즘은 어떤 변도 음수 값을 갖지 않는 그래프에서 고리를 형성하지 않는 K 개의 최소 값들을 가지는 경로를 구하는 알고리즘이다[17]. [표 2.1]은 Yen 알고리즘을 설명하기 위한 용어 설명이다.

[표 2.1] Yen 알고리즘 용어 및 표기법[18]

표 기	설 명
N	• 그래프 사이즈, 네트워크 내의 노드 총 개수.
(i)	• 그래프의 i^{th} 노드, 여기서 i 의 범위는 1부터 N 까지임. • (1)은 출발점, (N)은 도착점
A^k	• (1)부터 (N)까지의 k 번째 짧은 경로. k 범위는 1 부터 k 임. • A^k = (1) – (2^k) – (3^k) – (Q_k^k) – (N) 라 할 시, (2^k) 는 k 번째 짧은 경로의 두 번째 노드
A^k_i	• 노드 (i)에서 A^{k-1}와는 다른 경로, i 는 1부터 Q_k 까지 임. • A^k 와 A^{k-1} 의 경로는 i 번째 노드 까지는 같은 경로를 가짐. 이후 A^j 의 경로 $(i)^k$ – $(i+1)^k$ 는 기존의 경로와는 다름.
R^k_i	• A^k_i 의 기본 경로는 A^{k-1}의 i 번째 노드 까지는 A^{k-1}를 따름.
S^k_i	• A^k_i 의 차후 경로는 A^k_i의 i^{th} 노드에서 시작해서 도착점에서 끝남.

Yen 알고리즘은 두 부분으로 나뉘어 진다. 첫 번째는 검증 된 최단 경로들의 집합(A 집합)이고 두 번째는 후보 최단 경로 집합(B 집합)이다. 먼저, 출발점으로부터 도착점까지의 최단 경로를 다익스트라 알고리즘을 이용해서 구한다. 이를 A^1이라 한다. 이후, A^1을 이용해서 K 개의 경로를 구한다. A^k 경로를 찾기 위해서는 두 개의 과정이 이루어지는데 첫 번째는 A_i^k 의 모든 다른 경로를 찾는 것이고, 두 번째는 A^k가 될 최적 경로를 찾는 것이다. 이러한 과정은 목적지 노드 전까지 이루어진다. 먼저, 앞에서 말한 첫 번째 과정은 다음과 같이 이뤄진다. A^{k-1} 의 경로에서 출발점과 도착점을 제외한 모든 노드에서 R^k_i 를 정한 이후 기존에 이용했던 경로는 무한대로 설정하고 각각의 노드에서 모든 S^k_i 를 찾게 된다. 다음으로 R^k_i 와 S^k_i 를 각 노드별로 더하여 A^k_i 를 구한 후, 이를 B 그룹에 포함시킨다. 이후 두 번째 과정에서 B 그룹에서의 최소 값을 찾아 A 그룹에 포함을 시킨 후에 B 그룹에서 삭제하는 것을 반복한다[19]. [그림 2.2]은 Yen 알고리즘을 그림으로 표현한 것이다. 먼저 출발점 S로부터 도착점 D까지의 최적 경로 A^1(S – R3 – R2 – D)을 구한 후에 이를 이용하여 A^2(S – R3 – R4 – D), A^3(S – R1 – R2 –D) 등을 구한다[18].

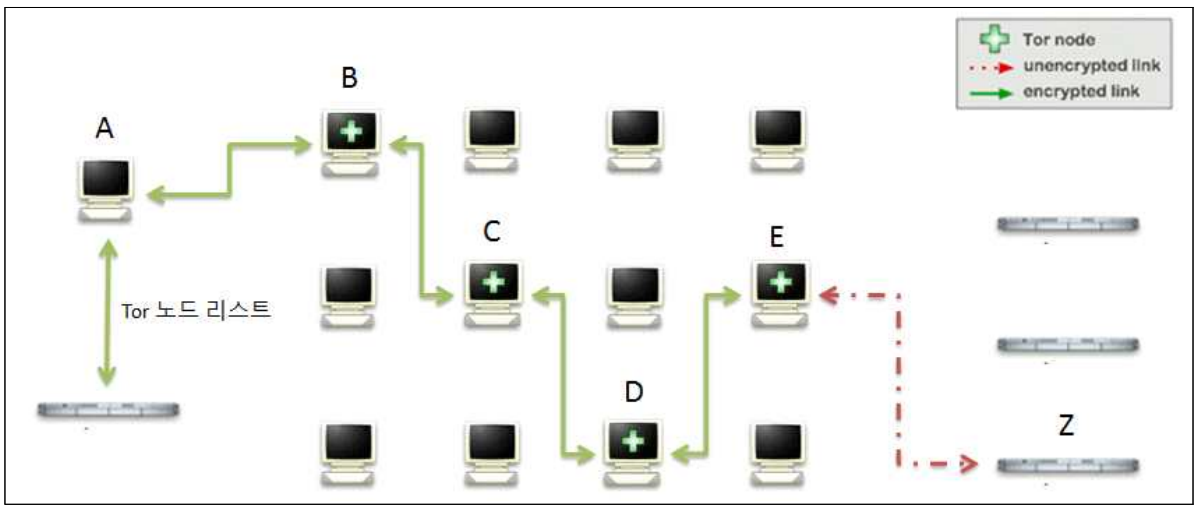


[그림 2.2] Yen 알고리즘(Yen's algorithm) 예[18]

제 3 장 관련 연구

3.1 토어(The Onion Router, Tor)

토어는 최초 미국 해군 연구소에서 시작이 되었으며, 트래픽 분석 공격으로부터 사용자 보호를 위해 사용된다. 토어는 전 세계에서 제공되는 프록시 서버를 이용하여 무작위로 경로를 선정한다. 또한, 각 구간 구간에 데이터를 전송 시에는 암호화 전송하여 출발점이 어디인지를 알 수가 없다[20][21]. [그림 1]은 토어의 데이터 전송 예이다. A에서 Z로 데이터를 전송 할 때 디렉토리 서버로부터 토어 노드 리스트를 받아 무작위로 전송 경로를 선정한다. 즉, A → Z 로 바로 전송하는 것이 아니라 A → B → C → D → E →Z 와 같이 여러 경로를 거쳐서 보낸다. 바로 이 때문에 전송 속도가 매우 느리다는 단점이 있다. 또한, 패킷을 암호화 하여 전송을 하고, 각각의 노드의 공개키를 통해 암호화[22] 하므로 패킷의 출발지와 목적지를 알려면 거의 모든 노드를 장악해야 하는데 이는 실질적으로 불가능 하다. 따라서 Z에서 봤을 시에는 E를 출발점으로 알 수 밖에 없다[23].

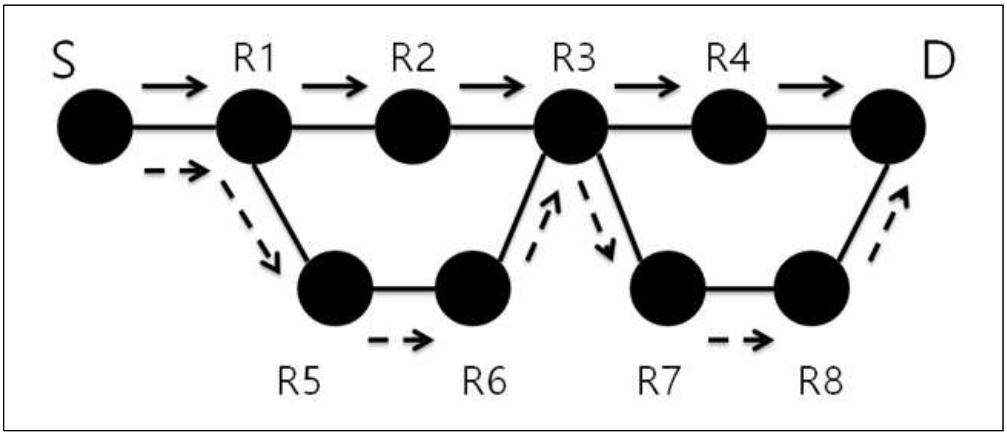


[그림 3.1] 토어(The Onion Router, Tor) 데이터 전송 예[23]

3.2 임의 경로 변화(Random Route Mutation, RRM)

RRM은 현재 유선 네트워크의 대부분이 고정 된 경로를 사용함에 따라 네트워크 공격자로부터 서비스 거부 공격, 중간자 공격 등을 받게 되어 이러한 공격을 방어하기 위해 제안되었다. 하지만 아직까지는 Satisfiability Modulo Theories (SMT)를 이용하여 이미 사용이 된 경로 또는 노드를 사용하지 않게 하는 문제점과 사용자 품질 측면에서 문제점을 가

지고 있다. RRM은 다음과 같은 절차를 따른다. 첫째, 최적화 된 경로 변화를 찾고, 둘째, 데이터 전송이 방해되지 않고, 보안 요구 조건을 만족 시킬 수 있는 경로 변화 순서를 확인한다. 마지막 세 번째로 RRM이 네트워크에서 일관성 있게 지속 되도록 하는 것이다. [그림 3.2]은 RRM의 예이다. 위의 과정을 [그림 3.2]을 보고 설명을 하자면 먼저 출발점 S 로부터 도착점 D 까지의 경로 S → R1 → R2 → R3 → R4 → D(실선 경로)와 S → R1 → R5 → R6 → R3 → R7 → R8 → D(점선 경로)를 찾는다. 이후, 패킷을 보낼 시마다 두 경로 중 하나 만을 임의적으로 선택해서 전송하면 된다. 이때, 주의 점은 앞에서 말했던 네트워크의 일관성이 유지가 되도록 두 경로가 섞이면 안 된다는 것이다. 예를 들면 S → R1 → R2 → R3 → R7 → R8 → D와 같은 경로를 이용하면 안 된다. 이를 위해서 패킷 전송 전 라벨을 패킷에 붙여 라우터에서 경로를 알 수 있도록 하면 되겠다[7].



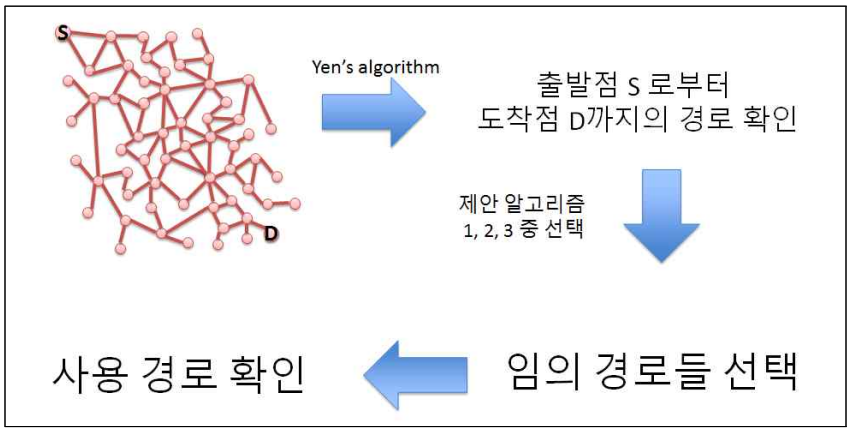
[그림 3.2] 임의 경로 변화(Random Route Mutation, RRM)[7]

제 4 장 S-RAY

본 논문은 기존 RRM[3] 보다 효율성이 우수한 Yen 알고리즘을 응용한 보안 네트워크 라우팅(Secure Routing Algorithm based on Yen's algorithm for a Network)을 제안한다. S-RAY는 제안 알고리즘 Ⅰ, Ⅱ, Ⅲ 으로 구성 되어 있다. 특징으로는 사용자 판단에 따라 그룹별로 원하는 만큼의 무작위 경로를 지정 할 수가 있으며 또한 허브 노드를 찾아 어느 부분에 보안 장비를 더욱 집중해야 하는지를 사용자에게 알려 줄 수 있는 점이다. 본 제안 알고리즘은 GNU에서 제공하는 무료 소프트웨어[24]를 바탕으로 수정 작성 하였으며 Python 2.7 버전을 사용하였다.

4.1 디자인

S-RAY는 [그림 4.1]에서와 같이 네트워크 그래프에서 출발점과 도착점이 주어지면 Yen 알고리즘을 활용하여 출발점 S 로부터 도착점 D 까지의 모든 경로를 구한다. 이후 제안 알고리즘 Ⅰ, Ⅱ, Ⅲ 중에서 사용자가 선택을 하여 그룹별 무작위로 사용자가 원하는 경로를 선택 후 사용하면 된다.

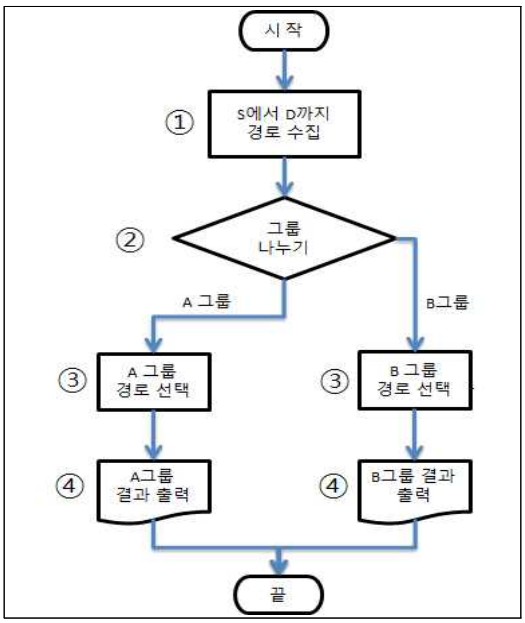


[그림 4.1] S-RAY 디자인

4.2 제안 알고리즘 Ⅰ

제안 알고리즘 Ⅰ은 [그림 4.2]와 같다. ①에서 출발점 S 로부터 도착점 D 까지의 모든 경로를 구한 후, ②에서 사용자가 몇 개의 그룹으로 나눌 지를 정한다. 본 논문에서는 편의

상 두 개의 그룹으로 하였으며 이 보다 많은 그룹으로 편성 가능하다. 이 후 ③에서는 A 그룹과 B 그룹의 경로가 겹치지 않게 사용자가 필요한 만큼의 경로를 무작위로 선정하여 ④에서 경로를 확인 하면 된다.



[그림 4.2] 제안 알고리즘 I 순서도

[그림 4.3]은 A 그룹의 경로를 구하는 프로그램이며 [그림 4.4]는 A그룹과 겹치지 않는 B그룹의 경로를 구하는 프로그램이다.

```
while True:
        a_list = []
        check = 0
        for i in range (0,10):
            number = random.randrange(0, len(result_sum))
            a_list.append(number)

        for k in range(len(a_list)):
            for p in range(len(a_list)):
                if a_list[k] == a_list[p]:
                    check += 1
        if check == len(a_list):
            break
```

[그림 4.3] A 그룹 경로 선택

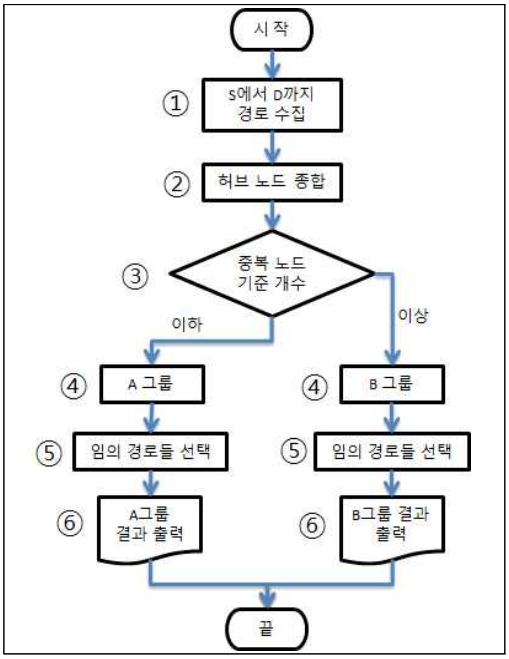
```
while True:
        b_list = []
        check = 0
        for i in range (0,10):
              number = random.randrange(0, len(result_sum))
              b_list.append(number)

        for k in range(len(a_list)):
              for p in range(len(b_list)):
                    if a_list[k] == b_list[p]:
                          check += 1

        for k in range(len(b_list)):
                      for p in range(len(b_list)):
                            if a_list[k] == a_list[p]:
                                  check += 1

                    if check == len(b_list):
                          break
```

[그림 4.4] B 그룹 경로 선택

4.3 제안 알고리즘 Ⅱ

제안 알고리즘 Ⅱ는 [그림 4.5]와 같다. ①까지는 제안 알고리즘 Ⅰ과 같다. ②에서는 ①에서 구한 모든 경로가 지나가는 허브 노드를 종합한다. 차후 이를 이용해서 보안장비를 증설 할 시 허브 노드를 우선으로 설치하면 전체적으로 네트워크의 보안 수준을 향상 시킬 수가 있을 것이다. ③에서는 ②에서 구한 허브 노드를 참고로 하여 다익스트라 알고리즘으로 구한 최적 경로와 중복 허용 노드 수를 지정하여 ④와 같이 이하인 경로들은 A 그룹에 이상인 경로들은 B 그룹으로 분류한다. 이때, 그룹에 따른 다음과 같은 특성이 있다. A 그룹은 다익스트라 알고리즘으로 구한 최적 경로와 중복이 적으므로 공격자가 노드를 예상하고 공격하기 어려우며 경로가 B 그룹에 비해 짧아 효율성이 좋다는 특성이 있다. 따라서 군과 같이 계급에 따라 보안성의 중요도가 차이가 나는 집단에서는 A 그룹은 상급자에게 B 그룹은 하급자에게 할당하면 되겠다. 이후, ⑤에서 사용자가 필요한 만큼의 경로를 무작위로 선

정하여 ⑥에서 경로를 확인하면 된다.



[그림 4.5] 제안 알고리즘 Ⅱ 순서도

[그림 4.6]은 허브 노드를 구하는 프로그램이며 [그림 4.7]은 A 그룹과 B 그룹으로 경로를 나누는 프로그램이다.

```
h_nodes = []
    for w in range(len(total_nodes)):
        sum_h = 0
        for p in range(len(path_v)):
            sum_c = 0
            for z in range(len(path_v[p])):
                if total_nodes[w] == path_v[p][z]:
                    sum_c = sum_c + 1
                    if sum_c == 1:
                        sum_h = sum_h + 1

        if sum_h == len(result_sum):
            h_nodes.append(total_nodes[w])
```

[그림 4.6] 허브 노드

```
for t_num in range(1, total_num):
        compare_num = 0
        for num_1 in range(total_1):
            total_c = len(path_v[t_num])
            for num_2 in range(total_c):
                if path_v[0][num_1] == path_v[t_num][num_2]:
                    compare_num += 1
        if compare_num <= di_number:
            first_c.append(path_v[t_num])
        else:
            second_c.append(path_v[t_num])
```

[그림 4.7] A · B 그룹으로 경로 나누기

[그림 4.8]과 [그림 4.9]는 A, B 그룹별로 사용 할 경로를 선택하는 프로그램이다.

```
while True:
        a_list = []
        check = 0
        for i in range(0,20):
            number = random.randrange(0, len(first_c)-1)
            a_list.append(number)

        for k in range(len(a_list)):
            for p in range(len(a_list)):
                if a_list[k]== a_list[p]:
                    check += 1
        if check == len(a_list):
            break
    print a_list, " \n"

    for w in a_list:
        print first_c[w]
```

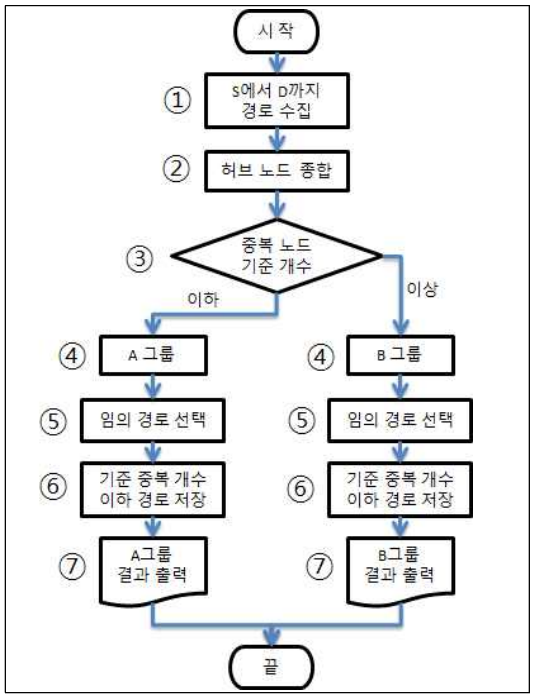
[그림 4.8] A 그룹 경로 선택

```
while True:
        b_list = []
        check = 0
        for i in range(0,20):
            number = random.randrange(0, len(second_c)-1)
            b_list.append(number)

        for k in range(len(b_list)):
            for p in range(len(b_list)):
                if b_list[k]== b_list[p]:
                    check += 1
        if check == len(b_list):
            break
    print b_list, " \n"

    for w in b_list:
        print second_c[w]
```

[그림 4.9] B 그룹 경로 선택

4.4 제안 알고리즘 Ⅲ

제안 알고리즘 Ⅲ은 [그림 4.10]과 같다. ④까지는 제안 알고리즘 Ⅱ와 같으며, ⑤ 부터 차이가 있다. 제안 알고리즘 Ⅱ에서는 사용자가 지정하는 수만큼의 경로를 바로 각 A, B 그룹에서 무작위로 선정을 했다면, 제안 알고리즘 Ⅲ에서는 ⑤에서 기준이 될 경로를 각 그룹에서 무작위로 선정을 하고 난 이후에 사용자가 지정한 수보다 중복 된 노드 수가 이하인 경로 만을 ⑥에서 선택을 하여 저장을 한다. 그리고 ⑦에서 그룹별로 출력 된 경로들을 확인한다. 단, 제안 알고리즘 Ⅲ의 주요 특징으로는 제안 알고리즘 Ⅰ, Ⅱ와는 달리 ⑦에서 출력이 되는 경로의 수를 사용자가 정할 수 없다는 단점이 있다. 이유는 각 그룹에서 임의로 선정 된 경로가 노드를 적게 거치는 경로가 선택이 된다면 사용자가 지정한 중복 가능 한 노드 수 보다 이하로 겹치는 경로가 많이 선택이 될 것이고, 반대로 노드를 많이 거치는 경로가 선택이 된다면 사용자가 지정한 중복 가능 한 노드 수 보다 이하로 겹치는 경로를 찾기가 힘들어 지기 때문이다. 즉, 사용자가 중복 가능한 노드가 같을지라도 프로그램을 실행 시마다 출력이 되는 경로 수는 달라 질 수가 있다. 하지만 이 때문에 장점도 있는데 중복 되는 노드 수가 최소화 되는 경로들을 선정 할 수 있다.



[그림 4.10] 제안 알고리즘 Ⅲ 순서도

[그림 4.11]은 각 그룹에서 무작위로 기준이 되는 경로를 선택하는 프로그램이다.

```
# Random First_Select
random_first_number = random.randrange(0, first_num)

# Random Second_Select
random_second_number = random.randrange(0,second_num)
```

[그림 4.11] 그룹별 무작위 기준 경로 선택

[그림 4.12]은 그룹별로 무작위로 선정된 경로를 기준으로 중복 허용 노드 수 이하의 노드가 중복되는 경로를 구하는 프로그램이다.

```
first_list_a = []
for d in range(first_num):
        first_uni_a = first_c[d].split()
        first_list_a.append(first_uni_a)
```

```
for k in range(len(first_list_a)):
        list_counter = 0
        list_compare = []
        for j in range(len(first_list_a[k])):
                for h in range(len(first_list_a[random_first_number])):
                        if first_list_a[k][j] == first_list_a[random_first_number][h]:
                                list_compare.append(first_list_a[k][j])
                                add_num = 0
                                for m in range(len(list_compare)):
                                        if list_compare[m] == first_list_a[k][j]:
                                                add_num += 1
                                if add_num < 2:
                                        list_counter += 1
        if list_counter <= r_first:
                first_select_a.append(first_list_a[k])
for n in range(len(first_select_a)):
        print first_select_a[n]
```

[그림 4.12] 그룹 경로 선택

제 5 장 시뮬레이션

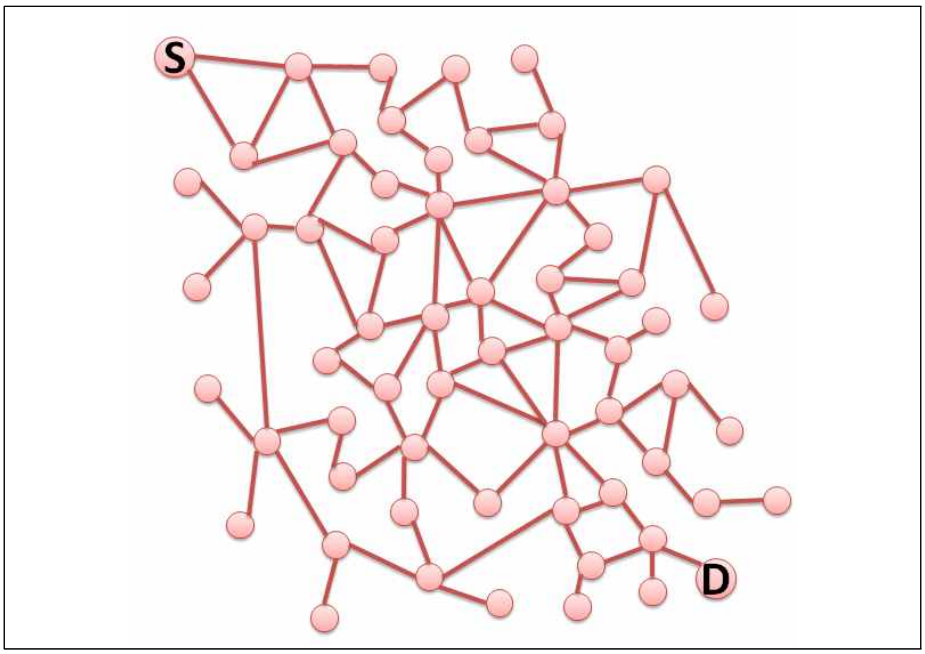
본 장에서는 S-RAY의 실행 시나리오와 네트워크 그래프, 그리고 제안 알고리즘 구현에 대해서 말하겠다.

5.1 시나리오

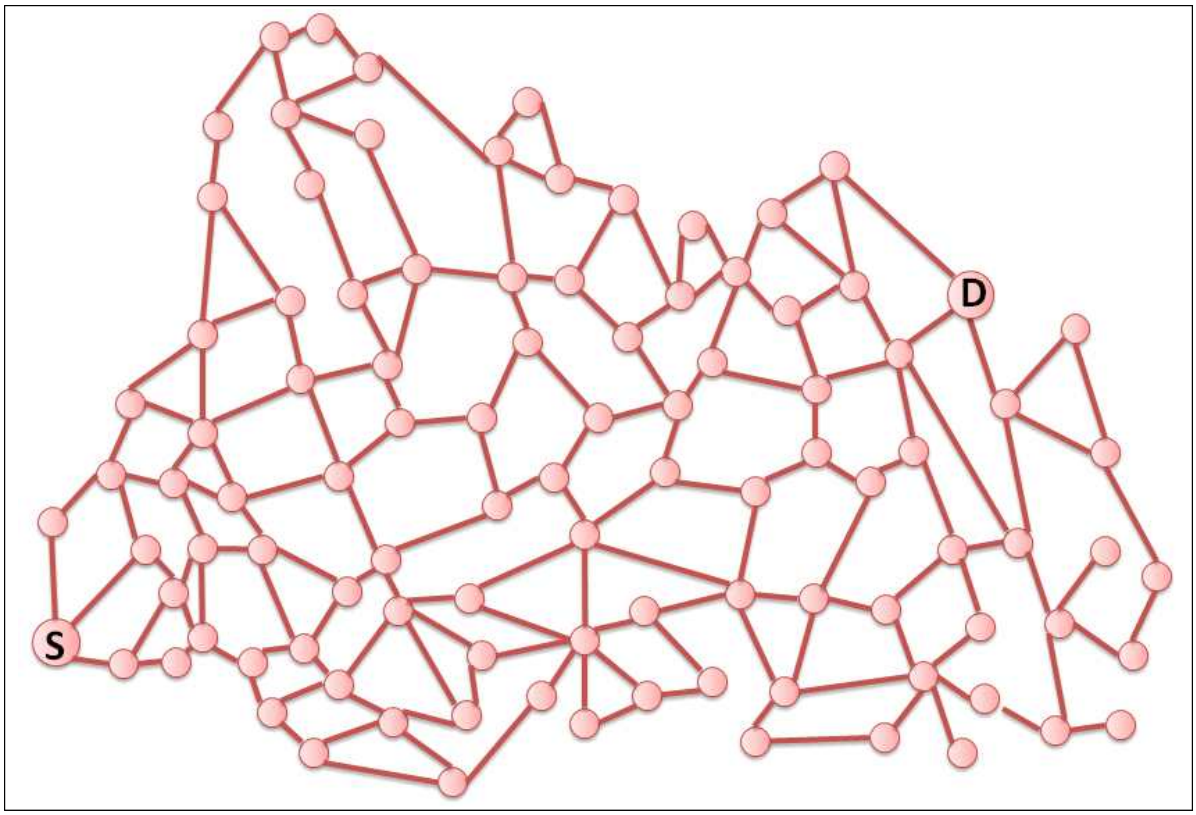
제안 알고리즘과 비교를 위해 먼저 일반라우팅을 실시하여 소요시간을 구한다. 다음으로 Yen 알고리즘을 이용하여 경로 수별 소요시간을 측정한다. 이후, 제안 알고리즘 Ⅰ, Ⅱ, Ⅲ을 실행하여 소요시간을 구한다.

5.2 네트워크 모형

제안 알고리즘을 실행 할 네트워크 모형은 가상의 네트워크에서 널리 사용되어지는[그림 5.1](60 노드 네트워크)과 [그림 5.2](100 노드 네트워크)이다. 출발점은 S 이며, 도착점은 D 이다.



[그림 5.1] 60 노드 네트워크



[그림 5.2] 100 노드 네트워크

5.3 제안 알고리즘 구현

본 논문 제안 알고리즘에서 그룹은 두 개로 하였으며 이는 사용자에 목적에 따라서 변경이 가능하다. 또한, 제안 알고리즘 Ⅱ, Ⅲ에서 그룹을 나누는 중복 가능 노드 개수 역시 사용자의 목적에 따라서 지정 할 수 있다. 제안 알고리즘 구현은 인텔 i7과 우분투 14 버전에서 실행 하였다.

제 6 장 평 가

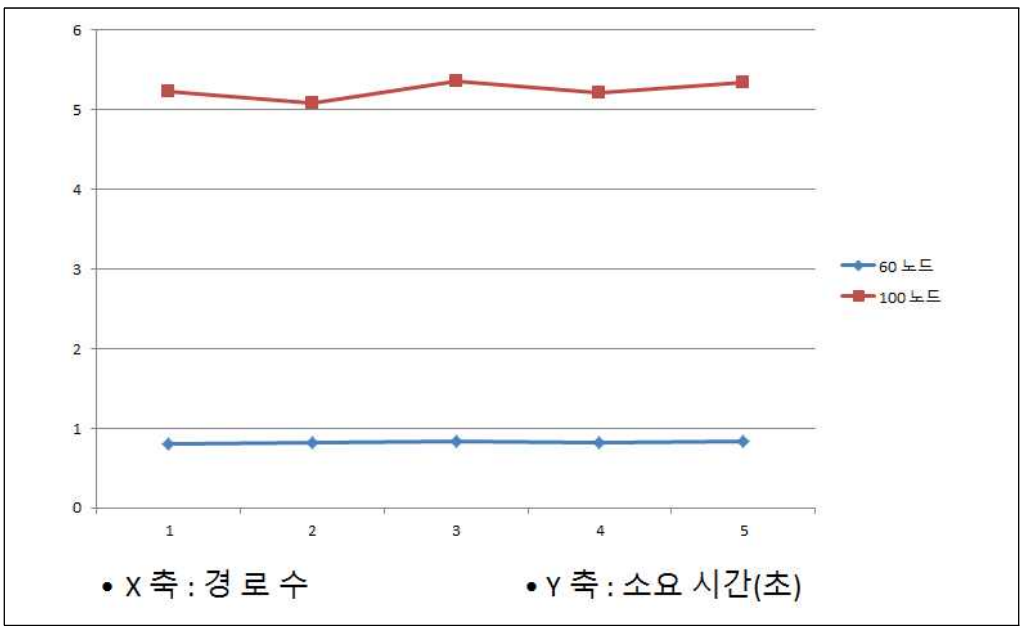
본 장에서는 일반 라우팅과 Yen 알고리즘을 이용한 경로 수 별 소요시간, 제안 알고리즘 Ⅰ, Ⅱ, Ⅲ의 결과를 분석 하겠다. 그리고 제안 알고리즘 Ⅰ, Ⅱ, Ⅲ 을 비교 분석 하겠다.

6.1 일반 라우팅

일반적인 인트라넷 라우팅은 최적 경로를 찾아 데이터를 전송한다. 하지만, 최적 경로가 단선이 되거나 데이터의 양이 대역폭을 초과 한다면 다른 경로를 찾아야 할 것이다. 따라서, 최적 경로 하나를 찾는데 걸리는 시간부터 5개의 경로를 찾는 시간을 측정하였다. [표 6.1]은 경로를 구하는 소요 시간이며, [그림 6.1]은 그래프이다. 여기서 일반 라우팅에서 경로를 찾는 시간은 60 노드 네트워크는 약 0.82 초, 100 노드 네트워크는 5.24 초가 걸리는 걸 알 수 있다.

[표 6.1] 경로를 찾는 소요 시간 단위 : 초

경 로(수)	1	2	3	4	5
60 노드 네트워크	0.80	0.81	0.83	0.82	0.83
100 노드 네트워크	5.23	5.08	5.36	5.21	5.34



[그림 6.1] 60 노드 네트워크 및 100 노드 네트워크 경로 산출 소요 시간 확인

6.2 Yen 알고리즘을 통한 경로 수에 따른 소요 시간

Yen 알고리즘을 활용하여 60 노드 네트워크 및 100 노드 네트워크의 경로 수별 소요시간을 측정했다. 이때, 60 노드 네트워크는 228개의 경로, 100 노드 네트워크는 192개의 경로가 구해 졌다. [표 6.2]은 60 노드 경로 수에 따른 소요 시간이다.

여기서 알 수 있는 사실은

일반 라우팅 시간 : t (0.82초)
경로 수 : R
경로 수 별 소요 시간 : T

로 하였을 시 다음과 같은 식이 성립 한다는 것을 알 수 있다.

$$T \fallingdotseq t + 0.0074 \times R$$

[표 6.2] 60 노드 네트워크 경로(수)에 따른 소요 시간

경로(수)	10	50	100	150	200	228
시간(초)	0.87	1.16	1.5	1.85	2.3	2.68

[표 6.3]는 100 노드 네트워크 경로 수에 따른 소요 시간이며,

일반 라우팅 시간 : t (5.24초)
경로 수 : R
경로 수 별 소요 시간 : T

로 하였을 시 다음과 같은 식이 성립 한다는 것을 알 수 있다.

$$T \fallingdotseq t + 0.015 \times R$$

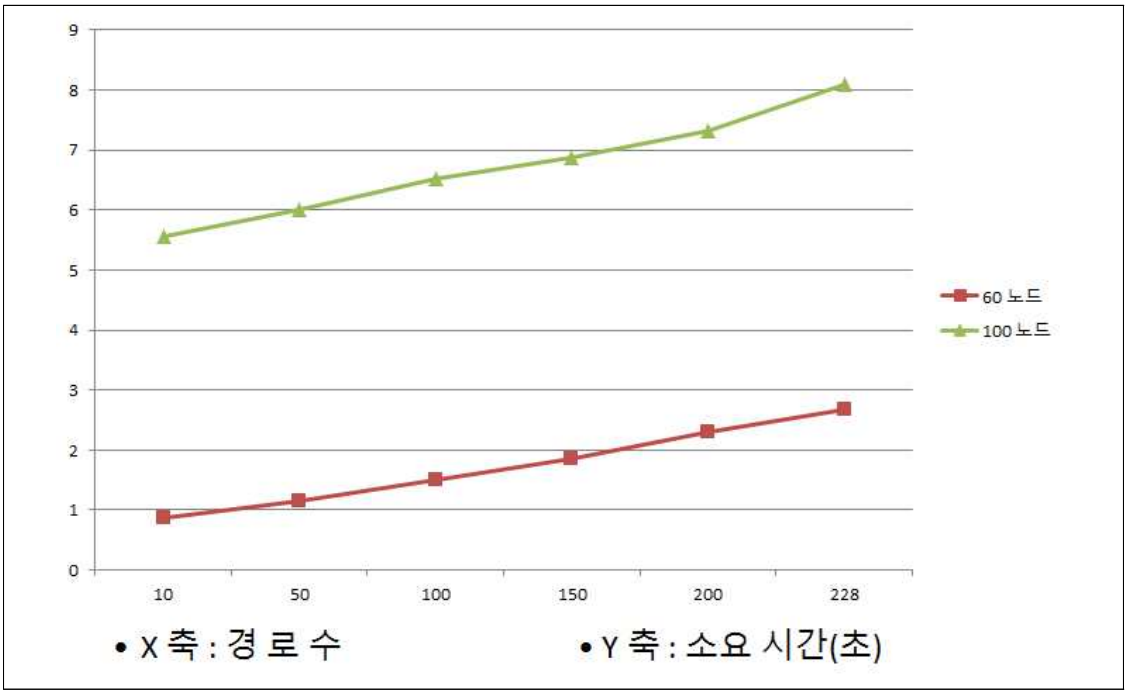
[표 6.3] 100 노드 네트워크 경로(수)에 따른 소요 시간

경로	10	40	80	120	160	192
시간(초)	5.56	6.01	6.51	6.87	7.32	8.09

60노드 네트워크 및 100 노드 네트워크의 변수 값 0.0074 및 0.015는 다음과 같이 구했다.

$$\frac{(T-t)}{R}$$

이 때 60 노드 네트워크와 100 노드 네트워크의 변수 값 비교를 위해 60 노드 네트워크는 200 경로 수 기준, 100 노드 네트워크는 192 경로 수 기준으로 계산 하였다. [그림 6.2]는 60 노드 네트워크와 100 노드 네트워크의 경로 수에 따른 소요 시간 비교 그래프이다.



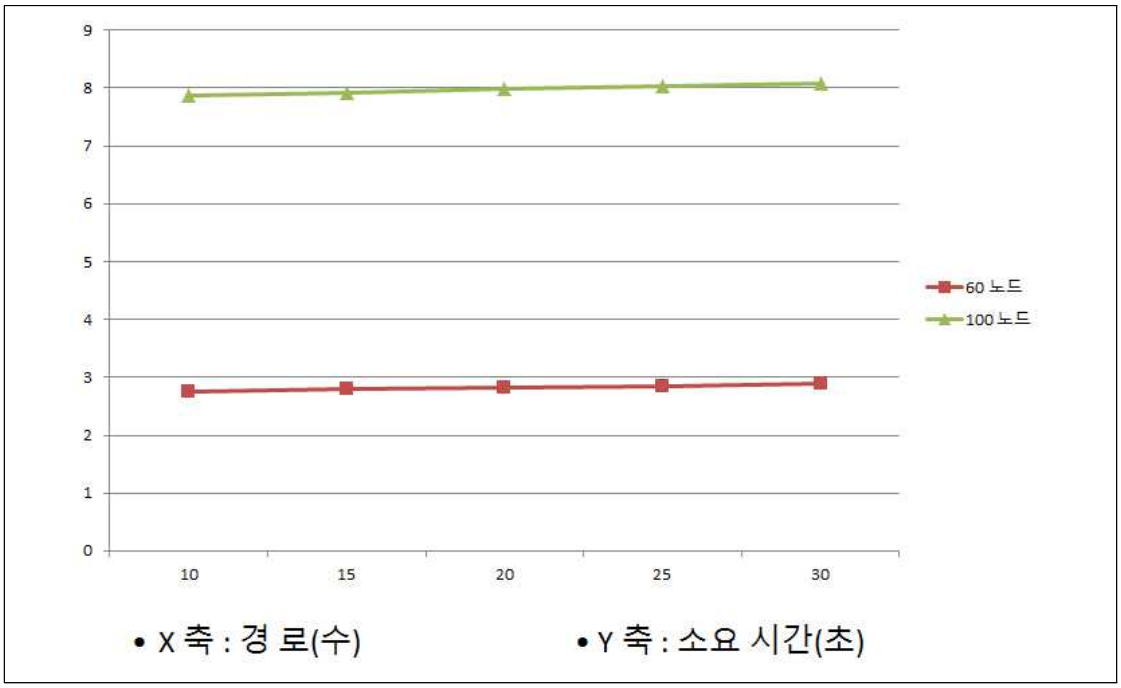
[그림 6.2] 60 노드 네트워크 및 100 노드 네트워크 경로(수)에 따른 소요시간

6.3 제안 알고리즘 Ⅰ 결과

제안 알고리즘 Ⅰ에서 60 노드 네트워크 및 100 노드 네트워크의 사용 경로 수는 Yen 알고리즘을 이용해서 구할 수 있는 최대 경로 수인 228 및 192로 하였으며 사용자의 의도에 따라 이 숫자 이하로 정할 수도 있다. [표 6.4]는 제안 알고리즘 Ⅰ의 소요 시간을 나타낸다. A 그룹 및 B 그룹에서 필요 경로 수는 [표 6.4]과 같이 입력하고 테스트를 하였으며, 이는 사용자의 필요에 따라 변경이 가능 하겠다. 여기서 알 수 있는 사실은 제안 알고리즘 Ⅰ의 소요 시간은 Yen 알고리즘을 이용하여 구할 수 있는 최대 경로를 구하는 시간(60 노드 네트워크 : 2.68초, 100 노드 네트워크 : 8.09초)과 거의 비슷하다는 것을 알 수가 있다. 이는 제안 알고리즘 Ⅰ의 소요 시간은 그룹 별 필요한 경로 수를 어떻게 정하느냐 보다 Yen 알고리즘을 이용하여 최초 몇 개의 경로를 얻는 가 임을 알 수 있다. 즉, 제안 알고리즘 Ⅰ의 소요 시간은 60 노드 네트워크는 [표 6.2]와 같이 100 노드 네트워크는 [표 6.3]과 비슷한 시간을 예상 할 수 있다. [그림 6.3]은 제안 알고리즘 Ⅰ의 결과를 그래프로 나타낸 것이다.

[표 6.4] 제안 알고리즘 Ⅰ 경로(수) 별 소요 시간

A 그룹 필요 경로 수	10	15	20	25	30
B 그룹 필요 경로 수	10	15	20	25	30
60 노드 네트워크(초)	2.76	2.79	2.83	2.84	2.89
100 노드 네트워크(초)	7.87	7.91	7.99	8.04	8.09



[그림 6.3] 제안 알고리즘 Ⅰ경로(수) 별 소요 시간

6.4 제안 알고리즘 Ⅱ 결과

제안 알고리즘 Ⅱ는 Yen 알고리즘을 활용하여 찾을 수 있는 출발점부터 도착점까지의 경로를 구한 이후에 허브 노드 네트워크를 찾는다. 여기서 허브 노드는 모든 경로가 지나가는 노드로 정의 한다. [표 6.5]은 60 노드 네트워크 및 100 노드 네트워크의 전체 경로 및 허브노드(수)이다.

[표 6.5] 60 노드 네트워크 및 100 노드 네트워크 전체 경로 및 허브 노드(수)

구 분	전체경로	허브노드
60 노드	228	3
100 노드	192	1

다음으로 60 노드 네트워크 및 100 노드 네트워크의 중복 허용 노드(수)에 따른 그룹별 경로(수)를 구한다. [표 6.6]은 60 노드 네트워크 [표 6.7]는 100 노드 네트워크의 중복 허용 노드(수)에 따른 그룹별 경로(수)이다.

[표 6.6] 60 노드 네트워크 중복 허용 노드(수)에 따른 그룹 별 경로(수)

중복 허용 노드(수)	3	4	5	6	7	8	9
A 그룹 경로(수)	2	8	22	51	115	193	228
B 그룹 경로(수)	226	220	206	177	113	36	0

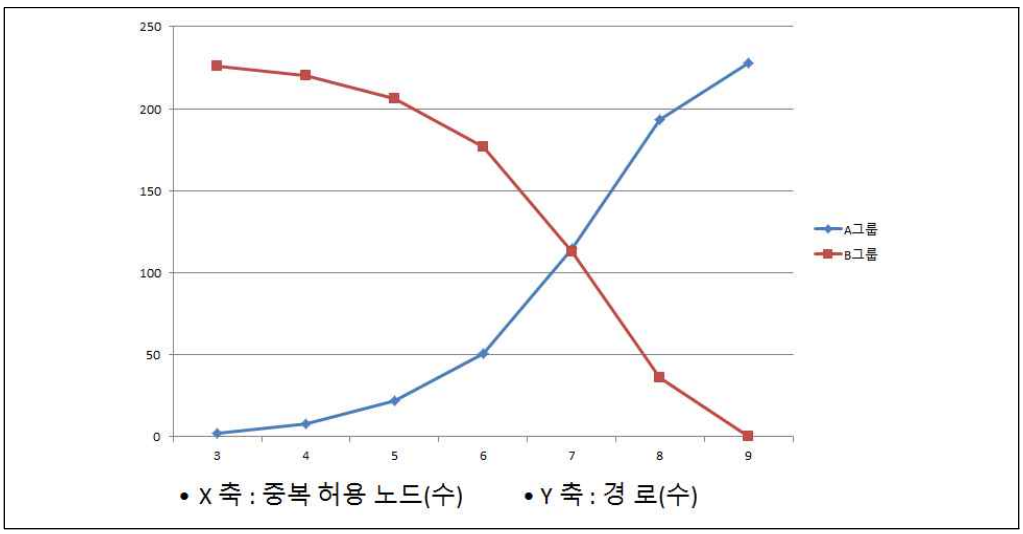
[표 6.6]에서 A 와 B 그룹의 경로 수가 30개 이상 인 중복 허용 노드 6, 7, 8 중에서 선택을 하면 되겠다. 본 논문에서는 중복 허용 노드 6을 선택하였다.

[표 6.7] 100 노드 네트워크 중복 허용 노드(수)에 따른 그룹 별 경로(수)

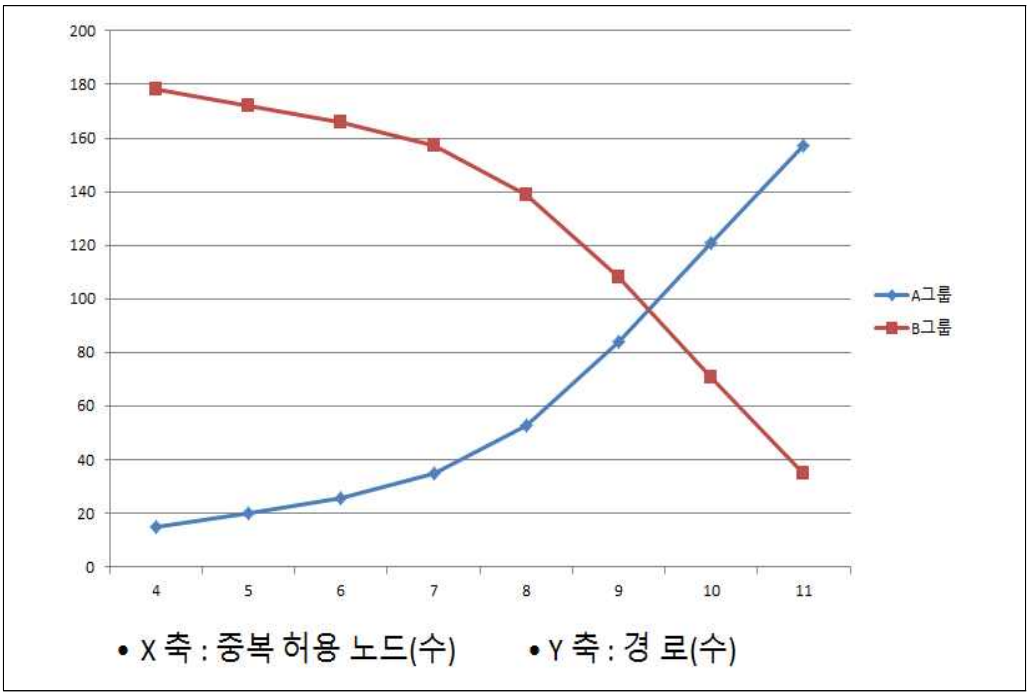
중복 허용 노드(수)	4	5	6	7	8	9	10	11	12
A 그룹 경로(수)	15	20	26	35	53	84	121	157	180
B 그룹 경로(수)	178	172	166	157	139	108	71	35	12

[표 6.7]에서 A 와 B 그룹의 경로 수가 30개 이상 인 중복 허용 노드 7, 8, 9, 10, 11 중에서 선택을 하면 되겠다. 본 논문에서는 중복 허용 노드 8을 선택하였다.

[그림 6.4]은 60노드 네트워크 중복 허용 노드(수)에 따른 그룹별 경로(수) 그래프이며 [그림 6.5]는 100 노드 네트워크 중복 허용 노드(수)에 따른 그룹별 경로(수)의 그래프이다.



[그림 6.4] 60 노드 네트워크 중복 허용 노드(수)에 따른 그룹별 경로(수)



[그림 6.5] 100 노드 네트워크 중복 허용 노드(수)에 따른 그룹별 경로(수)

[표 6.8]는 60 노드 네트워크의 중복 허용 노드를 6개로 하여 그룹 별 선택 경로를 구하는데 걸린 시간을 나타낸다.

[표 6.8] 60 노드 네트워크 그룹 선택 경로(수)에 따른 소요 시간(중복 허용 노드 6개 기준)

A 그룹에서 선택 한 경로(수)	10	15	20	25	30
B 그룹에서 선택 한 경로(수)	10	15	20	25	30
시 간(초)	4.42	4.29	4.35	4.6	7.59

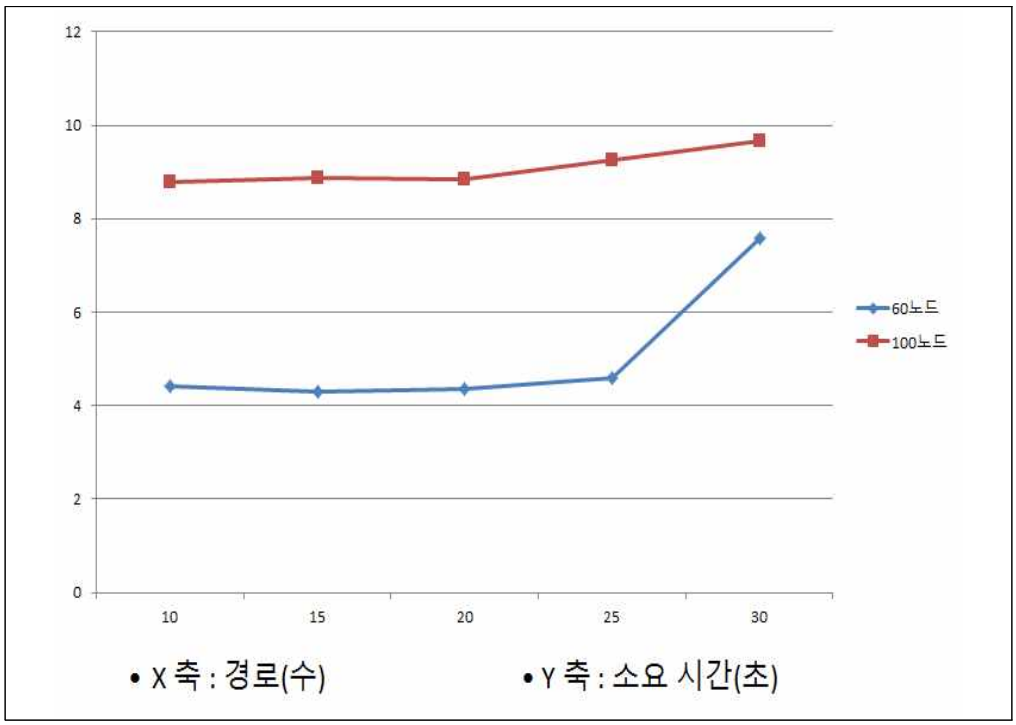
그룹 선택 경로 수가 25 까지는 소요 시간이 4초 대로 비슷한 것을 알 수가 있다. 하지만, 선택 경로 수가 30에서는 7초 대로 급격히 증가한 것을 알 수가 있다. 이는 중복 허용 노드 6개 기준으로 그룹 별로 경로를 나누었을 때, A 그룹의 경로 수가 51개이기 때문이라 분석된다. 즉, 51개의 50% 수준인 25개 까지는 무작위로 서로 중복되지 않는 경로를 선정하는 것이 쉬웠지만 60% 수준인 30개 경로를 선정 하는 것에서는 서로 중복이 되지 않는 경로를 찾는 것이 좀 더 어렵다는 것으로 분석된다.

[표 6.9]은 100 노드 네트워크의 중복 허용 노드를 8개로 하여 그룹 별 선택 경로를 구하는데 걸린 시간을 나타낸다.

[표 6.9] 100 노드 네트워크 그룹 선택 경로(수)에 따른 소요 시간(중복 허용 노드 8개 기준)

A 그룹에서 선택 한 경로(수)	10	15	20	25	30
B 그룹에서 선택 한 경로(수)	10	15	20	25	30
시 간(초)	8.78	8.87	8.83	9.25	9.65

[표 6.9]에서 보듯이 그룹에서 선택 한 경로 수가 25 이상부터 소요 시간이 증가하는 것을 알 수 있다. [그림 6.6]는 60 노드 네트워크와 100 노드 네트워크의 그룹 선택 경로 수에 따른 소요 시간 그래프이다.



[그림 6.6] 60 노드 네트워크와 100노드 네트워크의 그룹 선택 경로(수)에 따른 소요 시간

6.5 제안 알고리즘 Ⅲ 결과

제안 알고리즘 Ⅲ은 제안 알고리즘 Ⅱ에서 그룹 별 경로를 나누는 것까지는 같으며 이후 그룹별로 필요 한 경로를 선택하는 방식이 다르다. 제안 알고리즘 Ⅱ에서는 각 그룹별로 무작위로 사용자가 지정한 수 만큼의 경로를 선택 한다면 제안 알고리즘 Ⅲ은 각 그룹에서 무작위로 기준이 되는 경로를 선택하고 사용자가 지정한 중복 허용 가능 노드 개수 이하의 경로를 선택한다. 60 노드 네트워크와 100 노드 네트워크 결과를 구분해서 말하겠다.

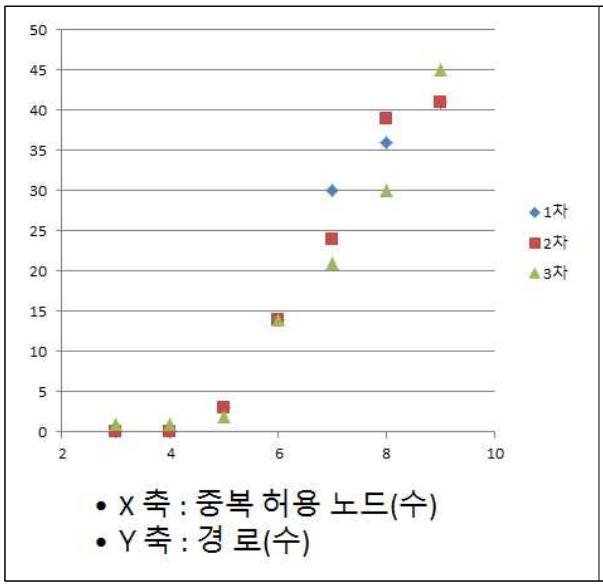
6.5.1. 60 노드 네트워크

[표 6.10]은 A 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수) 및 소요 시간을 나타낸다. 여기서 횟 수별로 경로 수가 다른 것을 볼 수가 있는데 이것은 기준이 되는 경로가 어떻게 선택이 되는지에 따라 달라지는 것을 알 수 있겠다. 그리고 최소 중복 허용 노드가 5 이상이 되야 경로를 얻을 수 있다는 것을 보여준다.

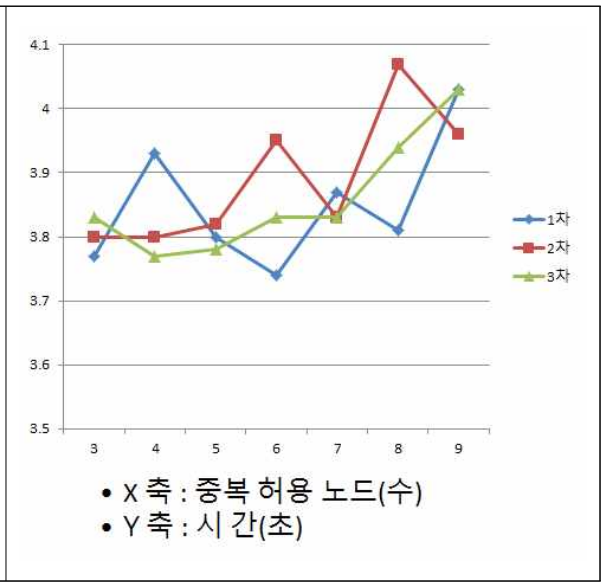
[표 6.10] A 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수) 및 시간

중복 허용 노드(수)		3	4	5	6	7	8	9
1차	경 로(수)	0	0	3	14	30	36	41
	시 간(초)	3.77	3.93	3.80	3.74	3.87	3.81	4.03
2차	경 로(수)	0	0	3	14	24	39	41
	시 간(초)	3.80	3.80	3.82	3.95	3.83	4.07	3.96
3차	경 로(수)	1	1	2	14	21	30	45
	시 간(초)	3.83	3.77	3.78	3.83	3.83	3.94	4.03

[그림 6.7]은 중복 허용 노드(수)에 따라 얻을 수 있는 횟 수별 경로(수) 그래프이며 [그림 6.8]는 중복 허용 노드(수)에 따른 횟 수별 시간이다.



[그림 6.7] A 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수)



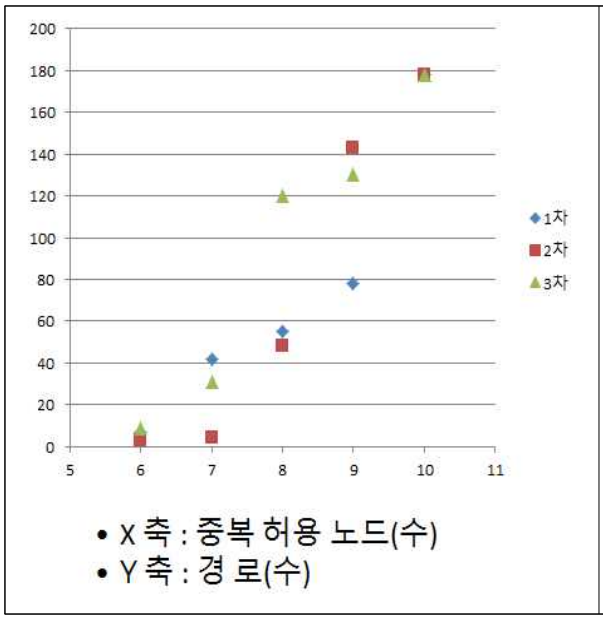
[그림 6.8] A 그룹 중복 허용 노드(수)에 따른 횟 수별 시간(초)

[표 6.11]는 B 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수) 및 소요 시간을 나타낸다. 마찬가지로 횟 수별로 선택 경로 수가 달라지는 것을 알 수가 있는데 이 역시 기준이 되는 경로가 어떻게 선택이 되냐에 따라서 달라지는 것을 알 수가 있다.

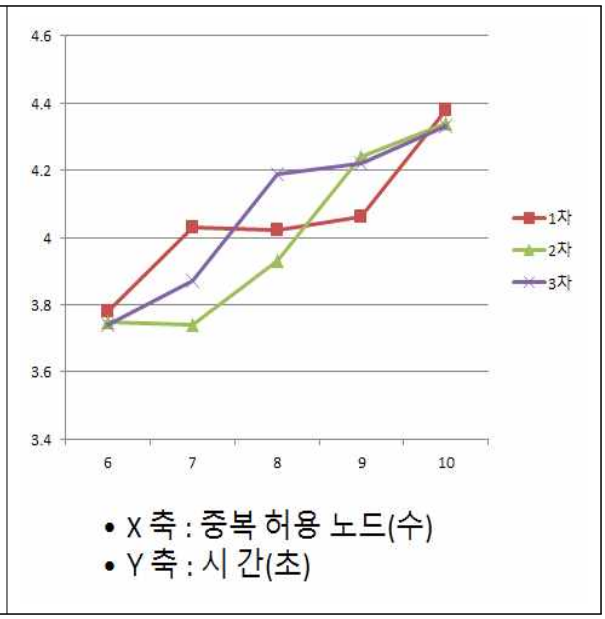
[표 6.11] B 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수) 및 시간

중복 허용 노드(수)		6	7	8	9	10
1차	경 로(수)	7	42	55	78	178
	시 간(초)	3.78	4.03	4.02	4.06	4.38
2차	경 로(수)	3	4	48	143	178
	시 간(초)	3.75	3.74	3.93	4.24	4.34
3차	경 로(수)	9	31	120	130	178
	시 간(초)	3.74	3.87	4.19	4.22	4.33

[그림 6.9]은 중복 허용 노드(수)에 따라 얻을 수 있는 횟 수별 경로(수) 그래프이며 [그림 6.10]는 중복 허용 노드(수)에 따른 횟 수별 시간이다.



[그림 6.9] B 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수)



[그림 6.10] B 그룹 중복 허용 노드(수)에 따른 횟 수별 시간(초)

6.5.2. 100 노드 네트워크

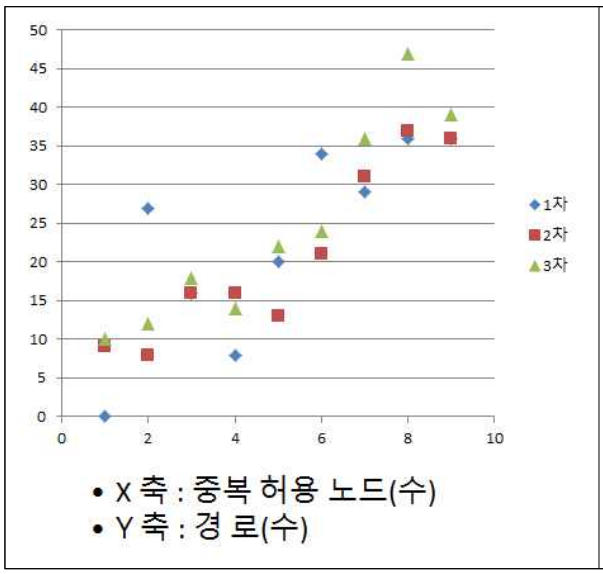
[표 6.12]은 A 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수) 및 소요 시간을 나타낸다. 60 노드 네트워크에서와 마찬가지로 횟 수별로 경로 수가 다른 것을 볼 수가 있다. 중

복 노드 수가 2 이상은 되야 안정적으로 사용 할 경로를 얻을 수 있겠다.

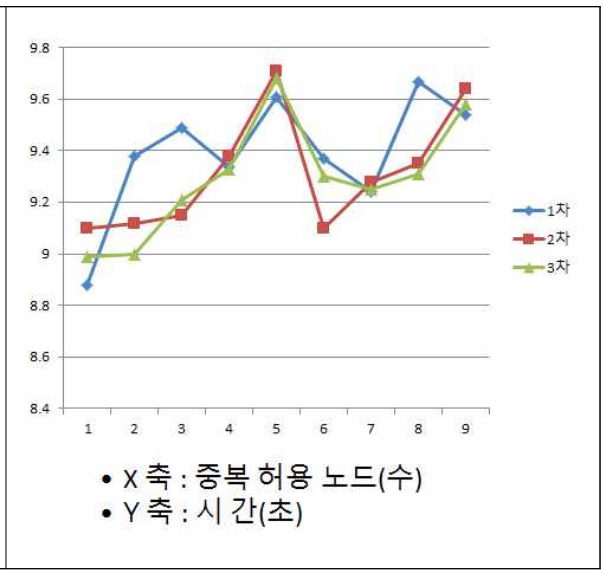
[표 6.12] A 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수) 및 시간

중복 허용 노드(수)		1	2	3	4	5	6	7	8	9
1차	경 로(수)	0	27	16	8	20	34	29	36	36
	시 간(초)	8.88	9.38	9.49	9.34	9.61	9.37	9.24	9.67	9.54
2차	경 로(수)	9.06	8	16	16	13	21	31	37	36
	시 간(초)	9.10	9.12	9.15	9.38	9.71	9.1	9.28	9.35	9.64
3차	경 로(수)	10	12	18	14	22	24	36	47	39
	시 간(초)	8.99	9.00	9.21	9.33	9.68	9.30	9.25	9.31	9.58

[그림 6.11]은 중복 허용 노드(수)에 따라 얻을 수 있는 횟 수별 경로(수) 그래프이며 [그림 6.12]는 중복 허용 노드(수)에 따른 횟 수별 시간이다.



[그림 6.11] A 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수)



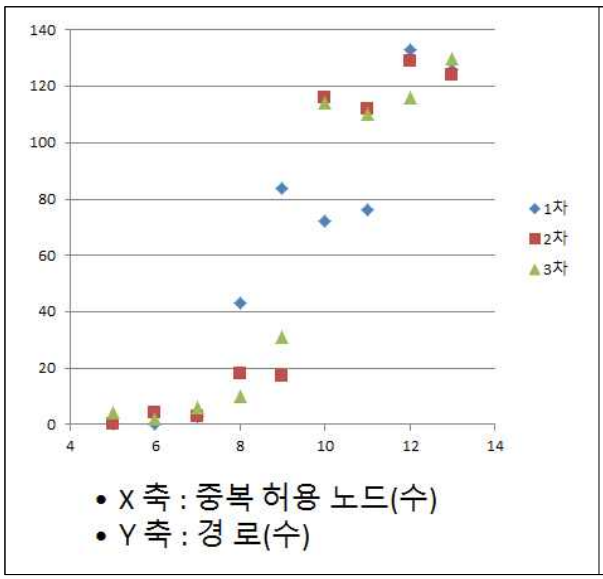
[그림 6.12] A 그룹 중복 허용 노드(수)에 따른 횟 수별 시간(초)

[표 6.13]는 B 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수) 및 소요 시간을 나타낸다. 마찬가지로 횟 수별로 선택 경로 수가 달라지는 것을 알 수가 있다.

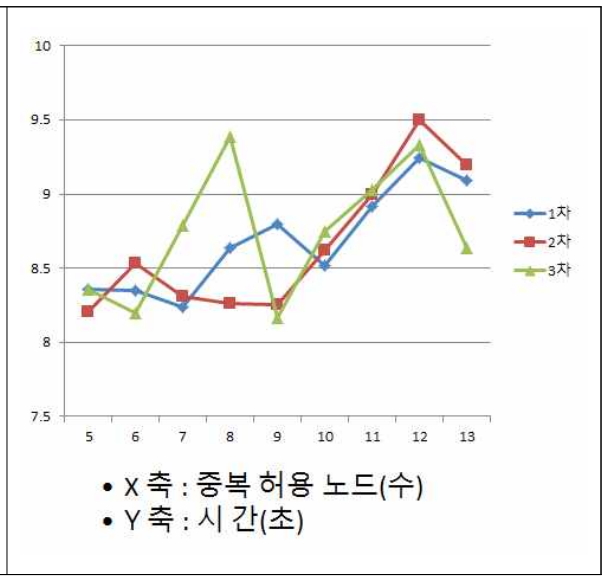
[표 6.13] B 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수) 및 시간

중복 허용 노드(수)		5	6	7	8	9	10	11	12	13
1차	경 로(수)	0	0	3	43	84	72	76	133	126
	시 간(초)	8.36	8.35	8.24	8.64	8.8	8.52	8.92	9.24	9.09
2차	경 로(수)	0	4	3	18	17	116	112	129	124
	시 간(초)	8.21	8.53	8.31	8.26	8.25	8.62	9.00	9.50	9.20
3차	경 로(수)	4	2	6	10	31	114	110	116	130
	시 간(초)	8.36	8.20	8.79	9.39	8.17	8.75	9.03	9.33	8.64

[그림 6.13]은 중복 허용 노드(수)에 따라 얻을 수 있는 횟 수별 경로(수) 그래프이며 [그림 6.14]는 중복 허용 노드(수)에 따른 횟 수별 시간이다.



[그림 6.13] B 그룹 중복 허용 노드(수)에 따른 횟 수별 경로(수)



[그림 6.14] B 그룹 중복 허용 노드(수)에 따른 횟 수별 시간(초)

6.6 제안 알고리즘 비교

[표 6.14]은 제안 알고리즘을 비교 한 것이다. 먼저 소요시간은 제안 알고리즘 Ⅰ이 60 노드 네트워크에서는 2초 대이고 100 노드 네트워크에서는 8초 초반을 보이고 제안 알고리

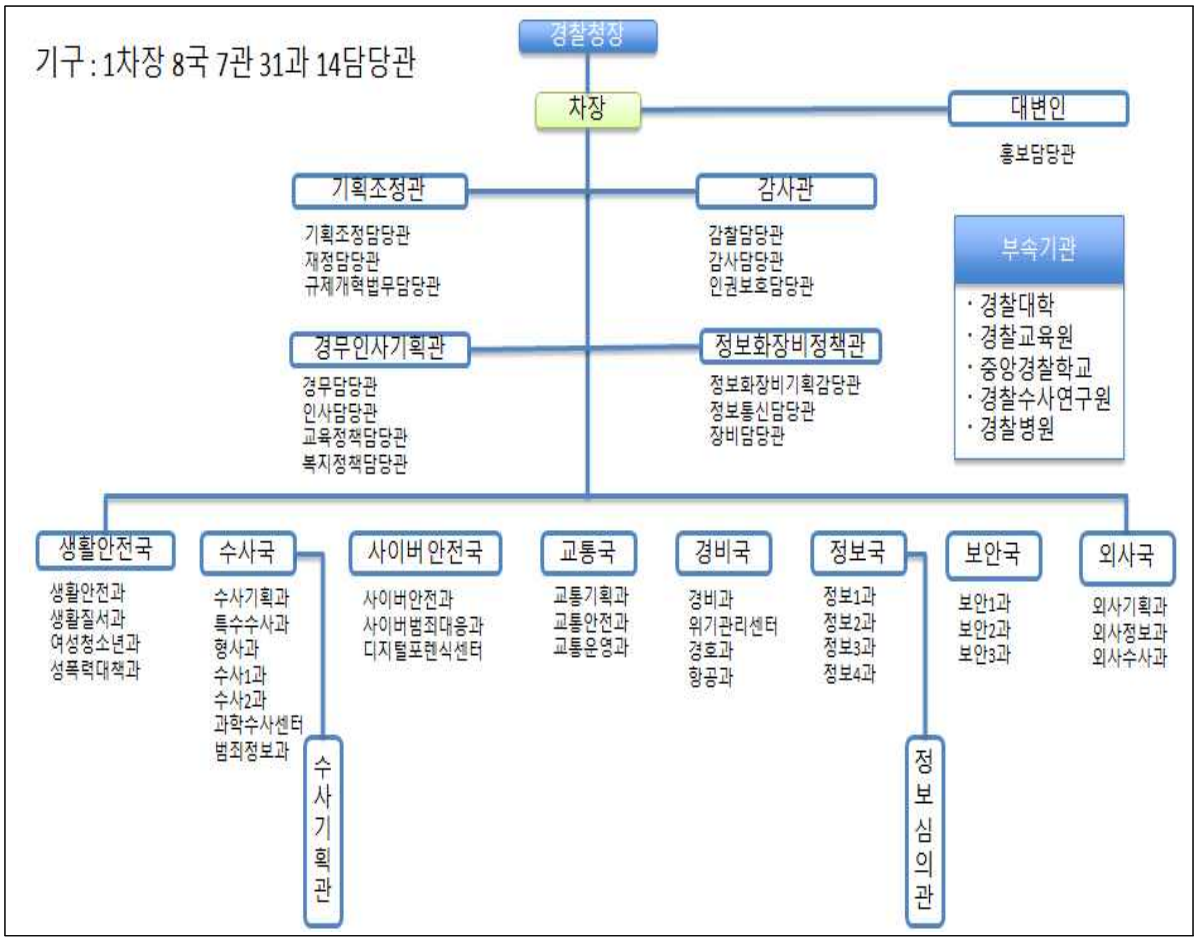
즘 Ⅱ, Ⅲ은 60 노드 네트워크에서는 4초 대와 100 노드 네트워크에서 8초 후반에서 9초대를 보인다. 따라서 소요 시간은 제안 알고리즘 Ⅰ이 우수하다. 제안 알고리즘 Ⅰ, Ⅱ, Ⅲ을 현재 네트워크에 적용하는데 있어 시설적으로 추가적인 비용은 필요 없다. 제안 알고리즘 Ⅱ, Ⅲ은 허브 노드를 찾아 어느 노드의 보안 시설을 더 강화 시켜야 하는지 알려 줄 수 있으나, 제안 알고리즘 Ⅰ은 그렇지 못해 보안성은 제안 알고리즘 Ⅱ, Ⅲ이 제안 알고리즘 Ⅰ 보다 우수하다. 경로 수 선택은 제안 알고리즘 Ⅰ, Ⅱ는 사용자의 의도에 따라 경로 선택 수를 정할 수 있다면 제안 알고리즘 Ⅲ은 무작위로 선택이 되어 사용자 편의성으로 본다면 제안 알고리즘 Ⅰ, Ⅱ 가 우수하다.

[표 6.14] 제안 알고리즘 비교

구 분	소요 시간	추가 비용	보안성	경로 수 선택	자원 사용
제안 알고리즘 Ⅰ	짧음	없음	보통	가능	낮음
제안 알고리즘 Ⅱ	많음	없음	우수	가능	보통
제안 알고리즘 Ⅲ	많음	없음	최우수	제한	많음

제 7 장 활용방안

본 장에서는 S-RAY의 활용방안을 보안서비스 품질(Quality of Security Service, QoSS)에 적용하여 설명 하겠다. 먼저, 보안 서비스 품질은 관리자와 사용자가 보안 서비스의 등급을 선택 사용 할 수 있게 하여 시스템의 전체적인 효율성을 향상시키는 것을 말한다[26]. S-RAY 는 제안 알고리즘 Ⅰ, Ⅱ, Ⅲ이 있으며 이는 보안 서비스 품질에 따라서 나뉘어 진다. S-RAY를 보안성이 중요 시 되는 정부, 군, 경찰 등을 예로 적용 방안을 설명 할 수 있겠으며, 본 논문에서는 자료가 공개 된 경찰 조직을 예로 설명 하겠다. [그림 7.1]은 경찰 조직도이다.



[그림 7.1] 경찰청 조직 구성도[25]

[표 7.1]은 경찰청 조직에서 제안 알고리즘 적용 예이다. 먼저, 제안 알고리즘 Ⅰ은 보안성은 보통이고, 자원 사용은 낮으므로 높은 보안 수준을 요구하지 않는 31과 및 14담당관 인원이 적합하겠다. 다음으로 제안 알고리즘 Ⅱ는 보안성은 우수하고, 자원 사용은 보통이

므로 높은 보안성이 요구되는 8국 및 7과의 장이 적합하다. 마지막으로 제안 알고리즘 Ⅲ은 가장 우수한 보안성을 제공하지만, 그 만큼 시스템 자원을 사용 한다. 따라서 무엇보다 보안성을 가장 우선 시 하는 경찰청장 및 차장에게 적합하겠다.

[표 7.1] 경찰청 조직에 대한 제안 알고리즘 적용 예

제안 알고리즘	Ⅰ	Ⅱ	Ⅲ
인 원	31과 및 14담당관 인원	8국 및 7과의 장	경찰청장, 차장

이와 같은 방법은 S-RAY는 사용자에 적합한 보안 서비스 품질을 제공함에 따라 시스템 자원을 효율적으로 사용 할 수가 있으며 또한, 사용자에게 S-RAY 사용에 대한 만족감을 줄 것이다.

제 8 장 결 론

현재 우리는 다양한 분야에서 인터넷을 사용하고 있으며, 이러한 추세는 앞으로 더욱 증가 할 것으로 예상한다. 하지만 아직까지 유선 네트워크의 대부분은 고정된 경로를 사용하고 있어 서비스 거부 공격(Denial of Service, DoS)이나 중간자 공격(Man In The Middle, MITM)에 취약한 것이 사실이다.

따라서, 본 논문에서는 Yen 알고리즘을 응용한 보안 네트워크 라우팅(Secure Routing Algorithm based on Yen's algorithm for a Network)을 제안 했다. 이는 고정된 경로가 아닌 다양한 경로를 무작위로 선정하여 데이터를 전송함으로써 공격자들로부터 네트워크의 안전성을 유지 할 수가 있으며 또한 기존의 임의 경로 변화(Random Route Mutation, RRM)가 사용자 품질 측면에서 문제가 있었다면 S-RAY는 이러한 문제점을 해결하였다. 또한, 허브 노드가 어떤 것인지를 발견하여, 이 노드의 보안성을 강화 시킴으로써 전체 네트워크의 보안성을 강화 시킬 수도 있으며 사용자에 따른 보안 서비스 품질을 제공 할 수 있어 전체적인 시스템을 효율적으로 사용 할 수 있겠다. 이 방법은 인터넷과 인트라넷의 프로토콜에서 모두 사용이 가능 하겠으며, 특히, 보안성이 중요시 되는 군에서 활용을 한다면 그 효과는 더욱 좋을 것이다. 또한, 현재 네트워크 및 차세대 네트워크라 불리는 소프트웨어 정의 네트워크(Software Defined Network, SDN)에서도 적용 가능하다.

본 논문에서 사용 된 제안 알고리즘은 Python 2.7 버전을 사용하였다. 차후 C 언어를 사용하여 효율성을 향상 시키겠다.

참 고 문 헌

[1] 한국인터넷진흥원(KISA), http://isis.kisa.or.kr/

[2] TaTao Shu, Marwan Krunz, and Sisi Liu. Secure data collection in wireless sensor networks using randomized dispersive routes. IEEE Transactions on Mobile Computing, 9(7):941 – 954, July 2010.

[3] Wenjing Lou, Wei Liu, and Yuguang Fang. Spread: enhancing data confidentiality in mobile ad hoc networks. In IEEE INFOCOM, pages 2404 – 2413, 2004.

[4] Stephen M. Specht, Ruby B. Lee, “Distributed Denial of Service:Taxonomies of Attacks, Tools and Countermeasures, Taxonomies of Attacks, Tools, and Countermeasures”, Proceedingsof the 17th International Conference on Parallel and Distributed Computing Systems, 2004 International Workshop on Security inParallel and Distributed Systems, pp. 543-550, September 2004

[5] Italo Dacosta, Mustaque Ahamad, and Patrick Traynor, “Trust No One Else: Detecting MITM Attacks AgainstSSL/TLS Without Third-Parties”

[6] Sanjay Ramaswamy, Huirong Fu, Manohar Sreekantaradhya, John Dixon and Kendall Nygard, “Prevention of Cooperative Black Hole Attackin Wireless Ad Hoc Networks”

[7] Qi Duan, Ehab Al-Shaer, Haadi Jafarian, “Efficient Random Route Mutation Considering Flowand Network Constraints”, Department of Software and Information Systems University of North Carolina at Charlotte Charlotte, NC, USA, 2013 IEEE Conference on Communications and Network Security (CNS)

[8] Yen, Jin Y. (1970). "An algorithm for finding shortest routes from all source nodes to a given destination in general networks". Quarterly of Applied Mathematics 27: 526 – 530.

[9] YHedrick, C., "Routing Information Protocol", STD 34, RFC 1058, Rutgers University, June 1988.

[10] Microsoft, IP 라우팅 프로토콜, https://msdn.microsoft.com/ko-kr/library/ cc758294 (v=ws.10).aspx

[11] RIP v1.03 – Aaron Balchunas, All original material copyright © 2012 by Aaron Balchunas, http://www.routeralley.com/guides/rip.pdf

[12] Microsoft, IP용 RIP https://msdn.microsoft.com/ko-kr/library/cc736472(v=ws.10).aspx

[13] Moy, J. (April 1998). "OSPF Version 2". The Internet Society. OSPFv2. Retrieved 2007-09-28.

[14] Microsoft, OSPF, https://msdn.microsoft.com/ko-kr/library/cc778874(v=ws.10).aspx

[15] E. W. Dijkstra: A note on two problems in connexion with graphs. In: Numerische Mathematik. 1 (1959), S. 269 – 271

[16] DIJKSTRA'S ALGORITHM By Laksman Veeravagu and Luis Barrera, http://www.cs.utexas.edu/~EWD/

[17] E.Yen, Jin Y. (1970). "An algorithm for finding shortest routes from all source nodes to a given destination in general networks". Quarterly of Applied Mathematics 27: 526 – 530.

[18] Yen's algorithm, https://en.wikipedia.org/wiki/Yen%27s_algorithm

[19] John Hershberger, Matthew Maxel, Subhash Suri, "Finding the k Shortest Simple Paths:A New Algorithm and its Implementation", Supported in partby National Science Foundation grants CCR-9901958 and IIS-0121562

[20] JR. Dingledine, N. Mathewson, and P. Syverson. Tor: Thesecond-generation onion router. In Proceedings of the 13th USENIX Security Symposium (USENIX Security '04), 2004.

[21] D. Goodin. Tor at heart of embassy passwords leak. TheRegister, Sept. 10, 2007

[22] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending anonymous communication against passive logging attacks. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, May 2003.

[23] Tor, https://www.torproject.org/about/overview.html.en

[24] https://github.com/Pent00/YenKSP/commit/95e03e1741fd190c69d833c06b6896a196583f3c?diff=split

[25] 사이버경찰청, 정보공개 공공데이터개방 경찰통계자료 기회조정, http://www.police.go.kr/portal/main/ contents.do?menuNo= 200190

[26] Cynthia Irvine, Timothy Levin, "Quality of Security Service", This work was supported under the MSHN Project of the DARPA/ITO Quorum Program.

부 록

가. 제안 알고리즘 I (__init__.py)

```
from graph import DiGraph
import algorithms
import time
import random

def main():
    start_time = time.time()

    # Load the graph
    G = DiGraph("net5")

    # Get the painting object and set its properties.
    paint = G.painter()
    paint.set_source_sink("A1", "E7")

    # Generate the graph using the painter we configured.
    G.export(False, paint)

    cost_v = []
    path_v = []
    path_v1 = []
    result_sum = []

    # Get 30 shortest paths from the graph.
    items = algorithms.ksp_yen(G, "A1", "E7", 250)
    for path in items:
        result = "Cost:%s₩t%s" % (path['cost'], "->".join(path['path']))
        result_sum.append(result)

    result_sum.sort()

    print len(result_sum)
```

```
for i in range(len(result_sum)):
    print result_sum[i]

print
check = 0
number = 0

while True:
    a_list = []
    check = 0
    for i in range (0,10):
        number = random.randrange(0, len(result_sum))
        a_list.append(number)

    for k in range(len(a_list)):
        for p in range(len(a_list)):
            if a_list[k] == a_list[p]:
                check += 1
    if check == len(a_list):
        break


while True:
    b_list = []
    check = 0
    for i in range (0,10):
        number = random.randrange(0, len(result_sum))
        b_list.append(number)

    for k in range(len(a_list)):
        for p in range(len(b_list)):
            if a_list[k] == b_list[p]:
                check += 1

    for k in range(len(b_list)):
        for p in range(len(b_list)):
            if a_list[k] == a_list[p]:
                check += 1

    if check == len(b_list):
        break
```

```
    print check

    print a_list
    print

     for w in range(len(a_list)):
         print result_sum[w]

    print
    print b_list
    print
    for t in range(len(b_list)):
         print result_sum[t]



    end_time = time.time()
    print "the time taken : ", end_time - start_time

    return 0

if __name__ == "__main__":
    main()
```

나. 제안 알고리즘 Ⅱ(__init__.py)

```
from graph import DiGraph
import algorithms
import time
import random

def main():
    start_time = time.time()

    # Load the graph
    G = DiGraph("net5")

    # Get the painting object and set its properties.
    paint = G.painter()
    paint.set_source_sink("A1", "E7")

    total_nodes = ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9',
                   'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9',
                   'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9',
                   'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9',
                   'E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8', 'E9',
                   'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9',
                   'G1', 'G2', 'G3', 'G4', 'G5', 'G6']

    # Generate the graph using the painter we configured.
    G.export(False, paint)

    path_v = []

    result_sum = []

    # Get shortest paths from the graph.
    items = algorithms.ksp_yen(G, "A1", "E7", 250)
    for path in items:
        result = "Cost:%s₩t%s" % (path['cost'], "->".join(path['path']))
        result_sum.append(result)
        path_v.append(path['path'])

    result_sum.sort()
```

```
print len(result_sum)

for i in range(len(result_sum)):
    print result_sum[i]
print

for i in range(len(path_v)):
    print path_v[i]

# Hub node
h_nodes = []
for w in range(len(total_nodes)):
    sum_h = 0
    for p in range(len(path_v)):
        sum_c = 0
        for z in range(len(path_v[p])):
            if total_nodes[w] == path_v[p][z]:
                sum_c = sum_c + 1
                if sum_c == 1:
                    sum_h = sum_h + 1

    if sum_h == len(result_sum):
        h_nodes.append(total_nodes[w])

print
print "Hub nodes numbers : ", len(h_nodes)
print h_nodes, "₩n"
print "Total route : ", len(result_sum), "₩n“

# Compare with Dijkstra's route
print "Compare with Dijkstra route", "₩n"
total_num = len(path_v)
total_1 = len(path_v[0])

first_c = []
first_c.append(path_v[0])
second_c = []
second_c.append(path_v[0])
```

```
# di_number (first /second group)
di_number = 8

for t_num in range(1, total_num):
    compare_num = 0
    for num_1 in range(total_1):
        total_c = len(path_v[t_num])
        for num_2 in range(total_c):
            if path_v[0][num_1] == path_v[t_num][num_2]:
                compare_num += 1
    if compare_num <= di_number:
        first_c.append(path_v[t_num])
    else:
        second_c.append(path_v[t_num])

print "First Class : ", len(first_c)
for q in range(len(first_c)):
    print first_c[q]
print
first_num = len(first_c)

print "Second Class : ", len(second_c)
for w in range(len(second_c)):
    print second_c[w]
print
second_num = len(second_c)

while True:
    a_list = []
    check = 0
    for i in range(0,30):
        number = random.randrange(0, len(first_c)-1)
        a_list.append(number)

    for k in range(len(a_list)):
        for p in range(len(a_list)):
            if a_list[k]== a_list[p]:
                check += 1
```

```
        if check == len(a_list):
            break
    print a_list, "₩n"

    for w in a_list:
        print first_c[w]
    print

    while True:
        b_list = []
        check = 0
        for i in range(0,30):
            number = random.randrange(0, len(second_c)-1)
            b_list.append(number)

        for k in range(len(b_list)):
            for p in range(len(b_list)):
                if b_list[k]== b_list[p]:
                    check += 1
        if check == len(b_list):
            break
    print b_list, "₩n"

    for w in b_list:
        print second_c[w]
    print



    end_time = time.time()
    print "the time taken : ", end_time - start_time

    return 0

if __name__ == "__main__":
    main()
```

다. 제안 알고리즘 Ⅲ(__init__.py)

```
from graph import DiGraph
import algorithms
import time
import random

def main():
    start_time = time.time()

    # Load the graph
    G = DiGraph("net5")

    # Get the painting object and set its properties.
    paint = G.painter()
    paint.set_source_sink("A1", "E7")

    total_nodes = ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9',
                   'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9',
                   'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9',
                   'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9',
                   'E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8', 'E9',
                   'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9',
                   'G1', 'G2', 'G3', 'G4', 'G5', 'G6']


    # Generate the graph using the painter we configured.
    G.export(False, paint)


    cost_v = []
    path_v = []
    path_v1 = []
    result_sum = []

    # Get "number" shortest paths from the graph.

    # counter = input('How many routes do you need : ')
    counter = 250
```

```
items = algorithms.ksp_yen(G, "A1", "E7", counter)
for path in items:
    result = "Cost:%s₩t%s" % (path['cost'], "->".join(path['path']))
    result_sum.append(result)
    cost_v.append(path['cost'])
    path_v.append(" ".join(path['path']))
    path_v1.append(path['path'])

result_sum.sort()

for i in range(len(result_sum)):
    print result_sum[i]
print


# Hub node
h_nodes = []

for w in range(len(total_nodes)):
    sum_h = 0
    for p in range(len(path_v1)):
        sum_c = 0
        for z in range(len(path_v1[p])):
            if total_nodes[w] == path_v1[p][z]:
                sum_c = sum_c + 1
                if sum_c == 1:
                    sum_h = sum_h + 1

    if sum_h == len(result_sum):
        h_nodes.append(total_nodes[w])

print "Hub nodes numbers : ",len(h_nodes)
print h_nodes,"₩n"
print "Total route : ", len(result_sum),"₩n“

# Compare with Dijkstra's route
print "Compare with Dijkstra's route"
print
total_num = len(path_v)
total_1 = len(path_v1[0])
```

```
first_c = []
first_c.append(path_v[0])
second_c = []
second_c.append(path_v[0])




# di_number = input('Input a number you want to overlap nodes (first/second) : ')
di_number = 8
print




for t_num in range(1,total_num):
    compare_num = 0
    for num_1 in range(total_1):
        total_c = len(path_v1[t_num])
        for num_2 in range(total_c):
            if path_v1[0][num_1] == path_v1[t_num][num_2]:
                compare_num += 1
    if compare_num <= di_number:
        first_c.append(path_v[t_num])
    else:
        second_c.append(path_v[t_num])

print "  First Class :", len(first_c)
for q in range(len(first_c)):
    print first_c[q]
print
first_num = len(first_c)

print "  Second Class :", len(second_c)
for w in range(len(second_c)):
    print second_c[w]
print
second_num = len(second_c)
```

```
# Random First_Select
random_first_number = random.randrange(0, first_num)
print "First random number : ", random_first_number
print "First Standard route : ", first_c[random_first_number]
print

# Random Second_Select
random_second_number = random.randrange(0,second_num)
print "Second random number : ", random_second_number
print "Second Standard route : ", second_c[random_second_number]
print

# Select the routes of first_c
first_list = []
first_select = []
for d in range(first_num):
    first_uni = first_c[d].split()
    first_list.append(first_uni)

print

for k in range(len(first_list)):
    check_num = 0
    jkey = 1

    for j in range(len(first_list[k])):
        if jkey == 0:
            break
        if j == (len(first_list[k]) - 1):
            check_num=3

        hkey = 1
        for h in range(len(first_list[random_first_number])):
            if hkey == 0:
                break

                    hkey = 0
    if check_num == 3:
        first_select.append(first_list[k])
```

```
            if first_list[k][j] == first_list[random_first_number][h]:

                hub_counter = 0

                for g in range(len(h_nodes)):
                    if first_list[k][j] == h_nodes[g]:
                        hub_counter +=1
                        break
                if hub_counter == 0:
                    jkey = 0
                    hkey = 0
    if check_num == 3:
        first_select.append(first_list[k])


# Select the routes of second_c
second_list = []
second_select = []
for d in range(second_num):
    second_uni = second_c[d].split()
    second_list.append(second_uni)

for k in range(len(second_list)):
    check_num = 0
    jkey = 1

    for j in range(len(second_list[k])):
        if jkey == 0:
            break
        if j == (len(second_list[k]) - 1):
            check_num=3

        hkey = 1
        for h in range(len(second_list[random_second_number])):
            if hkey == 0:
                break
```

```
            if second_list[k][j] == second_list[random_second_number][h]:

                hub_counter = 0

                for g in range(len(h_nodes)):
                    if second_list[k][j] == h_nodes[g]:
                        hub_counter +=1
                        break
                if hub_counter == 0:
                    jkey = 0
                    hkey = 0
    if check_num == 3:
        second_select.append(second_list[k])

print "This List is selected(Not overlap) "
print "first_select : ", first_select
print "second_select :", second_select
print

# Random Select_first_class
print "This List is selected(overlap) "
print "  Random Selected First List"

# r_first = input('How many nodes do you want to overlap : ')
r_first = 2

first_list_a = []
first_select_a = []

for d in range(first_num):
    first_uni_a = first_c[d].split()
    first_list_a.append(first_uni_a)

for k in range(len(first_list_a)):
    list_counter = 0
    list_compare = []
    for j in range(len(first_list_a[k])):
        for h in range(len(first_list_a[random_first_number])):
```

```
            if first_list_a[k][j] == first_list_a[random_first_number][h]:
                list_compare.append(first_list_a[k][j])
                add_num = 0
                for m in range(len(list_compare)):
                    if list_compare[m] == first_list_a[k][j]:
                        add_num += 1
                if add_num < 2:
                    list_counter += 1

    if list_counter <= r_first:
        first_select_a.append(first_list_a[k])

for n in range(len(first_select_a)):
    print first_select_a[n]

print "number : ", len(first_select_a)
print

# Random Select_second_class
print
print "  Random Selected Second List"



# r_second = input('How many nodes do you want to overlap : ')
r_second = 13

second_list_a = []
second_select_a = []
for d in range(second_num):
    second_uni_a = second_c[d].split()
    second_list_a.append(second_uni_a)

for k in range(len(second_list_a)):
    list_counter_s = 0
    list_compare = []
    for j in range(len(second_list_a[k])):
        for h in range(len(second_list_a[random_second_number])):
            if second_list_a[k][j] == second_list_a[random_second_number][h]:
```

```
                    list_compare.append(second_list_a[k][j])
                    add_num = 0
                    for m in range(len(list_compare)):
                        if list_compare[m] == second_list_a[k][j]:
                            add_num += 1
                    if add_num < 2:
                        list_counter_s += 1


        if list_counter_s <= r_second:
            second_select_a.append(second_list_a[k])

    for n in range(len(second_select_a)):
        print second_select_a[n]

    print "number : ", len(second_select_a)

    end_time = time.time()

    print
    print "took time :",end_time - start_time
    print
    print "  First Class :", len(first_c)
    print "  Second Class :", len(second_c)



    return 0

if __name__ == "__main__":
    main()
```

감 사 의 글

어느 새 2년의 석사과정을 마치고 석사 논문을 제출하게 되었습니다. 지난 2년의 시간 동안 저에게 도움을 주신 분들이 정말 많습니다. 이 분들이 있었기에 만족 할 만한 논문을 작성 할 수 있었습니다.

먼저, 누구보다도 많은 관심을 가져 주시고, 부족한 부분은 따듯한 지도로 언제나 세심하고 꼼꼼한 손실로 지도해 주신 저의 지도교수님이신 김광조 교수님께 진심으로 감사드립니다. 또한, 바쁘신 와중에서도 공동 지도를 맡아 주시고 네트워크 분야의 지식을 쌓는데 도움을 주신 신승원 교수님께 감사드립니다.

다음으로 연구실 생활에 적응을 할 수 있도록 많은 도움 뿐만 아니라 영어 지도를 해준 최락용, 김학주 군, 힘들 때마다 옆에서 많은 신경을 써준 박준정 선배님, 안수현, 김경민 군 및 홍진아 양에게도 감사를 드립니다.

마지막으로 한없이 부족한 저를 믿어주는 가족과 언제나 저와 함께 계시고 저를 지켜 주시는 하느님 아버지께 감사드립니다.

2015년 12월

정 제 성

이 력 서

성 명 : 정제성
생년월일 : 1981. 3. 3.
연 락 처 : taegm01@kaist.ac.kr
주 소 : 34141 대전광역시 유성구 대학로 291 한국과학기술원(KAIST)

학 력

1997. 3. - 2000. 2. 부천고등학교
2002. 3. - 2006. 2. 육군사관학교 전자공학과 (학사)
2014. 3. - 2016. 2. 한국과학기술원 전산학부 정보보호대학원 (석사)

경 력

2014. 7. - 2014. 9. Workshop on Cryptographic Hardware and Embedded Systems 2014 (CHES 2014), 부산, 파라다이스 호텔, 9.23.-26. 운영 지원

연 구 과 제

2014. 7. - 2015. 6. Intrusion Detection System for Critical Infrastructures Using Big Data Analytics (KAIST-KUSTAR Institute)
2015. 4. - 2015. 8. 국가재난안전통신망의 장기간 보안성을 보장하는 산업 발전 전략 (KAIST 미래전략연구센터)

학 회 활 동

1. 정제성, 김경민, 김학주, 박준정, 안수현, 이동수, 최락용, 김광조, 김대영, “경량 암호를 이용한 IoT Secure SNAIL 플랫폼 구성(I),” 2014 한국정보보호학회 동계학술대회(CISC-W ’14), 2014.12. 6, 한양대학교, 서울

2. 정제성, 김광조, “Software Defined Network의 보안 취약성,” 2015 정보보호학회학술발표회 영남지부, 2015.2.13. 부산대학교, 부산
3. 정제성, 김광조, “Cuckoo Sandbox를 회피하는 악성코드 탐지 방안 연구,” 2015 한국정보보호학회 하계 학술대회(CISC-S ‘15) 2015. 6.25, 한국과학기술원, 대전
4. 정제성, 신승원, 김광조, “Yen의 알고리즘을 응용 한 임의 경로 선택 방식”, 2015 정보보호학술대회 충청지부, 2015.10.16., 서원대학교
5. 정제성, 신승원, 김광조, “S-RAY : Yen 알고리즘을 응용 한 보안 네트워크 라우팅”, 2015 한국정보보호학회 동계 학술대회(CISC-W ‘15), 2015.12.5., 서울여자대학교