SNMP 기반의 엔터프라이즈 IP네트워크 연결 정보 관리 시스템

이성주[0], Suman Pandy[1], 최미정[1], 홍원기[1]

포항공과대학교 정보통신대학원[0], 포항공과대학교 컴퓨터공학과[1]

{forstar, suman, mjchoi, jwkhong}@postech.ac.kr

요 약

엔터프라이즈 IP 네트워크 관리를 효율적으로 제공하고 IP 네트워크내의 결함 원인과 위치를 탐지하기 위하여, 네트워크 연결 정보를 유지하는 것은 필수적이다. 최근 10년간 엔터프라이즈 네트워크 구성도를 자동으로 탐지하는 방법에 대한 연구가 많이 이루어졌다. 본 논문에서 우리는 기존의 네트워크 구성도 탐색 메커니즘들의 다양한 문제점에 대하여 알아보고, L2/L3/L4/L7 스위치, 라우터, 프린터, 호스트와 같은 다양한 장치들과 이러한 장치들간의 물리적인 링크를 탐색하기 위한 SNMP 기반의 시스템을 제안한다. 본 논문에서 제시한 시스템을 통해 탐색된 네트워크 연결 정보들을 그래프나 트리와 같은 형태로 보임으로써 사용자에게 직관적인 네트워크 토폴로지 정보를 제공한다. 또한 네트워크 구성도를 물리적인 연결 상태뿐 아니라 VLAN과 Subnet과 같은 논리적인 구성도를 탐색하는 방법에 대해서도 논의함으로써 네트워크의 다양한 연결 정보를 제공하는 방법을 제시한다.

1. 서 론

현재 네트워크 관리 시스템은 자원 관리, 서버 관리, 결함 모니터링, 결함 원인 분석, 네트워크 부하 조정, 바이러스 예방 등의 다양한 관리 기능들을 포함한다. 네트워크 전체에 대해서 이런 관리 기능을 효율적이고 직관적으로 보여주기 위해서는 네트워크 구성과 장비들의 연결 정보들을 한 눈에 보기 위한 도구들을 필요로 한다. 네트워크 연결 정보를 자동으로 생성하기 위한 기존 연구들은 ping, icmp echo 요청, SNMP 등의 방법을 사용하여 진행되어 왔다. 이러한 기존 연구들은 다음과 같은 이슈들이 존재한다.

- 네트워크는 다양한 제조사들의 장비들로 구성되며, 각 장비는 각각의 고유한 private MIB을 가지고 있다. SNMP 기반으로 네트워크 연결 정보를 탐색하기 위해서는 네트워크를 구성하는 각 장비의 private MIB을 알아야 한다. 그러나 이를 모두 알고 적용하는 것은 쉽지 않다. IETF에서는 물리적인 구성 MIB[8]을 소개하였지만 CISCO 장비만이 이를 지원하고 있다. 그러므로 SNMP 기반의 네트워크 구성 탐색 방법에 있어서 private MIB에 의존하지 않고 표준 MIB 정보를 기반으로 연결 정보를 생성하는 방안이 마련되어야 한다.
- 현재 LAN 환경에서 보안과 불필요한 네트워크 트래픽 감소를 목적으로 VLAN을 설정하여 많이 사용하고 있다. 네트워크 연결 정보를 구성함에 있어서 물리적인 연결 정보뿐만 아니라 논리적인 연결 정보인 VLAN에 대한 연결 정보도 관리자에게는 네트워크 구성 설정을 위해 필요한 자료이다. 그러나 기존의 연구에서는 VLAN 탐색에 대한 부분이 제대로 논의되지 않았다.
- 기존의 네트워크 연결 정보 탐색에 대한 연구는 레이어 2와 레이어 3 각 레이어별 네트워크 구성도를 탐색하는 방법에 대해 진행되었다. 그러나 레이어 2와 레이어 3 장비간의 상호 연결 정보 제공에 대해서는 연구가 충분히 이루어지지 않았다.

따라서 본 논문에서는 이러한 주요 이슈들에 대하여 다루고 있다.

네트워크 구성도는 네트워크 내의 요소들(링크, 노드 등)간의 물리적이거나 논리적인 상호 연결들을 반영하는 두 가지로 분류될 수 있다. 물리적인 네트워크 구성은 각 장비들이 전달 매체를 통해 장비 포트별로 연결되어 있는 것이고, 논리적인 구성은 subnet과 VLAN과 같은 논리적인 세그먼트들로 나뉘어진 네트워크의 다른 추상화 계층이다. 또한 네트워크 연결 정보 탐색은 인터넷이나 백본망 레벨의 탐색과 LAN이나 enterprise 조직망 레벨의 구성 탐색으로 분류될 수 있다. 본 논문에서는 이 중 enterprise 레벨의 LAN 구성 탐색 메커니즘만을 목표로 하고 있다. 본 논문의 SNMP 기반의 엔터프라이즈 IP 네트워크의 연결 정보를 탐색하여 자동으로 토폴로지 맵을 생성하는 방법에 대한 알고리즘을 설계하고 연결 정보 관리 시스템을 구현하는 방안에 대해서 제시한다.

본 논문의 구성은 다음과 같다. 2 장에서는 네트워크 연결정보 탐색에 대한 기존의 연구에 대해 알아보고 3 장에서는 시스템 아키텍쳐와 알고리즘을 정의한다. 4 장에서는 시스템을 설계하고 구현하며,

본 연구는 두뇌한국 21 과 지식경제부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음" (IITA-2008-C1090-0801-0045)

5장에서는 구현된 시스템을 테스트하고 그 결과를 분석한다. 마지막으로 6장에서는 결론을 맺는다.

2. 관련 연구

이 장에서는 관련 연구로 네트워크 연결 정보 탐색에 대한 기존의 연구에 대해 알아본다. 네트워크 구성 탐색 방법은 네트워크의 출현 이래 광범위하게 연구되어 왔다. 구성 탐색에 관한 다수의 흥미로운 메커니즘들은 ping, trace route, SNMP, network traffic 기반의 추론 기술 등을 토대로 연구되었다[9]. 각 제조사들이 만든 다양한 장비들로 구성된 네트워크의 연결 정보를 SNMP 기반으로 제공하기에는 몇 가지 문제점들이 있다. RFC 2922에서는 표준화된 PTOPO-MIB[8]을 정의하고 있지만 대부분의 장비들이 이를 지원하지 않아 널리 적용되지 못하고 있다. 표 1에서는 우리가 제안하는 시스템과 기존의 시스템에 대한 비교를 보여준다.

표 1. 기존 연구 비교

	Discovering Internet Topology(1999)[1]	**Topology Discovery in Heterogeneous IP Networks: The NetINventory System(2004)[2]**	**Topology Discovery for Large Ethernet Networks SMU(2001)[3]**	**A Complete IP Network Topology Discovery Solution Constella(2007)[4]**	**SNMP기반의 엔터프라이즈 IP 네트워크 연결 정보 관리 시스템**
기법	Ping / Broadcast Ping / Traceroute / Zone transfer DSN server / SNMP / Subnet Guessing Algorithm	SNMP(ICMP Spoofing Heuristics)	SNMP	SNMP(Ping)	Only SNMP
지원하는 장비 타입	Router / 호스트	라우터 / L2스위치 / 호스트	L2 장비, 허브, Dumb Devices	라우터 / L3 / L2스위치 / 호스트 / 프린터	라우터 / L2,L3,L4,L7스위치 / 프린터 / 호스트
Subnet / Inter Subnet 연결정보	Subnet Guessing알고리즘 사용 Inter Subnet 연결 언급 없음	Inter Subnet연결 정의를 위해 subnet 정보 사용	Subnet정보 사용 없음	Intelligent Subnet Guessing알고리즘 사용	Subnet/ Inter Subnet연결 정보 사용
성능	SNMP, ping, echo등의 성능 비교(SNMP가 가장 좋은 성능, Echo는 시간이 오래 걸림)	ICMP spoofing과 AFT테이블 계산에 많은 시간 소요	탐색 소요 시간에 대한 결과 없음(기 탐색된 네트워크에 대한 연결 탐색 시간만 제시)	탐색된 노드수가 적어서 성능 비교 어려움	7000개 이상의 다수의 노드 탐색 수월
VLAN	언급 없음	VLAN고려,단 매우 복잡한 공식에 의한 VLAN탐색	언급 없음	언급 없음	VLAN 탐색 방법 제시
복잡도	많은 트래픽 발생	ICMP패킷 전송시 인터벌 전송으로 적당한 트래픽 발생	적은 트래픽 발생	탐색하는 노드수에 비례하는 트래픽 발생	매우 적은 트래픽 (적은 수의 SNMP query를 통해 MIB정보 수집, Spoofing 또는 Ping 기법 사용 안함)

탐색하기 위한 기존의 연구에서는 ping, icmp, trace route, SNMP등의 다양한 기술들을 조합하는 방법 제안하였다[1]. 이러한 방법들은 규모가 작거나 크게 복잡하지 않은 네트워크내에서 장비 타입에 대한 세부사항에 대한 언급 없이 레이어 3의 장비들만을 탐색하는 것이 대부분이었다. [2]에서는 레이어 2와 레이어 3 계층의 장비들을 탐색하며, 네트워크 구성도 탐색을 위한 혼합된 방법을 제시하였다. 이 방법은 spoofed ICMP 요청을 네트워크 노드로 보내고, 그들의 알고리즘을 이용하여 각 노드들의 AFT(Address Forwarding Table)을 만든다. 이것은 효율적이지만 spoofed ICMP 패킷을 보내는데 있어 엄청난 트래픽을 생산하는 단점이 있다. 현재의 대부분 네트워크 설정에서 spoofed ICMP 패킷은 블록화 되기 때문에 spoofed ICMP로부터 데이터를 수집하여 AFT 테이블을 만드는 것은 비현실적이다. 이 알고리즘은 완성된 AFT를 필요로 한다. 완성된 AFT테이블을 만들기 위해서는 다수의 spoofed 패킷들을 생산하고, 데이터베이스내의 연결 정보의 부하를 야기한다. 이러한 과정은 자원을 집중시키고 시간을 소비하게 된다. 불완전한 AFT 테이블의 경우 연결 정보를 찾기 위해 heuristics을 사용하게 되는데 본 논문에서는 이에 대해 논하지 않았다. [3] 에서는 [2] 와는 반대의 접근 방법을 제안하는데, 여기서는 AFT를 만들지 않고 spoofed ICMP 패킷을 사용한다. 하지만 여기서는 노드 간의 연결 정보를 찾는데 어떻게 불완전한 AFT가 사용되는지 증명하였다. 본 논문은 [3] 의 개념을 확장하여 네트워크내의 L2장비들의 연결 정보를 계산하는데 기존의 개념을 이용하였다.

본 논문은 다양한 측면에서 주목할 만하다. 본

논문에서는 탐색 시스템의 구현 부분을 연구하고, 어떤 식으로 SNMP Bridge MIB을 사용하여 AFT테이블을 만들것인지 서술한다[12]. 일반적인 구성 탐색을 위하여 표준 MIB을 사용하며, Cisco와 같은 제조사의 VLAN의 탐색과 같은 특정한 기능을 위하여 사설 MIB을 사용한다.

본 논문의 탐색 알고리즘은 Subnet 레벨의 탐색과 이를 시각적으로 표현하는 방법에 대해서도 논하고 있다. 여기서는 완전한 MIB 객체와 정교한 SNMP 아키텍처를 이용한 알고리즘을 제공한다. 비록 IP 등의 L3계층과 MAC과 같은 L2계층의 구성도를 탐색하는 방법에 대한 많은 연구가 이루어졌지만 L2계층과 L3계층을 서로 연결하는 부분에 대한 세부적인 연구는 없었다. 본 논문에서는 이에 대한 완성된 시스템과 뷰를 제안한다.

네트워크 구성도 탐색기능을 제공하는 HP사의 Open view나 IBM사의Tivoli, 그리고 AdventNet사의 OpManager와 같은 대부분의 최신의 툴에서 좋은 네트워크 뷰를 제공하는 것은 또다른 문제이다. 본 논문에서는 네트워크에 대하여 VLAN이나 Subnet 그리고 Graph뷰와 같은 다양한 뷰를 제공한다. Graph뷰의 복잡성을 최소화하기 위하여 Graph뷰는 L3계층만을 보여준다. Graph뷰 내에서 L2계층의 노드나 스위치에 대해 보고 싶다면, 해당 노드에 마우스를 클릭함으로써 Tree뷰로 전환하여 구성도를 볼 수 있다. 스위치들은 또 다른 스위치나 끝단의 호스트들과 서로 연결되어 있기 때문에 네트워크의 끝단까지의 구성도는 Tree형태로 보여지게 된다. 스위치들은L2레벨의 구성도를 탐색하는데 사용되는 브리지 MIB을 지원한다. 브리지 MIB은 spanning tree protocol을 지원한다. 그러므로 L2계층과 네트워크 끝단의 구성을 표현하는데는 Tree뷰가 가장 적합하다. 이를 통해 노드들을 전개하거나 네트워크 계층을 보기 위한 기능들을 제공받는다.

3. 디자인

이 장에서는 본 논문에서 제안하고 있는 연결 관리 정보 시스템의 아키텍처에 대해서 설명하고, 연결 정보를 탐색하기 위한 SNMP MIB 정보에 대해서 알아본다. 마지막으로 연결 정보 탐색 알고리즘에 대해서 자세히 설명한다.

3.1. 시스템 아키텍처

그림 1에서 보듯이 네트워크 구성도 탐색 시스템은 세 가지 메인 모듈로 이루어진다. 첫 번째 모듈은 다양한 입력과 뷰를 담당하는 Web Interface 모듈이다. 프로그램에 입력해야 하는 구성 정보들로는 네트워크 구성도를 유지하는데 필요한 데이터베이스 정보와 게이트웨이 IP 주소를 포함하는 네트워크 정보, SNMP community string, 그리고 한 AS에서 탐색해야 하는 IP 주소 범위와 SNMP 포트와 같은 설정 정보들이 있다. 탐색을 시작하기 위해서는 기본적인 입력 절차가 필요하다. 이렇게 입력된 정보들은 장비들을 탐색하고 연결 정보를 찾는 탐색 모듈로 보내진다. 또한 Web Interface 모듈에서는 Administrator가 특정 노드에 대한 customizing을 하고자 할 때 필요한 정보 입력을 담당한다. 기본적인 Administrator customization 기능은 노드의 위치나 제조사 정보, 그리고 타입 등을 변경하는 것이다. 또한 이 모듈에서는 탐색된 네트워크 구성도에 대하여 그래프나 트리 형태의 다양한 사용자 인터페이스 뷰를 제공한다.

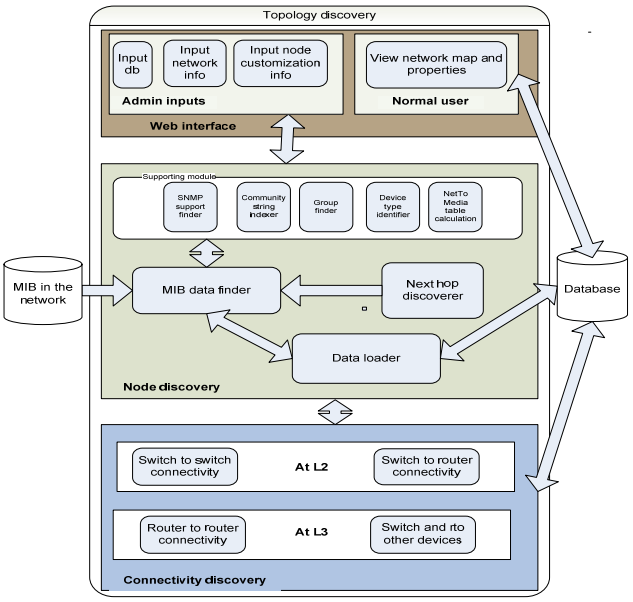

그림 1. 시스템 아키텍처

Node Discovery 모듈은 웹 인터페이스 모듈로부터 정보를 받아 네트워크 내의 노드들을 탐색한다. 여기서는 3.3장에서 설명할 Next hop discover 메커니즘을 사용하여 탐색을 한다. 새로운 노드를 찾은 후에 그 노드와 연관된 MIB 정보는 Data Loader를 이용하여 데이터베이스에 저장한다. Node Discovery 모듈은 다양한 기능들을 지원하는 내부의 작은 모듈들을 포함한다. 탐색된 노드의 SNMP 지원 여부를 확인하는 SNMP support finder나 Cisco사의 장비에 대하여 브리지 MIB을 불러오는 community string indexer와 같은 다양한 내부 모듈들이 Node Discovery 모듈을 지원한다[5]. 만약 동일 장비에 다수의 IP 주소가 있다면 group finder가 이러한 IP 주소들을 한 개의 장비로 그룹화하며, device type identifier는 장비의 타입을 결정하고, net to media 모듈은 더 많은 장비들과 그들의 연결 정보를 탐색하는 것을 지원한다.

일단 노드들이 탐색되고 관련된 MIB 정보들을

표 2. MIB 정보

MIB	MIB Object
MIBII (REF-1213-MIB)	**system** → sysServices
	system → sysDescr
	iftable → ifIndex
	iftable → ifDescr
	iftable → ifPhyaddress
	ip → ipForwarding
	ip → ipRouteTable → ipRoutNextHop
	ip → ipRouteTable → ipRoutType
	ip → ipAddrTable → ipAdEntAddr
	ip → ipAddrTable → ipAdEntNetMask
	ip→ ipNetToMediaTable → ipNetToMediaNetAddress
	ip→ ipNetToMediaTable → ipNetToMediaPhysAddress
CISCO-VTP-MIB	**vtpVlanState**
BRIDGE-MIB for Cisco switch (modify the mib name)	**dot1dBrige** → dot1dBase → dot1dBasePortTable → dot1dBasePortEntry → dot1dBasePort
	dot1dBrige → dot1dBase → dot1dBasePortTable → dot1dBasePortEntry → dot1dBasePortIfIndex
	dot1dBrige → dot1dTp → dot1dTpFdbTable → dot1dTpFdbEntry → dot1dTpFdbAddress
	dot1dBrige → dot1dTp → dot1dTpFdbTable → dot1dTpFdbEntry → dot1dTpFdbPort
	dot1dBrige → dot1dTp → dot1dTpFdbTable → dot1dTpFdbEntry → dot1dTpFdbStatus
Q-BRIDGE-MIB	qBridgeMIB → qBridgeMIBObject → do1qVlan → do1qPortVlanTable → do1qPortVlanEntry → dot1qPvid
BRIDGE-MIB for other vendor devices except CISCO	dot1dBrige → dot1dStp → dot1dStpPortTable → dot1dStpPortEntry → dot1dStpPort
	dot1dBrige → dot1dStp → dot1dStpPortTable → dot1dStpPortEntry → dot1dStpPortState
	dot1dBrige → dot1dStp → dot1dStpPortTable → dot1dStpPortEntry → dot1dStpPortDesignatedRoot
	dot1dBrige → dot1dStp → dot1dStpPortTable → dot1dStpPortEntry → dot1dStpPortDesignatedBridge
	dot1dBrige → dot1dStp → dot1dStpPortTable → dot1dStpPortEntry → dot1dStpPortDesignatedPort

불러오면, Connectivity Discovery 모듈에서 관련 MIB 정보들을 사용하여 장비들간의 연결 정보를 탐색하게 된다. 여기서 필요한 네 가지 기본 알고리즘은 3.4장에서 자세하게 설명한다.

3.2. SNMP MIB

본 논문에서 제안하는 탐색 메커니즘은 SNMP에 기반한다. 표2에서는 본 논문의 탐색 알고리즘을 지원하는 모든 SNMP MIB에 대하여 설명하고 있다. 본 논문에서는 연결 정보를 얻기 위해 RFC1213-MIB[11], BRIDGE-MIB[12], Q-BRIDGE-MIB[13]과 같은 표준 MIB과 Cisco 스위치의 VLAN과 관련된 정보를 위한 CISCO-VTP-MIB과 같은 private MIB 정보를 이용하였다. 데이터베이스 내의 모든 필드들은 표2에서 설명하는 MIB에 대응한다. 각각의 탐색된 장비에는 singleton class identifier를 사용하여 유일한 identifier를 생성한다. 우리의 프로그램은 단 하나의 identifier class의 인스턴스를 가지고 있으며 이 class는 각각의 새로운 노드에 대하여 유일한 키를 생성하는 역할을 한다. 한 장비가 다중 IP 주소를 가질 수 있기 때문에 IP 주소는 identifier로 취급하지 않는다. 새로운 장비를 탐색하자마자 그 장비가 SNMP를 지원하는지 확인하기 위해 SNMP GET 메시지를 보내고, 지원할경우 표2에서 언급했던 모든 MIB 정보를 수집하고 그 정보를 데이터베이스의 해당 테이블에 저장한다. 일단 모든 장비에 대한 정보를 데이터베이스에 업데이트 하면 그림 1의 Connectivity Discovery 모듈에서 연결 정보를 찾기 시작하고, ConnectivityTable을 생성한다.

3.3. 탐색 알고리즘

이 장에서는 장비를 탐색하고 탐색된 장비에 대한 연결 정보를 찾기 위해 사용된 알고리즘에 대하여 설명한다. 그림 2은 본 논문에서 제안하는 탐색 시스템에 대한 전체 흐름도를 보여준다. 탐색 알고리즘이 동작을 시작하면 Input Network Info 모듈은 게이트웨이 IP, SNMP community string 또는 비밀번호, SNMP 포트 그리고 탐색 범위의 제한 등의 정보를 입력받는다. 입력받은 정보는 게이트웨이 노드로부터 도달 가능한 모든 next hop 노드를 찾기 위해 게이트웨이 IP 주소의 ipRoutNextHop MIB정보를 가져오는 *Discover Device using the findResourceWithNexthopMechanism(R)* 모듈로 보내진다. 만약 새로운 노드가 ipRoutNextHop 테이블에 있다면 그 노드의 MIB 정보를 데이터베이스로부터 불러온다. 이 모듈은 ipRoutNextHop 테이블의 도움으로 연결된 장비들을 탐색하고, next available hop으로 이동하기 위해 이 테이블 정보를 이용한다. 동일한 과정이 연결 정보를 찾기 위해 반복되며, 이 과정은 ipRoutNextHop 테이블에 더 이상 next hop에 대한 entry가 없을 때까지 진행된다.

일단 Discover Device using the findResourceWithNexthopMechanism(R) 모듈이 장비 탐색을 마치면 *Discover Device using the findResourceFromNetToMedia()* 모듈로 제어권이 이동한다. 이 모듈에서는 netToMedia 테이블을 이용하여 장비를 탐색하는데, 여기에 Discover Device using the findResourceWithNexthopMechanism(R) 모듈에서 탐색된 모든 장비들을 저장하며, 각 장비의 SNMP 지원 여부를 알기 위해 SNMP GET 메시지를 보낸다. 만약 그 장비가 SNMP 를 지원한다면, 3.2장에서 설명한 MIB 정보를 SNMP 에이전트로부터 읽어와서 해당 데이터베이스에 저장한다.

일단 모든 장비가 탐색되면 제어는 탐색된 장비 타입을 라우터, L2/L3/L4/L7 스위치, 프린터, 그리고 네트워크 터미널 노드 등으로 분류하는 Discover Device Type 모듈로 넘어간다. 모든 장비에 대한 타입을 결정한 후, 각 장비에 대한 연결성을 확인하는 connectivity function 이 호출된다. *Find L2 level connectivity findConnectionBetweenSwitchToSwitch() and findConnectionBetweenSwitchToRouter()* 모듈은 스위치와 스위치 또는 스위치와 라우터 사이의 연결 정보를 찾는 일을 수행하고 *Find L3 level connectivity findConnectionBetweenRouterToRouter() and findConnectionBetweenOtherDeviceWithSwitchAndRouter()* 모듈은 라우터와 라우터 또는 스위치와 다른 그 외 다른 장비간의 연결 정보를 찾는다.
그림 2 모듈의 수행 알고리즘에 대해서는 다음 각 장에서 자세히 설명한다.

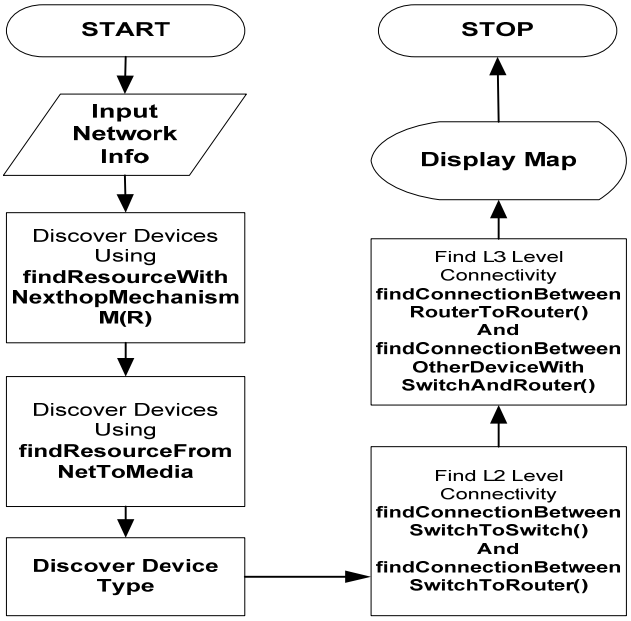

그림 2. 시스템 흐름도

3.3.1. Discover Devices

이 모듈은 게이트웨이 IP 주소, SNMP community string, port, IP의 범위 등의 설정 정보를 입력받아 장비를 탐색하는 역할을 수행한다. 알고리즘 1과 2는 장비를 탐색하는데 사용되고, 알고리즘 3은 탐색된 장비의 MIB 정보를 데이터베이스에 저장하는데 사용된다. 데이터베이스의 구조와 그에 해당하는 MIB정보는 3.2장에서 이미 논의되었다.

알고리즘 1에서는 ipRoutNextHop MIB을 이용하여 장비를 탐색하며 여기서는 게이트웨이 IP 주소를 입력받는다.

Algorithm 1: findResourceWithNexthopMechanism (R)
R- Gateway IP address
RS - Set of Resource, RV - Set of Resource visited
NR - Set of ipRoutNextHop, retrieved by snmpget.
loadMIBforNode(R)
RS = RS.push(R)
Run loop on each element of RS
  R=RS.pop
    If (!RV.contains(R))
      RV.add(R)
      NR=getNextHop(R)
      Run loop on each element of NR
        R_{NR}=NR.next()
        If(getSynonymGroupElement(RNR)==NULL)
          RS=RS.add(R_{NR})
          loadMIBForNode(R_{NR}, C)

알고리즘 2에서는 알고리즘 1에서 이미 탐색된 장비들의 ipNetToMediaTable을 이용하여 더 많은 리소스(R)를 찾는다. ipNetToMediaTable은 도달 가능한 장비들의 리스트를 제공하는 캐시 역할을 한다.

Algorithm 2: findResourceFromNetToMedia()
RV – Set of resources already discovered from algorithm 1
N_{RV} – *NetToMediaTable* set for RV stored in database.
R – New resource
Run loop on each element of RV
  Retrieve N_{RV} for each RV
  Run loop on each element of N_{RV}
    R= element of N_{RV}
    If(getSynonymGroupElement(R) is equal to null)
      LoadMIBForNode(R, C)

알고리즘 3은 알고리즘 1과 2로부터 호출된다. 이 알고리즘은 데이터베이스로부터 장비 R에 관련된 system, ifTable, ipRouteTable, ipAddrTable, ipNetToMediaTable, vtpVlanState, dot1dBridge과 같은 모든 MIB 정보를 불러온다. IP 주소, forwarding info, sysService, sysDesc와 같은 장비의 일반적인 정보는 데이터베이스의 IPTable에 저장된다. Next hop에 대한 정보는 RouteTable에 저장되고 다중 subnet을 위한 다중 IP 주소를 가진 라우터나 스위치의 다중 주소 정보는 AddressTable에 저장된다. 스위치나 라우터의 port 정보와 같은 인터페이스 정보는 InterfaceTable에 저장된다. 장비들간의 연결 정보를 탐색할 때에는 Bridge MIB을 지원하는 장비들간의 인터페이스 연결 정보 역시 탐색하게 된다. 각 노드로부터 도달 가능한 장비들의 리스트는 NetToMediaTable에 저장되며, 이는 IP 주소와 같은 물리적인 주소로 대응된다. 모든 L2장비의 Bridge MIB 정보는 BridgeMibTable에 저장된다. VlanTable은 선택적인 테이블이다. 만약 네트워크가 Cisco사가 아닌 다른 장비들로 구성되었다면, VLAN 및 VLAN과 관련된 확장 트리를 계산하는데 이 테이블을 사용할 수 있다.

Algorithm 3: loadMIBforNode (R, C)
R is the newly discovered device
C is community String
If R does not exist in Database
  database.store (SNMPLoadMIBforIPtable ())
  database.store (SNMPLoadMIBforRoutTable ())
  database.store (SNMPLoadMIBforAddressTable ())
  database.store (SNMPLoadMIBforInterfaceTable ())
  database.store (SNMPLoadMIBforNetToMediaTable ())
  database.store (SNMPLoadBridgeMib ())
  database.store (SNMPLoadVlan ())

3.3.2. 다중 IP장비 탐색

새로운 장비를 탐색할 때, 어떤 장비는 그것이 속한 다중 subnet과 연관된 다중 IP 주소를 가질 수 있다. 그러한 장비의 다중 IP 주소는 AddressTable에 저장되며, 이러한 정보를 얻기 위해 ipAddrTable라는 MIB 정보가 사용된다. 새로운 IP 주소가 탐색되었을 때, 이미 탐색된 장비가 다른 IP 주소를 가지고 있을 수 있는데, 이러한 상태를 확인하기 위해서 알고리즘 1과 2의 getSynonymGroupElement라는 함수를 호출한다. 이 함수는 새롭게 탐색된 IP 주소가 없을 경우

null값을 리턴한다.

3.3.3. Discover Device Type

이 모듈은 네트워크내의 장비들의 타입을 탐색한다. 여기서는 sysService MIB object를 사용하여 이것을 7bit의 string으로 변환한다. 각각의 bit는 OSI 7 네트워크 모델의 7가지 계층으로 대응된다. 예를 들어 어떤 장비가 sysServices 값이 78(1001110)이라고 할 때, 2번째/3번째/4번째/7번째 bit이 1로 셋팅되게 됨으로 이 장비는 이러한 4가지 계층의 서비스를 제공하는 L7 스위치임을 의미한다. 이러한 알고리즘은 데이터베이스 내에 저장된 dot1dBridge, iftable, Printer MIB 정보에도 사용된다. 이 방법은 장비가 Layer 2에서 port to port 연결 정보를 지원하는지 확인하기 위해서 사용된다. 이러한 장비들은 스위치로 분류된다. Iftable MIB정보는 L3 장비가 다중 인터페이스에서 동일한 MAC 주소를 가지도록 구성되어 있는지 결정하는데 사용되고, Printer MIB은 해당 장비가 프린터인지를 확인하는데 사용된다.

3.3.4. Discover Connectivity

이 장에서는 데이터베이스에 저장된 MIB 정보를 사용하여 연결 정보를 탐색하는 알고리즘에 대하여 설명한다. 알고리즘 4는 Layer 2 장비들 간의 연결 정보를 계산한다. 이 알고리즘을 통해 찾아낸 연결 정보는 ConnectivityTable에 저장된다. InterfaceTable로부터 대응하는 인터페이스의 MAC 주소를 얻을 수 있으며, ConnectivityTable의 인터페이스와 인터페이스간의 대응을 저장한다. 이것은 순전히 L2 장비에서 L2 장비로의 연결 탐색 알고리즘이다.

Algorithm 4: findConnectionBetweenSwitchToSwitchS – Set of L2 and L3, L4 and L7 switches with Bridge MIB support
SV- Switches Visited
I_i– List of Interface for Switch S_i
For each pair of the Switches S_i and S_j
  For each pair of I_{im} and I_{jn}
    If BridgeMIB of S_i and S_j has the physical address I_{im} and I_{jn} of each other
      Set the connectivity between S_i and S_j for Interface I_{im} and I_{jn}
      Store the connectivity information in *ConnectivityTable*

알고리즘 5는 L2 장비와 L3 장비간의 연결 정보를 계산한다. L3장비는 sysService에서 Layer3을 지원하며, Bridge MIB은 제공하지 않고 다중 인터페이스를 위해 동일한 MAC 주소를 가지도록 구성된다. L3 장비에 대해서는 Bridge MIB을 갖고 있지 않기 때문에 L2 장비로부터 L3 장비로의 인터페이스(포트) 연결 정보는 저장하지만, L3 장비로부터 L2 장비로의 인터페이스 연결 정보는 저장할 수 없다.

Algorithm 5: findConnectionBetweenSwitchToRouter
S – Set of Switch
I_S – List of Interface of Switch S
R – List of Routers
RM – List of Mac address of Router R
For each S
  Find I_S of S for which the mapping was not found.
  For each I_S
    For each R
      If RM is found in BridgeMIB of I_S
        Set the connectivity between R and S for Interface I_S

알고리즘 6은 L3 장비와 L3 장비간의 연결 정보를 탐색하는 알고리즘이다. 여기서는 ConnectivityTable의 인터페이스와 인터페이스간의 연결(포트 연결) 정보 없이 연결 정보를 저장한다.

Algorithm 6: findConnectionBetweenRouterToRouter
R - List of Routers
For each pair of Router R_I and R_J
Check if the connectivity information exists in ConnectivityTable
  If connectivity does not exist.
    Check the *NextHopTable* to get the connectivity
    Set the connectivity between R_I and R_J

알고리즘 7은 스위치, 라우터 이외의 터미널 장비들의 연결 정보를 탐색하는 알고리즘이며 ipNetToMediaTable MIB으로부터 입력을 받은 NetToMediaTable을 이용한다. 여기서 얻은 연결 정보는 인터페이스와 인터페이스간의 연결 정보와 같은 L2 정보를 가지지 않는다. 이 연결 정보는 ConnectivityTable에 저장된다. 알고리즘 4, 5, 6, 7이 모두 실행되고 난 뒤, 사용자에게 네트워크 연결 뷰를 제공하기 위해 그래프를 만드는데 사용되는 최종 ConnectivityTable을 얻을 수 있다. 그래프를 만드는데 사용된 방법은 4.1장에서 설명한다.

Algorithm 7:
findConnectionBetweenOtherDeviceWithSwitchAndRouter
A - All devices in ConnectivityTable i.e. the devices already visited
A´ – Set of all the Devices not in ConnectivityTable
Retrieve subnet of the devices in A´
Create subsets A'_{S1}, A'_{S2}, A'_{S3} based on the subnet id.
For each subnet S_I
    Create mapping of the devices in A_{SI} with the Switch A_J
    Store the mapping information in *ConnectivityTable*.

3.3.5. Discover Subnet

특정 IP 주소의 subnet을 계산하기 위해 데이터베이스의 AddressTable에 저장된 ipAdEntNetMask MIB 정보를 이용한다. 예를 들어 192.168.5.10과 같은 장비의 IP 주소와 ipAdEntNetMask로부터 255.255.255.0과 같은 subnet

mask를 가지고 있다면, IP 주소와 subnet mask를 bit wise AND 연산을 통해 네트워크 주소(subnet 주소)인 192.168.5.0를 얻을 수 있다. 이 방법을 통해 모든 장비의 subnet 주소를 찾을 수 있다. 일단 모든 장비의 subnet 주소를 찾은 후, 일치하는 subnet끼리 분류할 수 있다. 그리고나서 이 장비들간의 연결 정보를 찾고, subnet 구성도를 보여줄 수 있다. Subnet 내부의 연결 정보를 찾기 위하여 한 개 이상의 subnet에 속하는 장비들을 걸러내면 subnet 내부 연결 정보를 찾게 되고, 네트워크 구성도에 보여줄 수 있다. 그러므로 이러한 단계의 추상화는 사용자로 하여금 네트워크에 대한 subnet 기반의 논리적 뷰를 제공한다.

3.3.6. Discover VLAN

각각의 VLAN은 VLAN identifier로 구분된다. 네트워크가 VLAN을 지원하고 패킷들이 그 네트워크로 전달된다면, VLAN identifier 태그가 패킷에 붙게 된다. VLAN을 인지하는 브리지는 VLAN 태그가 달린 프레임을 인식하고, 해당 패킷을 정확하게 전달한다. 각각의 VLAN은 그들만의 spanning tree를 가질 수 있거나, 제조사의 구현 방법에 따른 모든 VLAN과 연관된 single spanning tree 가질 수 있다. Cisco사는 다중 VLAN을 위한 다중 spanning tree를 지원한다. 본 논문에서는 비록 표준 MIB에 기반하는 VLAN을 지원하는 vendor neutral solution과 함께 Cisco 장비의 VLAN 연결 정보를 찾는 방법을 제시한다.

Cisco 스위치의 VLAN을 찾는 것은 community string indexing[7]에 근거한다. 각각의 VLAN에 대하여 세분화된 인스턴스를 갖는 MIB 정보에 접근하기 위하여 community string indexing을 사용한다. 그 다음 해당 스위치에 유효한 특정 VLAN을 찾는다. CISCO-VTP-MIB[14]으로부터 vtpVlanState object를 사용하는 스위치의 유효한 VLAN을 찾을 수 있다. vtpVlanName이나 다른 object 대신에 vtpVlanState object를 사용하는 이유는 한번의 SNMP 동작으로 유효한 VLAN의 index number를 결정할 수 있기 때문이다. 각각의 VLAN에서 community string indexing을 사용하거나, dot1dTpFdbAddress MIB정보를 가지고 SNMP 쿼리를 하여 community name의 suffix에 @vlanid를 더함으로써 dot1dTpFdbAddress로부터 MAC 주소를 얻을 수 있다. 각각의 VLAN에 대해 Bridge MIB을 찾아낸 후 VLAN의 장비들의 spanning tree를 만들고, 네트워크 구성도에서 이를 tree 뷰로 보여준다.

만약 장비가 Cisco 사설 MIB을 지원하지 않을 경우에 모든 VLAN에 연관된 하나의 spanning tree가 있는 IETF의 표준 MIB에 기반하는 해결책을 제시한다. 이를 위해 BRIDGE-MIB(RFC 1493)[12]과 Q-BRIDGE-MIB(RFC 2674)[13]을 사용한다. 먼저, 탐색하고자 하는 범위를 결정하고, 범위 내에서 3.3.1장에서 설명한 nextHop 메커니즘을 사용하는 IP 주소와 같은 Virtual Bridge LAN을 목표로 한다. 다음으로 장비의 타입을 결정하고, L2와 L3 장비의 Bridge MIB 정보를 가져온다. STP(spanning tree protocol) 정보는 BRIDGE-MIB의 dot1dStpPortTable에 저장된다. 각 행의 dot1dStpPortTable 변수를 체크하고, port 상태가 비활성화된 엔트리는 제외시킨다. 또한 dot1qPortVlanTable의 연관된 엔트리로부터 dot1qPvid를 가져옴으로써 선택된 port와 관련된 VLAN을 추려낸다. 이렇게 얻은 정보들로, 각각의 VLAN identifier와 port identifier의 조합에서 내부 테이블인 VLANTable을 만들어낼 수 있다. 또한 Bridge 주소, port identifier, VLAN identifier, 지정된 root, 지정된 bridge, 지정된 port와 같은 정보를 저장하여 VLAN spanning tree를 생성하는데 사용한다.

알고리즘 8에서는 VLAN spanning tree를 생성하는 방법을 설명한다. VLAN spanning tree를 생성하기 위해서는 VLAN의 그룹 엔트리를 현재의 엔트리로 선택하고, bridge 주소와 현재 엔트리의 지정된 bridge주소가 일치하지 않을 경우, 이 bridge 주소를 spanning tree의 지정된 bridge의 child로 한다. 그리고 이 엔트리의 port id를 이 bridge의 exit port로 표시하고, 지정된 bridge의 port id를 지정된 bridge의 entry port로 표시한다. 만약 bridge 주소와 현재 엔트리의 지정된 bridge 주소가 일치할 경우 next unvisited 엔트리를 현재의 엔트리로 선택한다. Spanning tree는 VLAN identifier의 모든 엔트리를 방문하게 되면 생성된다.

Algorithm 8: CunstructSpanningTreeForVlan
V - vlan identifier
VS - Set of entries from VLAN table for each vlan V
T - tree type data structure. The nodes of the tree is an entry of VS
For each entry in VS
  If the bridge address and the designated bridge address of the current entry are not same
    Set the bridge address as the child of the designated bridge address of the spanning tree T
    Mark the interface id of this entry as the exit port on this bridge
    Mark the designated bridge port id as the entry port of the designated bridge

4. 구현

개발 환경으로 하드웨어는 CPU 2.80GHz Intel Pentium 4, 512MB RAM을 사용하였으며, 소프트웨어는 MS Windows XP 운영체제 하에 JAVA 1.5버전과 tomcat 5.5 웹 서버를 사용하였으며, 데이터 저장을 위하여 Oracle 데이터베이스 10g Express Edition을 사용하였다. SNMP library로는 AdventNet Java APIs[10]를 사용하였으며, 네트워크 구성도를 그래프 형태로 보여주기 위하여 수학적인 그래프 객체를 제공하는 공개 Java graph library인

JgraphT[6]를 사용하여 구현하였다.

4.1. Network MAP View

이 장에서는 본 논문에서 제안하는 시스템의 결과 출력 방식에 대하여 자세히 논한다. 이 시스템에서는 List View, Tree View, Graph View, VLAN View의 네 가지 메인 메뉴를 제공한다. Tree View와 Graph View는 각각 Device View, All Device View, Subnet View의 세 가지 서브 메뉴를 갖는다. 또한 VLAN View는 Switch→VLAN, VLAN→Switch의 두 가지 서브메뉴를 갖는다.

List View에서는 탐색된 장비들을 라우터, 스위치, 프린터, 노드 등의 카테고리 리스트 형태로 보여준다. 리스트 내의 각 장비를 클릭하면 해당 장비의 세부 속성에 대한 내용을 볼 수 있다.

Graph View와 Tree View에서는 네트워크의 장비들과 연결 정보를 Graph와 Tree형태로 보여준다. Graph View는 L3 계층의 장비의 연결 정보를 보여주는데 사용되고, Tree View는 L2 계층의 장비를 보여주는데 사용된다.

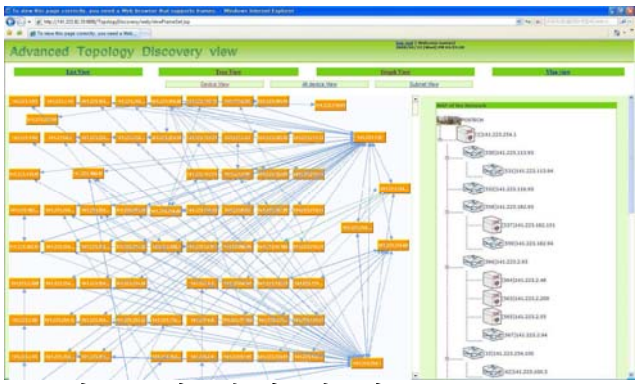

그림 3. 포항공대 네트워크의 Graph/Tree View

그림 3은 L3 장비의 Graph View와 L2 장비의 Tree View를 보여준다. 좌측의 프레임에는 L3 계층의 장비들을 Graph 형태로 보여주고 있으며, 우측의 프레임에는 L2 계층 장비의 연결 정보를 Tree 형태로 보여주고 있다. 우측 그래프의 각 노드를 더블 클릭하면 각 노드에 연결된 L2 장비들에 대하여 Tree 형태로 보여준다.

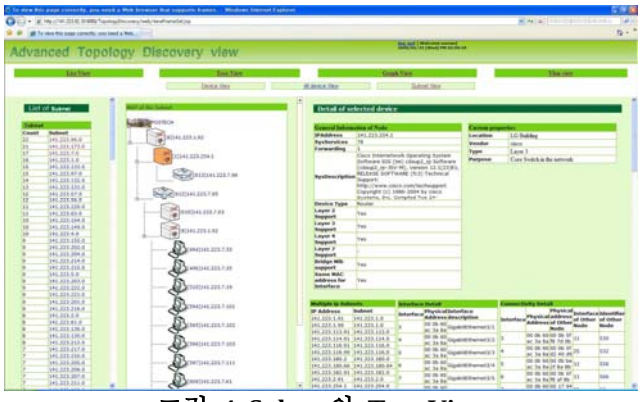

그림 4. Subnet의 Tree View

그림 4는 subnet의 Tree View를 보여준다. 각각의 장비를 더블 클릭하면 각 장비별로 IP 주소, sysServices, 전달 정보, system description, 다른 네트워크 계층 지원 여부, 장비가 속한 다중 subnet, 인터페이스 세부 내용, 인터페이스 연결 정보 등의 속성 정보들을 볼 수 있다. 가장 좌측의 프레임은 네트워크 내의 모든 subnet과 subnet내의 장비의 수를 리스트 형태로 표시하며, 중간 프레임은 subnet의 장비들 사이의 연결 정보를 Tree 형태로 보여준다. 우측의 프레임은 선택된 장비의 다양한 정보들을 보여준다. Subnet View는 administrator가 네트워크의 subnet 계층의 논리적 형태를 알고자 할 때 유용하다.

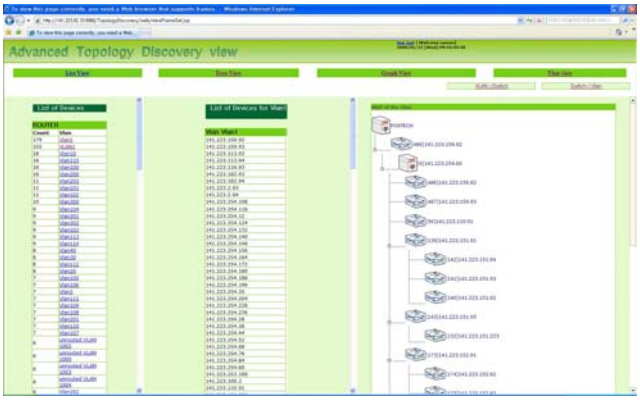

그림 5. VLAN View

그림 5는 네트워크를 VLAN별로 구분하여 보여주는 VLAN View이다. 가장 좌측의 프레임은 네트워크 내의 모든 VLAN을 리스트 형태로 표시하며, 중간 프레임은 해당 VLAN에 속하는 모든 장비들을 보여준다. 우측의 프레임에서는 VLAN내의 장비들간의 연결 정보를 Tree 형태로 보여준다.

5. 테스트 및 결과

이 장에서는 본 논문에서 개발한 시스템을 포항공과대학교와 고려대학교 서창캠퍼스의 서로 다른 두 엔터프라이즈 네트워크에 적용하여 테스트한 결과를 분석한다. 먼저 고려대학교 네트워크에 대하여 14개의 라우터, 29개의 스위치, 42개의 subnet과 28개의 VLAN등 총 2019개의 장비를 탐색하였다. 포항공과대학교 네트워크에 대해서도 동일하게 적용하여 76개의 L3 스위치, 522개의 L2스위치, 8개의 라우터, 5개의 L4스위치, 208개의 subnet, 149개의 VLAN등 총 7495개의 장비를 탐색하였다. 여기서는 다중 community string을 이용하여 서로 다른 네트워크 조건에서 수차례 테스트 하였다.

표 3은 포항공대의 네트워크에 대해 각 기능 모듈별로 수행 시간의 결과를 나타낸다. 여기서는 1개의 thread를 사용하였을 경우와 10개의 thread를

사용하였을 경우 각각의 결과를 보여준다. 본 논문에서는 총 소요시간의 대부분을 findResourceFromNetToMedia 모듈이 차지하기 때문에 이 모듈에는 thread 메커니즘을 사용하였다. SNMP-GET 요청에 대한 Time out 은 5초로 지정하였으며, NetToMediaTable로부터 라우터와 스위치 정보를 불러와서 테이블 내의 각 장비에 대해 SNMP-GET메시지를 보낼 때, 만약 해당 장비가 SNMP를 지원하지않는 경우 5초 이후에 응답을 받게 되므로 결국 이 과정에서 많은 시간이 소요되었다. 실험에서 findResourceFromNetToMediaTable 모듈은 총 7495 개의 장비 중 7496개의 장비를 탐색하였다. Scalability 테스트는 linear한 형태를 보인다. 탐색되는 장비의 수가 증가할수록 장비를 탐색하는데 걸리는 소요 시간도 선형적으로 증가하였다. 이러한 경향은 그림 7의 POSTECH 테스트 결과와 그림 7의 고려대 테스트 결과를 통해 확인할 수 있다. 그림 6에서는 총 7000개의 장비를 탐색하는데 소요되는 시간을 100개의 장비 단위로 기록하였다.

앞서 표 1에서, 본 논문에서 제안하는 시스템과 기존의 연구에 대한 비교를 하였다. 여기서는 기존의 테스트 결과와 본 시스템의 테스트 결과에 대해 비교한다. 기존 연구[1]의 테스트 결과에 따르면 SNMP를 이용하였을 경우 139개의 라우터, 93개의 subnet 등 총 602개의 장비를 탐색하는데 193분이 소요되었으며, DNS zone transfer/traceroute 방식을 사용한 경우에는 155개의 라우터 및 622개의 subnet등 총 7367개의 장비를 탐색하는데 2880분이 소요되었다. 그림 7에서 보여주듯이 본 논문에서 제안하는 시스템에서는 598개의 스위치,

표 3. 각 모듈별 수행 시간

Function	Details	Using 1 thread		Using 10 threads	
		Time in hours	Time in %	Time in hours	Time in %
findResourceWithNexthop Mechanism	discovers and load MIB for 40 devices	0:16:21	2.51%	0:16:21	12.54%
findResourceFrom NetToMediaTable	discovers and load MIB for 7455 devices	10:21:52	95.47%	1:40:52	77.35%
FindResourceType	calculates resource types for all 7495 devices	0:00:34	0.09%	0:00:34	0.43%
CalculateSubnet	calculates subnet for all 7495 devices	0:00:40	0.10%	0:00:40	0.51%
findConnectionBetween SwitchToSwitch	find connections among 522 L2 and 76 L3 switches	0:10:38	1.63%	0:10:38	8.15%
findConnectionBetween SwitchToRouter	find connection between 522 L2, 76 L3 switches with 8 Routers	0:00:57	0.15%	0:00:57	0.73%
findConnectionBetween RouterToRouter	find connection among 8 routers	0:00:12	0.03%	0:00:12	0.15%
findConnectionBetweenOther DeviceWithSwitchAndRouter	find the connection of the rest 6889 hosts with the routers and switches	0:00:10	0.03%	0:00:10	0.13%
Total time	total time taken	10:51:24	100%	2:10:24	100%

8개의 라우터, 5개의 레이어 4 스위치, 208개의 Subnet, 그리고 149개의 VLAN을 탐색하는데 100분이 소요되었다.

기존의 다른 연구[2]를 보면 243개의 라우터 및 스위치, 1363개의 노드, 72개의 subnet을 탐색하는데 42분이 소요되었다. 우리의 시스템으로 비슷한 장비를 탐색하는 데에는 약 60분 가량이 소요되었다. 그 이유는 장비별로 다양한 속성값을 보여주고, VLAN을 계산하고, 장비별 타입을 찾는것과 같은 다양한 기능들을 제공하기 위한 정보 처리 시간이 소요되기 때문이다.

이와 비슷하게 우리의 결과와 기존 연구[3]를 비교해보면, 기존의 연구에서는 최대 네트워크 크기가 단지 2000개의 host와 50개의 bridge에 불과하여 2000개의 노드에 대하여 연결 정보를 탐색하는데 25초의 시간이 소요되었으며, 50개의 bridge에 대해서는 25초의 시간이 소요된 데 반해, 우리의 시스템에서는 표 3에서 알 수 있듯이 598개의 라우터 및 스위치를 탐색하는데 11분이 채 걸리지 않았고, 7495개의 장비들간의 연결 정보를 찾는데 30초의 시간이 소요되었다. 라우터와 스위치 사이의 연결 정보를 찾는 시간이 조금 더 소요되는데 그 이유는 우리의 시스템에서는 인터페이스와 인터페이스간의 연결 정보 탐색을 하고, 이러한 정보를 구성도에 표시하고자 하였기 때문이다.

기존의 연구[4]에서는 각각의 네트워크에 대하여 총 탐색시간에 대해 명시하지 않았다. 대신에 노드 탐색하는데 걸린 시간을 그래프로 보여주었다. 이 알고리즘에 따르면 최초 25개의 장비를 탐색하는데 5초가 소요되었으며, 더 많은 장비를 탐색할수록 소요 시간은 차츰 증가하였다. 본 논문에서 제안하는 알고리즘은 장비의 수에 대해 일관된 성능을 보인다. 기존의 연구들은 작은 네트워크를 대상으로 하였지만 우리는 7495개의 장비를 탐색할 뿐만 아니라 각 노드와 링크에 대해 다양한 속성 정보들도 탐색하였다. 결과적으로 본 논문에서 제안하는 시스템이 기존의 연구[1, 2, 3, 4]에 비해서 더 나은 기능들을 제공하고, 더욱 안정적이라 할 수 있을 것이다.

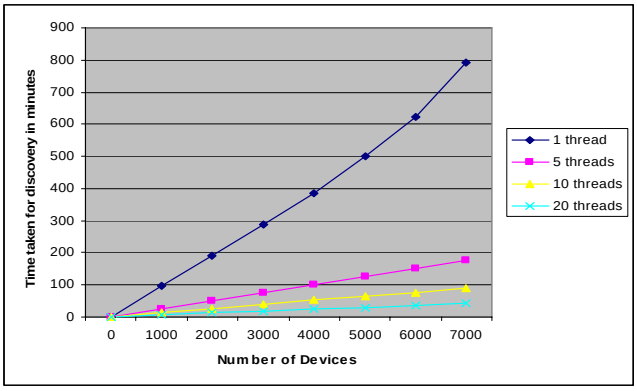

그림 6. Thread를 적용한 Postech 망 탐색

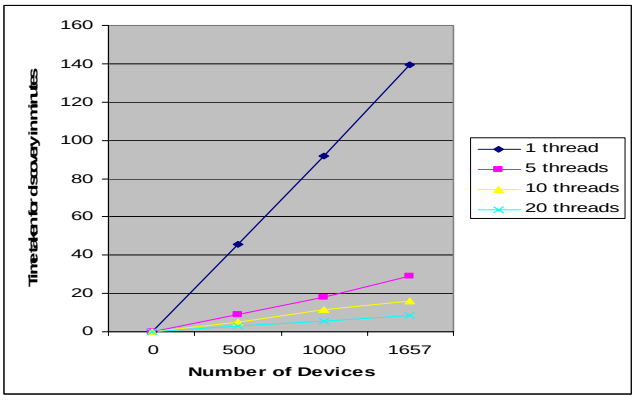

그림 7. Thread를 적용한 고려대 망 탐색

6. 결론 및 향후 과제

그동안 자동화된 네트워크 연결 정보 탐색에 대한 많은 연구가 행해져 왔음에도 불구하고, 기존의 연구들은 대부분 네트워크 뷰에 대한 논리적이거나 물리적인 계층의 추상화가 아닌 ping, spoofed icmp echo 요청 등을 이용하여 단순히 네트워크 연결 구성 정보를 탐색하는 것으로 여겨져 왔다. 더욱이 현재의 네트워크에서는 spoofed icmp 패킷은 여러 가지 보안의 이유로 block되어, ping이 네트워크내에서 지원되지 않는 노드들이 많이 있다. 본 논문에서는 L2/L3 계층의 구성 탐색 방법과 그것들을 어떻게 서로 혼합하여 연결 정보를 보여줄 수 방안에 대해서도 설명하였다. 본 논문에서는 라우터, L2/L3/L4/L7 스위치, 프린터, host 등 다양한 타입의 장비를 탐색하는 방법과 Subnet View, List View, Graph View, Tree View, VLAN View 등 다양한 연결 정보를 제공함으로써 사용자로 하여금 물리적/논리적인 뷰를 볼 수 있게 하였다. 또한 본 논문에서는 SNMP 기반의 네트워크 구성 탐색 시스템을 구현하였으며, 다양한 테스트를 통하여 이 시스템이 다수의 장비들을 탐색하는데 효율적임을 보였다.

향후 과제로, 더 많은 네트워크 관리 기능을 제공하기 위하여 network weather map이나 다른 네트워크 성능 monitoring tool과 우리의 시스템을 연계하는 연구가 이루어져야 한다. 또한 link capacity 나 mean delay 등 더 많은 연결 정보 특성을 고려하고, 네트워크의 증가 패턴을 찾는데 도움이 되는 시간의 흐름에 따른 네트워크 구성의 변화에 대한 다양한 분석이 필요할 것이다.

6. 참고 문헌

[1] R.Siamwalla, R. Sharma, and S. Keshav, “Discovering internet topology discovering Internet Topology,” http://www.cs.cornell.edu/skeshav/papers/discovery.pdf, July 1998.

[2] Yuri Breitbart, Minos Garofalakis, Member, IEEE, Ben Jai, Cliff Martin, Rajeev Rastogi, and Avi Silberschatz “Topology Discovery in Heterogeneous IP Networks: The NetInventory System,” IEEE/ACM Transactions on networking, vol. 12, no. 3, June 2004, pp. 401~414.

[3] B. Lowekamp, D. R. O’Hallaron, and T. R. Gross, “Topology discovey for large Ethernet networks,” in Proc. of ACM SIGCOMM, 2001, pp. 237~248.

[4] Fawad Nazir, Tallat Hussain Tarar, Faran Javed Chawla, Hiroki Suguri, Hafiz Farooq Ahmad, Arshad Ali, “Constella: A Complete IP Network Topology Discovery Solution,” In the Proc. of APNOMS, Oct. 2007, pp. 425-436.

[5] How To Get Dynamic CAM Entries (CAM Table) for Catalyst Switches Using SNMP, http://www.cisco.com/warp/public/477/SNMP/cam_snmp.html.

[6] JGraphT implementation and source code, http://jgrapht.sourceforge.net/.

[7] SNMP community String indexing, http://www.cisco.com/warp/public/477/SNMP/camsnmp40367.html.

[8] Bierman and K. Jones, “Physical topology MIB,” IETF, Internet RFC-2922, Sept. 2000.

[9] N. Duf_eld, F. L. Presti, V. Paxson, and D. Towsley, “Network loss tomography using striped unicast probes,” IEEE/ACM Transactions on Networking, vol. 14, no. 4, 2006, pp. 697~710.

[10] Advent Net API, http://snmp.adventnet.com/.

[11] K. McCloghrie, M. Rose, “RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II,” March 1991.

[12] E. Decker, P. Langille, A. Rijsinghani, K. McCloghrie, “RFC 1493-Bridge MIB,” July 1993.

[13] E. Bell, A. Smith, P. Langille, A. Rijhsinghani, K. McCloghrie, “RFC 2694 Q- BRIDGE-MIB,” August 1999.

[14] CISCO-VTP-MIB, http://tools.cisco.com/Support/SNMP/do/BrowseMIB.do?local=en&mibName=CISCO-VTP-MIB

[15] J. Case, M. Fedor,M. Schoffstall, and J. Davin, “A simple network management protocol (SNMP),” IETF, Internet RFC-1157, May 1990.

[16] D. Passmore, and J. Freeman, “The Virtual LAN Technology Report,” http://www.3com.com/nsc/200374.html, March 1997.

[17] IEEE 802.1Q, IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridge Local Area Networks, 1998.

[18] Benoit Donnet, Timur friedman, “Internt Topology Discovery: a survey,” IEEE Communications Surveys & Tutorials, 4th Quarter, vol. 9, no. 4, 2007, pp.56-69.