석 사 학 위 논 문

NETCONF기반 구성관리를 위한 성능향상 방법

유 선 미 (劉 善 美)

전자컴퓨터 공학부 (컴퓨터공학)

네트워크 전공

포항공과대학교 대학원

2006

NETCONF기반 구성관리를 위한 성능향상 방법

Performance Improvement Methods for NETCONF-based Configuration Management

Performance Improvement Methods for NETCONF-based Configuration Management

By

Sun-Mi Yoo

Division of Electrical and Computer Engineering

(Computer Science and Engineering)

POSTECH

A thesis submitted to the faculty of POSTECH in partial fulfillment of the requirements for the degree of Master of Science in the Division of Electrical and Computer Engineering (Computer Science and Engineering)

Pohang, Korea

December 21, 2005

Approved by

_____

Major Advisor: James Won-Ki Hong

NETCONF기반 구성관리를 위한 성능향상 방법

유 선 미

위 논문은 포항공과대학교 전자컴퓨터공학부 (컴퓨터공학) 석사 학위논문으로 학위논문 심사위원회를 통과하였음을 인정함.

2005년 12월 21일

학위논문심사 위원회 위원장 홍원기 (인)

위 원 서영주 (인)

위 원 송황준 (인)

MCC
20042010

유 선 미, Sun-Mi Yoo, Performance Improvement Methods for NETCONF-based Configuration Management, NETCONF기반 구성관리를 위한 성능향상 방법, Division of Electrical and Computer Engineering (Computer Science and Engineering), 2006, 57P, Advisor: J. Won-Ki Hong, Text in Korean.

[1]
[2]

ABSTRACT

Current networks are composed of many devices from diverse vedors in large-scale decentralized environment. So, configuration management in the center requires remote configuration management. IETF's NETCONF WG has taken efforts to standardize configuration management protocol. The standardization allows high interoperability of configuration management. However, configuration management requires interoperability as well as high performance. A lot of research has been done on functions of NETCONF but they have not yet considered about performance aspects of NETCONF.

In this thesis, we propose methods to improve performance by layers of NETCONF. We show the performance evaluations to verify efficiency of the proposed methods. Moreover we compare our results with others. This evaluation result can be used as a guideline for implementing NETCONF. Our methods improve performance of NETCONF and show process of evaluation performance of configuration management. Finally, we have applied the proposed methods to XCMS which is an XML-based configuration management system.

목 차

그림 목차

표 목차

1 서론

네트워크 구성관리는 네트워크를 구성하는 각 장치의 동작을 규정하는 운영 값을 설정하고 설정된 값을 수집하여 분석하는 작업이다. 예를 들면 라우터의 라우팅 파라미터를 설정하거나 침입탐지시스템이나 파이어월의 보안 설정 값을 설정하고 모니터링하는 작업이다. 구성관리는 지역적으로 분산되어 있는 네트워크 장치들을 중앙에서 관리를 해야하므로 원격 관리가 필수적이다.

원격에 있는 장치에 대한 구성관리를 위해서 IETF(Internet Engineering Task Force)[1]에서 NETCONF[2] 표준안을 제시하였다. NETCONF 표준은 현재 네트워크가 다수의 회사에서 제작한 다양한 종류의 장치가 서로 연결되어 있는 점을 고려하였다. 표준에 의하여 구성관리를 하게되면 구성관리를 효과적으로 수행할 수 있다.

구성관리는 표준에 의한 상호연동성 뿐만 아니라 구성관리에 대한 효율성도 중요하다. 구성관리는 다수의 장치를 대상으로 하기 때문에 효율성이 중요하다. 예를 들면 많은 수의 장치에 대한 설정값을 모니터링해야 하는 경우에 효율적으로 설정값을 모니터링하지 못하면 적절한 결과를 얻을 수 없다. 또한 구성관리에서 장치의 운영설정값을 변경하는 중간에는 네트워크가 불안정해 질 수 있기 때문에 신속한 구성관리 완료가 필요하다.

NETCONF 표준은 초기단계로서 기능적인측면이 충분히 논의되어 표준으로 만들어졌으나 효율적인측면에서의 기술적 검토가 부족한 상태이다. 현재까지 NETCONF 표준에 따르면서 효율적으로 구성관리를 실행할 수 있는 방법에 대한 학술논문이나 기술보고서가 없다. 즉 NETCONF 표준에 대하여 성능을 측정하여 분석한 결과도 없고 NETCONF 표준을 효율적으로 사용하기 위한 방안도 없다. 뿐만 아니라 NETCONF 표준 기반의 구성관리 시스템의 개발보고서나 학술논문

에서도 성능에 대하여 언급하고 있지 않다.

본 논문에서는 NETCONF 표준의 성능을 향상하여 구성관리의 효율성을 높일 수 있는 방안을 제시한다. NETCONF 표준에서는 OSI 나 인터넷 모델과 같이 구성관리 프로토콜을 기능별로 세분화하여 계층(Layer)별로 나누어 놓았다. 본 논문에서는 각 계층별로 성능을 향상할 수 있는 방안을 제시하고 성능을 측정하여 제시한 방법에 대한 효과를 검증한다. 제시하는 방안에는 표준을 준수하면서 성능을 향상할 수 있는 방법과 표준에는 제시되어 있지 않지만 성능을 향상할 수 있는 방법이 모두 포함된다.

성능에서 고려한 주요한 성능 변수는 응답시간(response time), 네트워크 사용량(Network Usage), 시스템 자원 소모량(Computing Resouce Usage)이다. 응답시간을 향상시켜 신속한 구성관리가 이루어질수 있게 하며 네트워크 사용량을 최소화하여 네트워크에 대한 영향을 줄일 수 있게 한다. 또한 시스템 자원 소모량을 줄여서 관리자 시스템이 많은 수의 장치를 관리할 수 있게하고 에이전트 시스템은 작은 네트워크 장치에도 적용할 수 있게한다.

또한, 본 논문에서는 제시한 성능향상 방법이 적용된 NETCONF 기반의 구성관리 시스템 XCMS 개발결과도 제시한다. 개발결과에는 매니저 시스템과 에이전트 시스템 각각에 대한 설계 결과, 구현결과가 포함되어 있다. 개발결과는 본 논문에 앞서 개발된 시스템의 성능을 개선한 것이다.

본 논문이 기여하는 점은 다음과 같다. 첫째 NETCONF 표준에 대하여 성능을 향상시킬 수 있는 방법을 제시하였다는 점이다. 이점은 본 논문의 핵심주제로 NETCONF 표준화에 반영되기를 기대하고 실제 NETCONF 시스템을 개발하는데 참고될 수 있다. 둘째로 구성관리에서 성능은 어떤 것을 고려해야하는지 제시하였다는 점이다.

본 논문의 구성은 다음과 같다. 서론에 이어 2 장에서는 관련 연구를 살펴보고, 3장에서는 NETCONF의 효율성에 관한 연구 결과에 대해서 설명한다. 4장에서는 본 논문에서 제시한 성능 향상 방법에 맞추어서 구현된 XCMS의 구조에 대해서 설명하고 5장에서 XCMS를 이용한 성능측정 결과를 제시한다. 마지막으로 6장에서 결론 및 향후연구에 대하여 기술한다.

2 관련 연구

이 장에서는 구성 관리의 선행 연구로서 Cisco[43]나 Juniper[42] Networks 같은 네트워크 장치 업체에서 개발된 XML[26] 기반 구성 관리 시스템에 대해서 알아보고, NETCONF 프로토콜에 대해서 소개한다. 다음으로, NETCONF 프로토콜을 구현한 구성 관리 시스템들에 대해서 살펴보며, XML 기반의 네트워크 관리의 성능에 대해 수행되었던 이전연구에 대해서 알아본다.

2.1 상용 구성관리 시스템

네트워크 관리에서 XML을 이용한 여러 시도들이 있어 왔으며, 구성 관리에서도 예외는 아니였다. 구성관리에서는 XML을 기반으로 한 RPC방식을 사용해 왔는데, 이는 장치들간의 데이터 교환을 위한 간단하고 호환성 있는 기술이었다. 하지만, 각 제조사들간에 사용하는 프로토콜들이 다양하고 동일하지 않기 때문에 다른 제조사의 장치들 간에는 서로 구성관리를 위한 통신이 가능하지 않았다. 다음의 두 시스템들은 XML을 사용한 대표적인 상용 구성관리 시스템들이다.

먼저, Juniper Networks' JUNOScript[9]는 JUNOS Network Operating System(NOS)에서 시스템의 조작을 위한 스크립트로써 처음 소개 되었으며, XML을 기반으로 한 네트워크 장비의 관리 노력과 비용을 줄일 수 있는 간단하고도 효율적인 모델을 제시한다. JUNOScript는 클라이언트의 XML-RPC를 통한 구성 정보로의 접근과 수행을 가능하게 한다. JUNOScript 서버는 태그의 분석을 통한 요청을 적절한 네트워크 장치에 배분하는 역할을 하며 수행 결과를 요청자에게 돌려준다.

다음으로, CISCO의 Configuration Registrar[8]는 CISCO IOS 네트워크 장비로 구성 파일을 자동으로 배포 시키기 위한 웹 기반 시스템이

다. 각 시스템에 위치한 시스코 네트워킹 서비스(CNS) 에이전트를 동작시킨다. 장비가 처음 수행을 시작 할 때 초기 구성 정보를 장비에 전송하고 이를 관리한다. 이는 HTTP[7]로 XML 메시지를 에이젼트와 통신하는 방식으로 동작된다.

위에 설명한 두 시스템을 비롯한 상용 구성 관리 시스템은 관리에 있어서 실질적인 편리함을 제공한다. 하지만 이와 같은 구성 관리 방법은 앞에서 설명된 바와 같이 제조사에 종속적인 데이터 구조와 통신 모델을 가진다. 이를 해결하기 위해 IETF에서는 시스코, 쥬니퍼 등의 주요 업체의 구성관리 시스템을 모델로 하여 NETCONF WG을 통해 구성 관리 모델의 표준화 작업을 하고 있다.

2.2 NETCONF의 개요

이 절에서는 현재 진행되고 있는 IETF[1]의 워킹그룹인 NETCONF[2]의 표준화 작업에 대해서 소개한다. 그리고 NETCONF의 관리 프로토콜[3] 메시지를 전송하기 위한 전송 프로토콜인 SSH[5], BEEP[6], SOAP/HTTP[4]에 대해 간략히 살펴본다.

2.2.1 NETCONF 프로토콜[3]

NETCONF는 매니저와 에이전트 간의 간소화된 RPC메커니즘 기반의 통신 기능을 제공한다. NETCONF 프로토콜은 매니저가 네트워크 장치들의 구성 정보를 쉽게 관리할 수 있도록, 그리고 에이전트간의 상호 운용성을 보장하기 위하여 에이전트와 매니저 사이에 전송되는 메시지를 구조화된 XML 형식으로 정의하고 있다. 이러한 구조화된 XML 메시지를 통하여 에이젼트에게 구성 정보를 가져오고, 수정하는 구성 관리 서비스를 RPC(Remote Procedure Call) 방법으로 제공하게 된다. 즉, 요청 메시지에 수행하고 싶은 오퍼레이션 이름을 XML 태그

안에 명시하여 보내면 이 메시지를 받은 쪽에서는 자신의 XML 파서를 통하여 오퍼레이션 이름을 추출하고 그 오퍼레이션을 실행시킨 후 수행 결과를 응답 메시지에 담아서 보내게 된다. 이때 NETCONF 세션을 통한 애플리케이션과 장치간의 논리적인 연결을 관리하기 때문에 하나 이상의 애플리케이션이 해당 네트워크 장치를 관리하는 경우 각 세션에 대한 관리 정보와 글로벌 구성 정보의 관리에 대한 방법도 제공한다.

NETCONF는 한 개 이상의 구성 정보 저장소들(configuration datastores)을 가진다. 구성 정보 저장소로는 <running>, <startup>, <candidate>등이 존재한다. 먼저, <running>은 현재 장비에 수행되고 있는 구성 정보를 나타낸다. <candidate>는 <running>에 직접 수정 작업을 수행 시, 잘못된 수정에 따른 장치의 잘못된 동작을 미연에 방지하기 위한 구성 정보 데이터이다. <candidate>에 수정 작업을 수행 후 <candidate>구성 정보 내용의 안전성이 검증되면 'commit' 오퍼레이션을 통해서 <running>에 반영된다. 즉, 장비의 구성 정보에 반영이 된다. 마지막으로, <startup>은 장비의 부팅 등의 재 수행 시 구성 정보로 사용되는 데이터이다. 그림 1은 구성 모델들간의 관계를 나타낸다.

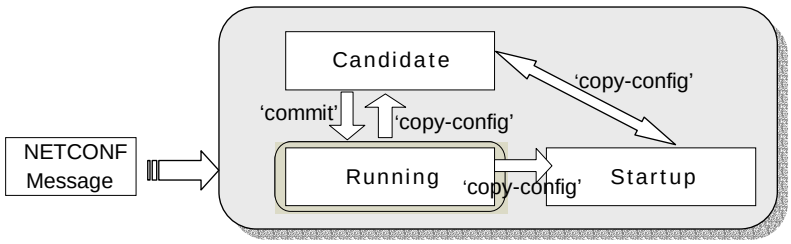


그림 1. 구성 모델(Configuration Model)

구성 관리를 수행하고자 하는 네트워크 장치의 전송 프로토콜, 구성 관리 데이터, 구성 관리 오퍼레이션 등의 환경은 매우 다양하다. 따라서, 하나의 전송 프로토콜, 구성 관리 데이터, 구성 관리 오퍼레이션 모델의 정의만으로는 예상되는 다양한 환경의 네트워크 장치에

대한 구성 관리의 요구를 명시하기가 쉽지 않다. 따라서 전송되는 메시지를 4가지 계층으로 나누어 정의함으로써 다양한 환경의 구성 관리 요구 사항을 충족시킬 수 있게 된다.

계층	내용
Application Protocol Layer	메시지들을 전달하는 전송 프로토콜의 사용방법 : BEEP, SSH, HTTP etc.
RPC Layer	RPC 오퍼레이션에 대한 요청과 응답을 나타냄 : <rpc>, <rpc-reply>, <rpc-error> etc.
Operations Layer	NETCONF 에서 정의한 RPC 를 제공할 오퍼레이션들을 나타냄 : <get-config>,<edit-config>,<copy-config>, etc.
Content Layer	XML 형태의 관리 구성 정보를 나타냄 Configuration data

표 1. NETCONF 의 계층적 구조

표 1은 NETCONF에서 정의하고 있는 4가지 계층과 해당 계층에 포함되는 내용들을 예로 보여주고 있다.

Application Protocol Layer

NETCONF는 계층으로 나누어 정의하여, 전송 프로토콜에 대해서는 중립적이다. 따라서 어떠한 전송 프로토콜도 지원하는 것이 기본 개념이다. 하지만 현재는 지원 프로토콜로써 SSH, BEEP 그리고 SOAP over HTTP에 대해서만 고려한다. Application 프로토콜 계층(Application Protocol Layer)은 매니저와 에이전트의 연결(connection)에 대해 정의한다.

매니저와 에이전트가 하나의 세션을 형성하면 NETCONF에서는 다중의 전송프로토콜 연결이 가능하다. NETCONF에서 이런 전송프로토콜 연결을 세션(session)이라 부르고 있다. 새 인터넷 초안에서는 예전의 세 가지 종류의 채널 중 두가지 채널(Management, Notification channels)의 정의가 프로토콜의 간소화를 위해 사라졌다.

그림 2은 ‘edit-config’ 오퍼레이션의 요청을 표현하는 NETCONF 프로토콜 메시지의 예로써 NETCONF에서 정의하고 있는 4가지 계층 중 Application 프로토콜 계층을 제외한 나머지 세 개의 계층과 메시지와의 관계를 보여 준다. 이 메시지는 실제 동작하는 네트워크 장치의 구성 정보인 <running>의 지정된 인터페이스 카드와 NETMASK의 구성 정보 값을 수정하게 된다.

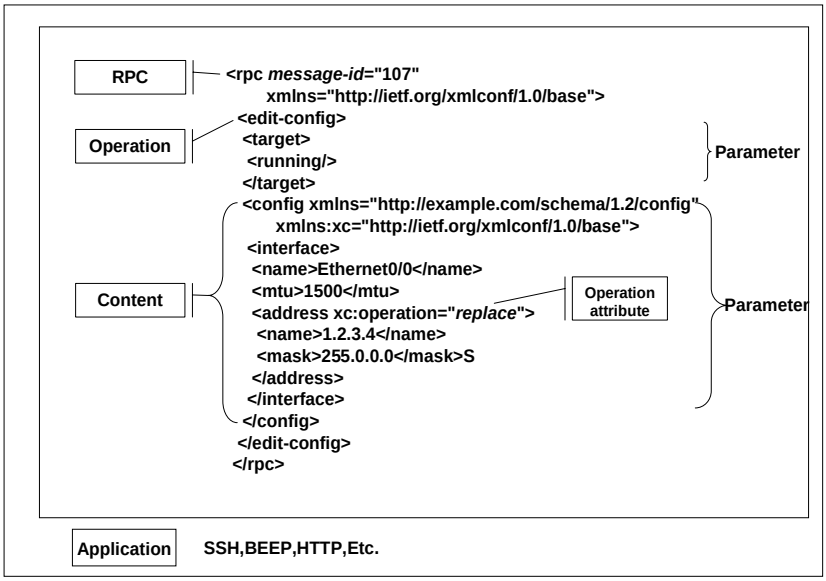


그림 2. ‘edit-config’ 오퍼레이션을 수행하는 NETCONF 프로토콜 메시지

RPC Layer

NETCONF 프로토콜은 RPC기반의 통신 모델을 사용한다. Application 프로토콜에 종속되지 않는 요청과 응답의 구조를 제공하기 위해 <rpc>, <rpc-reply>를 사용한다. 또한 매니저가 요청한 메시지에 대해 에이전트가 제어할 수 있는 방법을 <rpc> 태그 안에 요청 메시지로 명시한다. 이는 ‘message-id’의 정의를 통해 제공하게 되는데, 메시지 번호를 사용하여 이전에 요청한 메시지를 제어 할 수 있다.

즉, ‘message-id’는 매니저와 에이전트의 접속에 의해 세션이 이루어 진 후, 해당 세션을 통해 주고 받는 메시지를 관리 하기 위한 번호라고 할 수 있다. 그림 2에서의 RPC태그가 RPC 레이어를 보여준다. RPC 태그의 속성인 ‘message-id’ 값이 107인 것은, 해당 메시지가 매니저와 에이전트 사이의 세션에서 107번째의 NETCONF 프로토콜 메시지라는 의미이다. 또 RPC 레이어에는 오퍼레이션인 ‘edit-config’가 하위 항목임을 볼 수 있다. 이는 RPC를 통해 수행되는 오퍼레이션이 ‘edit-config’ 오퍼레이션임을 표시하는 것이다.

Operations Layer

NETCONF는 SNMP에서 제공하고 있는 ‘get’, ‘set’, ‘trap’에 대응할 수 있는 ‘get-config’, ‘edit-config’, ‘notify’ 오퍼레이션 뿐만 아니라, 트랜젝션(transaction) 처리 즉, 관리 오퍼레이션 수행 중에 에러가 나면 수행 전의 상태로 돌아가는 ‘rollback’, 또는 에러가 난 곳에서 멈추고 에러 메시지를 보내는 ‘stop-on-error’ 등이 정의된다. 또한 구성 정보의 일관성을 유지하기 위해서 여러 매니저가 동시에 한 장치의 구성 정보를 수정하고자 할 때, 수정되고 있는 구성 정보에 대해서 잠금(lock)을 걸 수 있는 오퍼레이션이 정의된다. 이는 SNMP에서 제공하지 않는 관리 오퍼레이션들이 새롭게 정의 되었음을 알 수 있다.

아래에는 ‘get’, ‘edit’, ‘copy’, ‘delete’에 해당하는 NETCONF 프로토콜의 기본 오퍼레이션을 보여준다.

<get-config>: 에이전트가 관리하고 있는 구성정보의 전체 혹은 특정 부분을 불러 올 때 사용하는 오퍼레이션이다. 즉 에이젼트가 가지고 있는 구성 정보를 매니저에게 전송하고자 할 때 호출 된다.

<edit-config>: 실제 구성 관리의 실질적인 일을 하는 오퍼레이션이다. 에이전트가 갖고 있는 구성 정보를 변경할 때 수행되는 세 가지 오퍼레이션을 가진다. merge는 새로운 구성 정보를 추가할 때,

replace는 기존 구성 정보를 변경 할 때, delete는 기존 구성 정보를 삭제 할 때 호출 된다.

<copy-config> : 매니저가 에이전트에게 구성 정보를 원격으로 로딩하거나 에이전트의 <running>을 <startup>으로 아니면 그 반대의 경우를 수행할 수 있는 오퍼레이션이다. ‘edit-config’가 구성 관리 정보의 일부분을 조작한다면 이 오퍼레이션은 전체의 백업 및 리스토어가 가능하게 한다.

<delete-config> : <running> 이나 <startup>, <candidate>의 삭제를 하는 오퍼레이션이다.

구성 정보 수정과 관계된 오퍼레이션들 ‘edit-config’, ‘copy-config’ 그리고 ‘delete-config’는 모두 lock 오퍼레이션과 연관 관계가 있다. 해당 세 오퍼레이션은 대상 구성 정보가 unlock 된 상태에서만 수행이 가능하다.

그림 2 메시지에서 NETCONF 오퍼레이션의 실제 사용을 보여준다. ‘edit-config’ 오퍼레이션에 필요한 파라미터인 target과 config정보가 메시지에 포함되어 있으며 target의 <running>은 실제 수정될 구성 정보 저장소이며 config에는 교체될 구성정보가 포함된다.

Content Layer

현재 관리 구성 정보인 content는 관리되어야 할 네트워크 장치마다 그 내용이 다르게 정의되기 때문에, 구성 정보 내용은 네트워크 장치 업체에 의존적이게 된다. 따라서 구성 정보를 정의하는 언어와 그 내용에 대한 표준화 작업은 별도로 이루어져야 한다. NETCONF는 content 계층 아래 부분에 대한 표준화 작업을 NETMOD를 통해 진행하고 있다. 그림 2에 content 계층의 실제 구조를 볼 수 있다. config 태그의 아래 부분이 content 계층이며, 포함된 내용이 교체 정보임을 표시하는 replace 값을 가진 operation attribute와 실제 교체 정보가 포

함된다. 이때 name와 mtu의 값으로 어떤 interface의 값이 교체될 것인지를 나타낸다.

2.2.2 NETCONF의 전송 프로토콜

NETCONF 프로토콜 메시지는 어떠한 전송 프로토콜에서도 전송되어야 한다. 그러나 현재 NETCONF에서는 전송 프로토콜로 SSH, BEEP, SOAP/HTTP에 대해서만 고려한다. NETCONF는 각각의 전송 프로토콜에 대해 별도의 인터넷 초안을 제안하고 있다. 이 장에서는 각 전송 프로토콜에 대해서 간략하게 살펴보며, 각 전송 프로토콜의 장단점을 요약 정리하고자 한다.

SSH[5](Secure Shell)는 기존의 네트워크 장비의 관리에 있어서 주로 사용해 오던 환경이다. 따라서 기본적으로 네트워크 오퍼레이터들의 강력한 요구에 의해 제공되어야 하는 전송 프로토콜이다. 그러나 NETCONF에서의 SSH는 구현의 간편성을 위하여 관리 수행을 위한 하나의 TCP연결만을 제안하고 있다. SSH의 경우 에이전트가 매니저에게 메시지를 비 동기적으로 보낼 수 있는 방법이 제공되지 않기 때문에 보고 채널 같은 경우는 유지할 수도 없다.

BEEP[6](Block Extensible Exchange Protocol)은 현재 Cisco와 Juniper Networks와 같은 대표적인 네트워크 장치 업체에서 추천하고 있는 전송 프로토콜이다. BEEP는 SSH와 달리 NETCONF에서 정의한 다중 채널을 지원하도록 제안하고 있다. BEEP은 서버/클라이언트의 구조가 아닌 peer의 구조를 가지기 때문에 매니저가 에이전트에게 또는 에이전트가 매니저에게 관리 정보를 요청할 수 있다. 따라서 에이전트로부터 매니저에게 전송되는 비 동기적 메시지를 쉽게 처리할 수 있는 장점이 있다.

SOAP over HTTP[4]에서의 SOAP은 NETCONF의 의도에 가장 적

합한 프로토콜이라고 할 수 있다. 왜냐하면 SOAP은 자체적으로 RPC 메카니즘을 제공하고 있으며, SOAP RPC 메시지는 다양한 전송 프로토콜을 지원할 수 있기 때문이다. NETCONF에서 전송 프로토콜을 SOAP 하나로 제한하지 못하는 것은 현재 SOAP을 제공하고 있는 구현 환경인 HTTP는 서버/클라이언트 구조를 따른다. 이는 서버 역할을 하고 있는 에이전트가 매니저에게 메시지를 전송하는데 문제가 발생한다. 이러한 문제는 에이전트가 매니저에게 비 동기적으로 보내는 메시지 처리에 제약을 준다. 이를 해결하기 위한 방법으로 매니저에서 주기적으로 에이전트에게 폴링하여 통보 메시지를 가져오는 방법이 있지만 매번 주기적으로 폴링을 해야 하기 때문에 불필요한 트래픽과 매니저의 수행 오버헤드를 발생 시킨다.

	SSH	**BEEP**	**SOAP over HTTP**
Operation Channel	support	Support	support
RPC Interface	Not	Not	Support
Independent-transport	Not	Not	Support
Note	Server/ Client	peer-to-peer	Server/Client

표 2. NETCONF 전송 프로토콜 비교

2.3 NETCONF 시스템

본 절에서는 NETCONF 프로토콜의 표준안에 따라 현재 구현된 대표적인 구성 관리 시스템들에 대해서 간략하게 살펴보겠다. 해당 시스템들은 본 논문의 이전 연구들인 XCMS[13][29]와 XCMS-WS[30], 그리고 오픈 소소 프로젝트인 EnSuite[31][33]이다. 이들의 차이점을 비교해 보고 각 시스템에서의 개선점을 알아 본다.

2.3.1 XCMS

이전 연구로 개발된 XCMS(XML Based Configuration Management

System)는 POSTECH, DPNM 연구실에서 진행한 NETCONF 프로토콜에 대한 연구이다. 표준 NETCONF 프로토콜의 개선점을 찾아 수정하고 구성 관리의 효율성을 높이고자 하였다.

특징은 개발 당시에는 표준으로 제시되지 않았던 XPath을 사용하여 구성 정보를 포인팅한다. NETCONF 프로토콜의 RPC 계층을 SOAP메시지에 통합하고, XPath모드를 통해 관리 정보를 전체가 아닌 일부만 가지고 조작이 가능하게 하여 NETCONF 프로토콜 메시지의 효율성을 높였다. 또한, 개발 당시에 NETCONF 전송 프로토콜인 NETCONF over SOAP[40], SOAP over HTTP에 대한 실제 구현의 예가 존재하지 않았는데, 해당 NETCONF 프로토콜을 사용한 예를 보여주었다.

하지만 최신 표준의 지원이 되지 않는 점과 표준에 부합되지 않은 오퍼레이션의 설계, RPC 레이어의 SOAP메시지에 통합에 의해 관리 정보와 통신 정보의 분리가 되지 않는 점은 개선이 필요하다.

2.3.2 XCMS-WS

XCMS-WS(XML Based Configuration Management System using Web Services)는 NETCONF에 웹 서비스의 레지스트리 기능을 접목시켜 개발된 구성관리 시스템이다. XCMS-WS 역시 본 연구의 이전연구이며, POSTECH, DPNM 연구실에서 진행 되었다.

NETCONF의 표준화로 구성 관리 시스템이 여러 네트워크 장치들에 대해서 효율적으로 구성 관리를 수행 할 수 있게 되었다. 하지만, 수많은 장치들로 구성된 네트워크 환경에서 구성 관리를 좀 더 쉽게, 또 많은 장치들에 대해서 수행하는데에는 NETCONF 프로토콜만을 사용해서는 한계가 있다. 이에, XCMS-WS는 웹 서비스 기술 중 UDDI[41] 레지스트리를 사용하여서 네트워크 장치들을 좀 더 쉽게,

또 더 많은 장치들을 관리 하는 방법을 제시하였다.

XCMS-WS는 해당 시스템 만이 사용하는 Private UDDI를 구현하여 사용하였다. Private UDDI를 사용함으로써, 해당 UDDI 레지스트리는 XCMS-WS 만이 사용할 수 있으므로 보완성을 높였다. 또한, UDDI specification을 새로 정의 함으로써, 좀 더 세분화되게 네트워크 장치를 분류할 수가 있다. 하지만, XCMS-WS는 NETCONF의 기능에 웹 서비스의 기능을 접목시키는데 초점을 맞춘 시스템이다. 따라서, XCMS-WS 역시 성능 측면을 고려 하지 않았을 뿐만 아니라, NETCONF 프로토콜의 모든 기능들을 제공하지 않는다. 이전 연구인 XCMS가 제공하던 기능들을 구현 당시의 표준화에 맞춰서 업그레이드를 한 시스템에 웹 서비스 기술을 접목 시켰다. 또한, 전송 프로토콜은 SOAP over HTTP만을 사용하고 있다.

2.3.3 EnSuite(Extended NETCONF SUITE)

EnSuite(Extended NETCONF SUITE)은 프랑스의 Loria 연구소에서 진행중인 Open Source Project이다. Loria 연구소에서는 예전에 Yenca[32]라는 NETCONF agent를 개발하여 발표하였었다. Yenca는 C언어로 구현되어 있었으며, 기능과 구성면에서 많은 부분이 부족했었다. 또한, NETCONF의 전송 프로토콜들인 SSH, SOAP, BEEP중 어느 것도 지원하지 않았었다. Loria에서는 기존의 Yenca[32]의 단점들을 보완한 새로운 구성 관리 시스템 EnSuite을 발표하였다.

EnSuite은 NETCONF 웹 기반의 매니저, NETCONF 에이전트 그리고 확장할 수 있는 모듈들로 구성된다. 또한, 모든 구성 요소들은 Python으로 구현 되었다. 먼저, Yenca-Man은 매니저 어플리케이션이며, 웹을 기반으로 하는 GUI이다. Yencap는 NETCONF agent 구현물이다. NETCONF의 여러 기능들을 제공하고 있으며, SSH를 전송 프로토콜로

사용하고 있다. 또한, Yencap는 구성 정보들을 모듈 단위로 나누어서 관리를 한다. 예를 들어, 구성 정보를 이루는 단위로 router, interface, IPSec등 여러가지가 있다. 이러한 단위들은 구성 정보 설정 방법이 다르고, 구성 정보 내용도 다르다. Yencap는 기본적으로 router, interface, system module들을 간단하게 제공하고 있으며, 다른 모듈들이 추가 되는데 있어서 편리함을 제공하는 것을 장점으로 내세우고 있다.

2.4 Network Management Performance

본 절에서는 네트워크 관리 분야에서 수행 되었던, 성능에 관련된 연구들에 대해서 간략하게 살펴보겠다. 먼저, 네덜란드의 Twente 대학에서 수행 되었던 웹 서비스와 SNMP의 성능을 비교한 연구[28]를 살펴본 후, 구성 관리 분야에서 수행 되었던 성능 연구[27]를 알아보겠다.

2.4.1 SNMP와 웹 서비스 기반 네트워크 관리 성능 비교 연구

네덜란드에 있는 Twente 대학교의 Aiko Pras와 그의 연구원들은 웹 서비스에 관련된 여러 연구를 해왔다. 여러 연구들 중에서, 그들은 SNMP와 웹 서비스를 이용한 네트워크 관리의 성능을 비교 분석하는 연구도 수행하였다. 해당 연구는 성능 비교를 위해서, 대역폭 사용량, CPU 시간, 메모리 사용량 그리고 수행시간을 측정하였다. 또한, 그들은 성능 테스트에 사용하기 위해 웹 서비스를 기반으로 하는 네트워크 관리 시스템을 프로토타입 형태로 구현하였다. 구현한 시스템과 SNMP를 사용하는 관리 시스템을 이용하여 비교하는 테스트를 수행하였다. 해당 연구의 결과는 SNMP와 웹 서비스를 이용하여 네트워크 관리시 대역폭 부분에서 확연한 차이를 보여준다. 또한, 그들은 SNMP가 아주 적은 수의 객체의 값을 읽는 경우에는 웹 서비스보다

더 좋은 성능을 보여준다고 결론을 내렸다. 하지만, 많은 수의 객체들의 값을 읽을 때는 웹 서비스가 SNMP보다 두드러지게 좋은 성능을 보여준다는 결론을 내렸다. 그들은 해당 연구를 통해서 웹 서비스를 이용한 네트워크 관리 방법은 큰 규모의 네트워크에 훨씬 더 적절하다고 제시하였다.

2.4.2 XML 기반 구성 관리 성능 연구

XML 기술은 IETF, UPnP 그리고 DSL 포럼들의 표준화 및 프로토콜로 많이 사용되어 왔다. 해당 XML 기술은 XML 데이터를 사용하므로써 긴 길이의 데이터를 지니며, 전송 프로토콜로는 SOAP RPC를 많이 사용한다. 많은 장비들을 관리해야하는 구성 관리 환경에서는 패킷의 크기가 문제가 될 수 있다. 해당 연구에서는, 패킷의 크기를 줄이기 위해 해결책으로 Lempel-Ziv[44] 압축 알고리즘 사용을 제시하였다. Lempel-Ziv 압축 알고리즘을 사용함으로써 얻는 효과를 실험을 통해서 증명하였다. 실험은 홈 네트워크 장비들을 관리하는 환경에서 수행되었다. 압축 알고리즘의 사용으로 줄어든 패킷의 크기가 줄어드는 것을 비교 실험으로 증명하였으며, 수행 시간을 측정하였다. 수행 시간은 네트워크의 속도를 변화시키면서 측정하였다. 해당 연구에서는 XML기술을 사용하는 구성 관리에 압축방식은 큰 효과를 제공한다고 결론을 내렸다. 또한, 압축과정이 수행시간에 많은 지연을 줄 수 있을 것 같지만 실제로는 큰 영향을 주지 않으며, 수행시간은 네트워크의 속도에 오히려 많은 영향을 받는다고 하였다.

3 NETCONF의 효율적 사용 방안

이번 장에서는 NETCONF 계층별로 성능을 측정하고 성능을 향상 시킬 수 있는 방법을 제시한다. NETCONF의 각 계층별로 성능을 측정하여 효율성을 검증하고 성능을 향상 시킬 수 있는 방안들을 제시한다.

3.1 계층별 분석 및 개선사항 제시

이번 절에서는 NETCONF의 각 계층별로의 성능을 측정하고, 성능 향상 및 좀 더 효율적으로 사용할 수 있는 방안을 제시한다.

3.1.1 Application Protocol Layer

NETCONF에서의 application 프로토콜 계층은 NETCONF 매니저와 에이전트 사이의 통신 방법을 나타낸다. NETCONF는 다수의 세션(session)들과 전송 프로토콜들을 제공한다. NETCONF는 전송 프로토콜로 SOAP over HTTP, SSH 그리고 BEEP을 제공하고 있다. 전송 프로토콜들에 대해서는 2장 관련 연구에서 자세히 설명하였다. 이번 절에서는 application 프로토콜 계층에서 사용되는 전송 프로토콜들의 성능을 측정하여 전송 프로토콜 사용지침을 제공한다. 또한, application 프로토콜 계층에서 성능을 향상 시킬 수 있는 방안을 제시한다. 전송 프로토콜들의 성능을 측정하기 위해서 다양한 크기의 구성 정보를 읽는 작업을 수행하였다. NETCONF 매니저는 해당 작업을 수행하기 위해서, ‘get-config’ 명령어에 subtree filtering 오퍼레이션을 사용한 메시지를 생성하여 전송하였다. 그림 3은 실험에 사용한 전송 메시지와 그에 대한 응답 메시지의 예제이다. 해당 연구는 읽어 오는 정보의 양을 변화시키기 위해서 interface의 구성 정보 중 iface의 개수를 2개씩 증가

시켜 수행하였다. 그림 4은 세 전송 프로토콜에 대해 수행한 실험 결과로 응답 시간의 변화를 나타낸다.

Request Message	Response Message
<?xml version='1.0'?> <rpc message-id='1'> <get-config> <source> <running/> </source> <filter type="subtree"> <interfaces/> </filter> </get-config> </rpc>	<?xml version="1.0"?> <rpc-reply message-id="1"> <data> <interfaces> <iface> <name>eth0</name> <address>141.223.82.1</address> <netmask>255.255.255.0</netmask> <bcast>255.255.255.255</bcast> <mtu>1500</mtu> </iface> <iface> <name>eth1</name> <address type="ipv4">141.223.82.3</address> <netmask>255.255.255.0</netmask> <bcast>255.255.255.255</bcast> <mtu>1000</mtu> </iface> </interfaces> </data> </rpc>

그림 3. NETCONF 메시지 예제

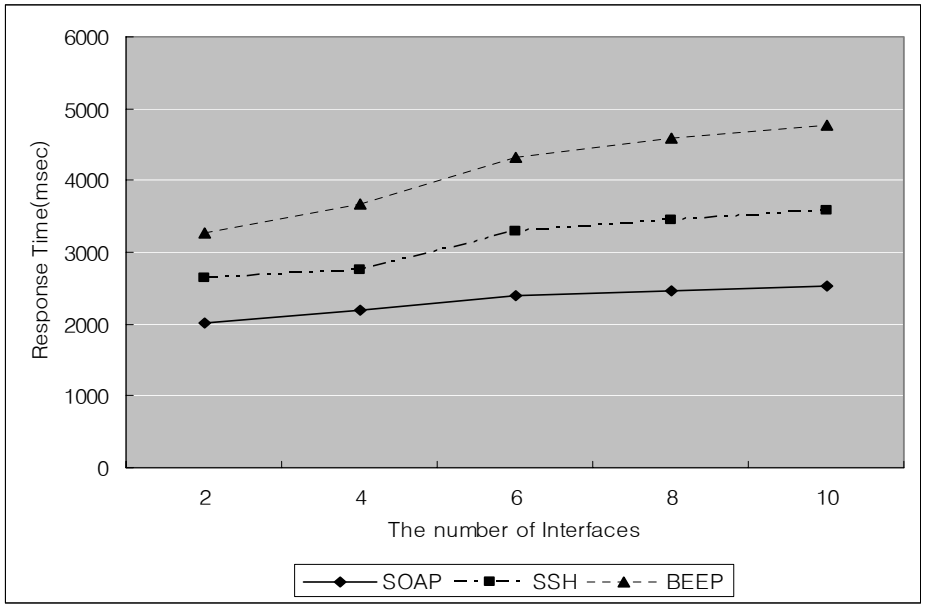


그림 4. 전송 프로토콜별 Response Time

그림 4의 그래프에 따르면, 예상외로 SOAP 프로토콜이 가장 적은 응답 시간이 소요되는 것을 확인할 수가 있다. 그림 5의 그래프에서 보여주는 것처럼 SOAP을 통해 전송되는 메시지는 크기가 커진다. 따라서, 보통 SOAP이 많은 응답 시간을 요구할 것이라고 예상을 할 것이다. 하지만, 각 전송 프로토콜은 통신방식이 다르다. SOAP 프로토콜은 WSDL[34]을 이용한 RPC 방식으로 수행하려는 오퍼레이션을 바로 호출하는 방식을 사용한다. 반면에, SSH는 포트를 포워딩 하는 방식 및 인증 방식을 수행하면서 중간 수행 시간에서 오버헤드가 발생한다. BEEP은 peer-to-peer 방식으로 통신이 된다. 클라이언트도 메시지를 전송할 수 있으며, 비 동기식 메시지 통신이 가능하다. 따라서, 매니저와 에이전트가 서로의 메시지 순서를 확인하는 메시지 전송이 필요하면서 메시지 전송 횟수는 많아지고 응답시간도 길어진다. 따라서, 메시지의 크기가 커질수록 응답시간이 다른 프로토콜들에 비해서 많이 길어지는 형상을 보여준다. 따라서, 전송 프로토콜의 응답시간은 그림 4과 같은 결과를 나타낸다.

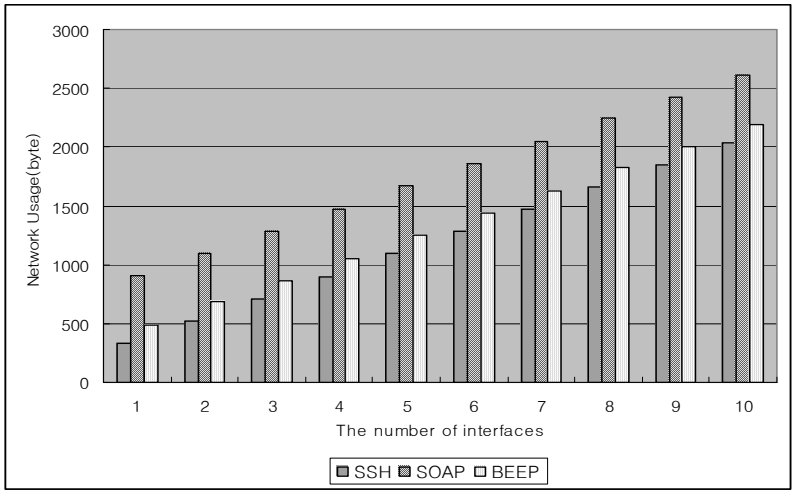


그림 5. 전송 프로토콜별 Network Usage

반면에, 네트워크 사용량에서는 그림 5과 같이 다른 결과를 보

여준다. 해당 실험은 다양한 크기의 데이터 전송시 네트워크 사용량을 측정하기 위해서 interface 구성 정보의 개수를 변화시켰다. 또한 NETCONF 에이전트에서 NETCONF 매니저로 메시지를 전송하는 부분만을 측정하였다. SSH를 제외한 두 프로토콜은 XML 기술들로, 부가적으로 꼭 붙어야 하는 내용이 많다. 특히, SOAP over HTTP는 크기가 작은 데이터를 전송 시 오히려 부가적으로 붙는 데이터가 전송하려는 데이터의 크기보다 더 클 수 있다. 그림 6은 각 프로토콜의 전송되는 메시지의 예제 파일이다.

(a) SSH-Based Message Example

Request Message	Response Message
<?xml version='1.0'?><rpc message-id='1'><get-config><source><running/></source><filter type="subtree"><interfaces/></filter></get-config></rpc>]]>]]>	<?xml version="1.0"?><rpc-reply message-id="1"><data><interfaces><iface><name>eth0</name><address>141.223.82.1</address><netmask>255.255.255.0</netmask><bcast>255.255.255.255</bcast><mtu>1500</mtu></iface></interfaces></data></rpc>]]>]]>

(b) SOAP over HTTP-Based Message Example

<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="urn:Netconfd"><SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><ns:netconfd><req-msg><?xml version='1.0'?><rpc message-id='1'><get-config><source><running/></source><filter type="subtree"><interfaces/></filter></get-config></rpc></req-msg></ns:netconfd></SOAP-ENV:Body></SOAP-ENV:Envelope>	<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns="urn:Netconfd"><SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><ns:netconfdResponse><rpl-msg><?xml version="1.0"?><rpc-reply message-id="1"><data><interfaces><iface><name>eth0</name><address>141.223.82.1</address><netmask>255.255.255.0</netmask><bcast>255.255.255.255</bcast><mtu>1500</mtu></iface></interfaces></data></rpc></rpl-msg></ns:netconfdResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>

(c) BEEP-Based Message Example

<?xml version="1.0"?><methodCall><methodName>netconfd</methodName><params><param><value><string><?xml version='1.0'?><rpc message-id='1'><get-config><source><running/></source><filter type="subtree"><interfaces/></filter></get-config></rpc></string></value></param></params></methodCall>	<?xml version="1.0"?><methodResponse><params><param><value><string><?xml version="1.0"?><rpc-reply message-id="1"><data><interfaces><iface><name>eth0</name><address>141.223.82.1</address><netmask>255.255.255.0</netmask><bcast>255.255.255.255</bcast><mtu>1500</mtu></iface></interfaces></data></rpc></string></value></param></params></methodResponse>

그림 6. 전송 메시지 예제

구성 관리 수행에 사용되는 전송 메시지들이 네트워크 트래픽에 많은 영향을 준다면 효율적인 구성 관리라고 할 수가 없다. XML을 기반으로 동작하는 NETCONF의 application 프로토콜 계층에서는 트래픽의 양을 줄일 수 있는 방안 제시가 필요하다. XML 메시지의 크기를 줄일 수 있는 방법으로는 binary XML[35], 압축 방법등 여러가지가 존

재한다. 본 논문에서는 2장 관련연구에서 살펴보았던 구성관리 성능에 관한 연구 결과[27]와 비교하기 위해서 NETCONF가 메시지를 전송할 때의 트래픽양을 줄이는 방법으로 압축 방법을 사용하였다. XML 메시지들의 특징은 데이터 값의 앞 뒤로 붙는 긴 태그들과 반복적으로 사용되는 단어들이다. 이 특징을 이용해서 XML 메시지를 압축하는 기술 중 하나인 Zlib[36]을 이용하여, 각 프로토콜의 payload 부분에 해당하는 값들에 압축 작업을 수행 후 전송한다. 수행 결과는 5장에서 비교 설명하도록 하겠다.

3.1.2 RPC Layer

NETCONF의 RPC 계층은 NETCONF 매니저로부터의 구성 관리 수행 요청 메시지와 NETCONF 에이전트로부터의 수행결과인 응답 메시지임을 나타내는 역할을 수행한다. RPC 계층에서는 <rpc> 요소로 수행 오퍼레이션을 감싸서 NETCONF의 요청 메시지임을 나타내고, <rpc-reply> 요소로 결과 값을 감싸서 NETCONF의 응답 메시지임을 나타낸다. 현재 NETCONF에서 제시하고 있는 <rpc> 요소 아래에는 단 한 개의 오퍼레이션만이 들어간다. 즉, NETCONF 매니저가 한 개의 오퍼레이션을 지닌 rpc 메시지를 생성하여 전송하고, NETCONF 에이전트는 전송 받은 메시지의 오퍼레이션을 수행한 후에, 결과 값인 rpc-reply 메시지를 생성하여 전송한다. 매니저는 그 뒤에 새로운 오퍼레이션을 수행할 메시지를 생성한 후 전송한다. 즉, 매니저는 에이전트의 수행 결과가 완료되어 응답 메시지가 올 때까지 그 뒤의 작업을 수행 못하고 기다려야 한다. NETCONF에서는 최근에 해당 방식을 RPC 계층에서의 비 효율성으로 지적하고, ‘pipelining’ 이라는 새로운 방법을 제시 하였다. 하지만, 현재까지 NETCONF에서는 단지 ‘pipelining방법을 사용할 수 있다’라는 내용만을 제시할 뿐, 해당 방법에 대한 지침서 및 성능, 해당 방법을 사용함으로써 얻을 수 있는 효

과에 대한 결과들을 제시하고 있지 않다. 이번 절에서는 RPC 계층에서의 효율성을 검증하고, 좀 더 효율적으로 수행 할 수 있는 방안들을 제시한다.

NETCONF에서 제시하고 있는 'pipelining' 방식은 기존의 'pipelining'방식과 유사하다. NETCONF 에이전트가 오퍼레이션 수행을 하는 중에, NETCONF 매니저로부터 메시지를 전송 받을 수 있다. 하지만, NETCONF의 특성 상, 오퍼레이션은 들어오는 순서대로 수행되어야 한다. 즉, NETCONF 에이전트는 이전 오퍼레이션이 완료되면, 해당 오퍼레이션이 완료되기 전에 전송 받은 메시지들 중 가장 먼저 받은 메시지를 처리한다. 그림 7은 'pipelining'을 사용하는 통신 방식과 사용하지 않는 경우의 통신 방식을 보여준다.

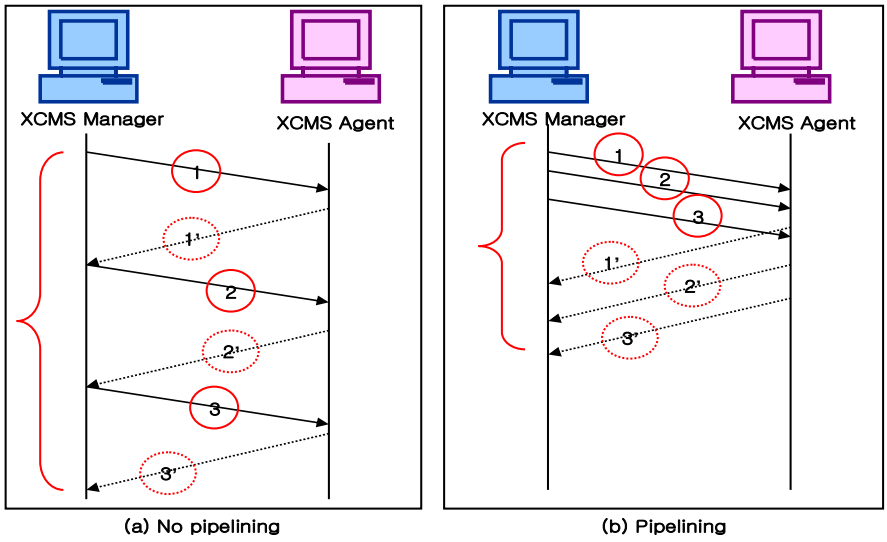


그림 7. No pipelining/ Pipelining 통신

'pipelining' 방식의 효율성을 측정하기 위해서, 우리는 명령어의 개수를 다르게 하여 응답 시간을 측정하였다. 또한, 'pipelining'방식을 사용하지 않은 경우에는 매니저가 응답 메시지를 받는 즉시, 요청 메

시지를 전송하는 방식으로 수행하여 측정하였다. 즉, 관리자가 메시지를 입력하고 생성하는 시스템 이외의 요소는 배제하였다. 전송 프로토콜은 SSH를 사용하였다.

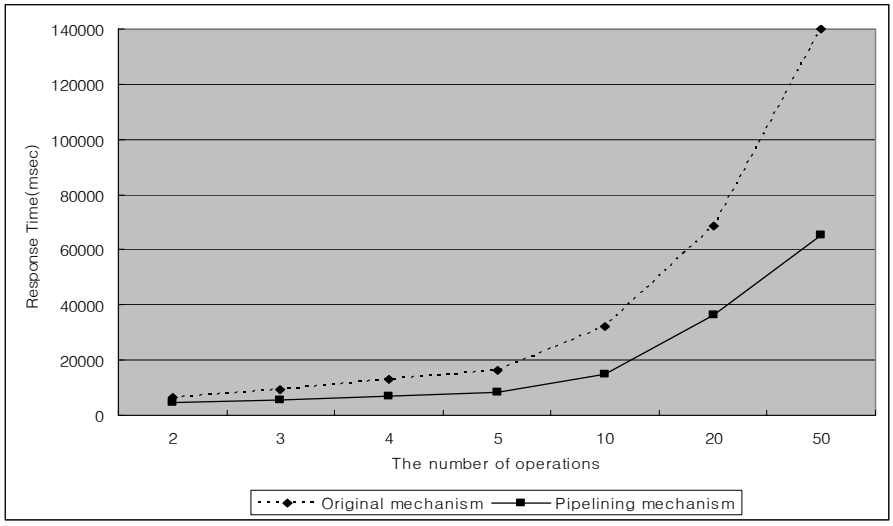
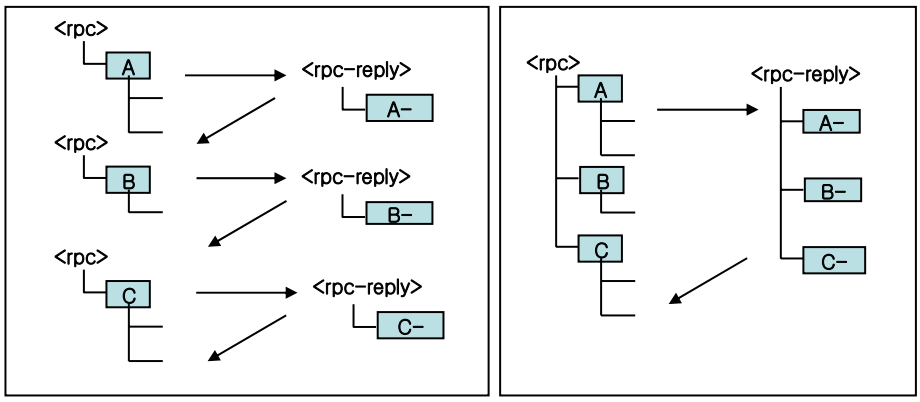


그림 8. No pipelining/pipelining response time (msec)

그림 8에서 보면, 오퍼레이션의 개수가 증가 할수록 ‘pipelining’의 응답 시간이 줄어 드는 것을 확인 할 수 있다. 그 이유는 그림 7의 시나리오 처럼, 이전 오퍼레이션이 수행되는 동안에 다음 요청 메시지가 전송이 되므로, NETCONF 에이전트는 새로운 요청 메시지를 기다리고 있지 않아도 된다. 또한, NETCONF 매니저는 응답 메시지를 전송 받은 뒤에, 새로운 요청 메시지를 전송하는 공백을 줄일 수 있으므로, 위와 같은 결과가 나온다. 따라서, 오퍼레이션의 개수가 많아질수록 ‘pipelining’을 사용한 경우와 사용하지 않은 경우의 응답 시간의 값 차이는 더욱 커지는 것이다. 하지만, ‘pipelining’을 사용함으로써, 얻을 수 있는 위험성은, 매니저가 메시지를 확인하기 전에 새로운 메시지를 전송해야 하므로, 잘못된 수행을 하는 경우가 있을 수 있다. 이러한 경우를 위해서, NETCONF 매니저는 ‘discard-chage’ 오퍼레이션

이나 ‘roll-back’ 기능을 제공하는 것이 선호된다.

‘pipelining’은 응답 시간 부분에서 효율성을 증가시키지만 네트워크 사용량에는 영향을 미치지 못한다. 현재 NETCONF는 <rpc> 요소 아래에는 단 한 개의 오퍼레이션만을 사용한다. 하지만, NETCONF의 오퍼레이션들은 대부분 서로 연관관계를 지닌다. 예를 들면, <candidate>를 수정하는 경우에는 해당 값을 읽는 작업, 수정하는 작업, 읽어오는 작업, ‘commit’하는 작업, <running>값을 확인하는 작업등이 순서대로 요구 된다. 즉, 하나의 작업을 완료하기 위해서는 연관된 여러 작업을 같이 수행해야한다. 하지만, 그림 6의 메시지들처럼 RPC 계층을 지나간 메시지들은 전송 프로토콜들의 여러가지 헤더들이 추가되어 해당 전송 프로토콜의 payload에 첨부되어 전송된다. 또한, BEEP같은 경우에는 패킷의 순서를 확인하기 위한 SEQ 패킷 전송으로 메시지 전송 횟수 및 트래픽을 증가시키며 응답 시간 또한 증가시킨다. 따라서, 매번 모든 요청 메시지와 응답 메시지에 전송 프로토콜의 오버헤드가 붙고, 네트워크 상에서 지연 된다면, 많은 장치들을 상대로 수행하는 경우에 비 효율적이다. 해당 논문에서는, 이러한 비 효율성을 해결하기 위한 방안으로 ‘multi command’ 방법을 제시한다.



(a) Original mechanism (b) Multi command mechanism

그림 9. Original/Multi command mechanism

그림 9는 원래의 NETCONF 통신 방식과 multi command 방식을 비교하는 그림이다. ‘Multi command’ 방식은 그림 9(b)와 같이 한 개의 <rpc>아래에 여러 개의 오퍼레이션들을 넣는 것이다. 단, NETCONF의 원래의 개념과 동일하게 순서적으로 오퍼레이션들을 수행하며, 만약 중간에 오퍼레이션이 오류를 일으키는 경우에는 수정된 값들을 원상 복귀시키는 ‘roll-back’기능을 수행한다. 또한, 해당 방식은 NETCONF의 RPC 계층의 기본 개념을 따라 한 개의 rpc 요청 메시지에는 단 한 개의 rpc-reply 응답 메시지를 전송한다.

<table>
<tr><th>Original NETCONF Request Messages</th><th>Multi command NETCONF Request Message</th></tr>
<tr><td><?xml version='1.0'?>
<rpc message-id='1'>
<edit-config>
<target>
<candidate/>
</target>
<config>
<interfaces>
<iface operation="merge">
<name>eth1</name>
<type>ethernet</type>
</iface>
</interfaces>
</config>
</edit-config>
</rpc></td><td rowspan="3"><?xml version='1.0'?>
<rpc message-id='1'>
<edit-config>
<target>
<candidate/>
</target>
<config>
<interfaces>
<iface operation="merge">
<name>eth1</name>
<type>ethernet</type>
</iface>
</interfaces>
</config>
</edit-config>
<commit/>
<get-config>
<source>
<running/>
</source>
</get-config>
</rpc></td></tr>
<tr><td><?xml version='1.0'?>
<rpc message-id='1'>
<commit/>
</rpc></td></tr>
<tr><td><?xml version='1.0'?>
<rpc message-id='1'>
<get-config>
<source>
<running/>
</source>
</get-config>
</rpc></td></tr>
</table>

표 3. Multi command NETCONF Request Message

표 3은 ‘multi command’ 방식을 사용하는 경우의 요청 메시지 예

제이다. 해당 메시지는 <candidate>에 대해서 ‘edit-config’를 수행한 후에, ‘commit’을 수행하여 <running>에 <candidate>의 내용들을 반영하고 반영된 결과를 확인하는 작업을 요청하는 메시지이다. ‘Multi command’ 방식 사용시의 성능 측정 결과는 5장에서 살펴보도록 하겠다.

3.1.3 Operations Layer

NETCONF에서의 operations 계층은 실제적인 구성정보 수행 내용들을 정의한다. Operations 계층은 구성 정보 관리에 필요한 ‘get’과 ‘set’의 역할을 지닌 여러 오퍼레이션들을 제공하고 있다. ‘set’기능은 관리 대상인 네트워크 장비들의 구성 정보를 변경하므로써 네트워크 장비들의 상태에 직접적인 영향을 끼친다. 따라서, ‘set’기능을 수행하는 오퍼레이션들의 성능은 구성 관리에 있어서 중요한 요소이다. NETCONF 프로토콜에서는 ‘edit-config’와 ‘copy-config’ 두 오퍼레이션이 구성 정보를 설정하는 기능을 수행한다. 이번 절에서는, NETCONF의 operation layer에서의 ‘set’기능을 수행하는 오퍼레이션들의 성능을 측정한다. 또한 해당 오퍼레이션들의 비 효율적인 면을 지적하고 향상 시킬 수 있는 방안을 제시한다.

NETCONF 프로토콜의 ‘edit-config’ 오퍼레이션은 구성 정보의 특정 부분을 설정하는 기능을 수행한다. ‘edit-config’ 오퍼레이션은 세부 오퍼레이션으로 merge, create, replace 그리고 delete 오퍼레이션들을 제공한다. 세부 오퍼레이션은 그림 10(a) 예제처럼 수행대상의 속성값으로 설정되어 전송된다. NETCONF 에이전트는 여러 매니저로부터 메시지를 전송받아 수행 될 수 있다. NETCONF는 구성 정보 수정시 발생할 수 있는 충돌을 방지하기 위해 ‘lock’ 오퍼레이션을 제공하고 있다. 따라서 ‘edit-config’를 수행하기 전에, 수행 대상인 구성정보에 ‘lock’ 오퍼레이션을 수행 한 뒤에 ‘edit-config’ 오퍼레이션을 수행해야한다. NETCONF에서는 ‘edit-config’의 이러한 특징 때문에 그림 10(b)예제처

럼 하나의 ‘edit-config’ 오퍼레이션에 여러 개의 세부 오퍼레이션들을 포함시키는 ‘multiple operations’방식을 제시하였다. 본 연구에서는 ‘multiple operations’의 효율성을 제시하기 위하여 성능을 측정하고 기존 방식과 비교 분석하였다.

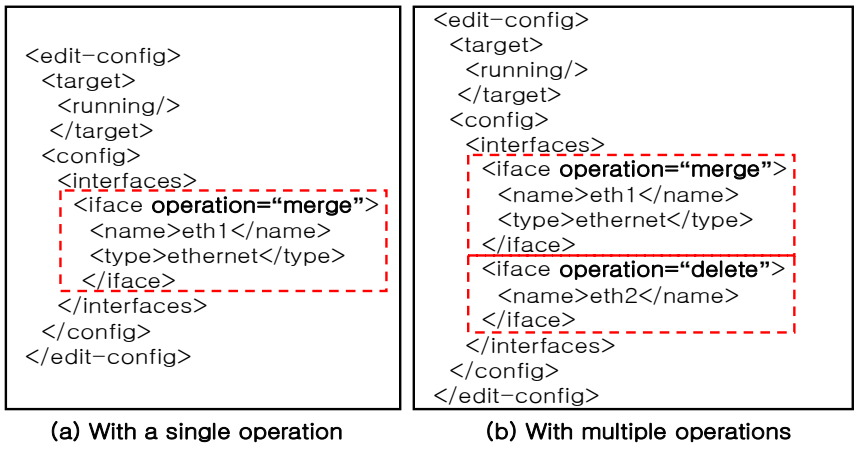


그림 10. ‘edit-config’ 요청 메시지 예

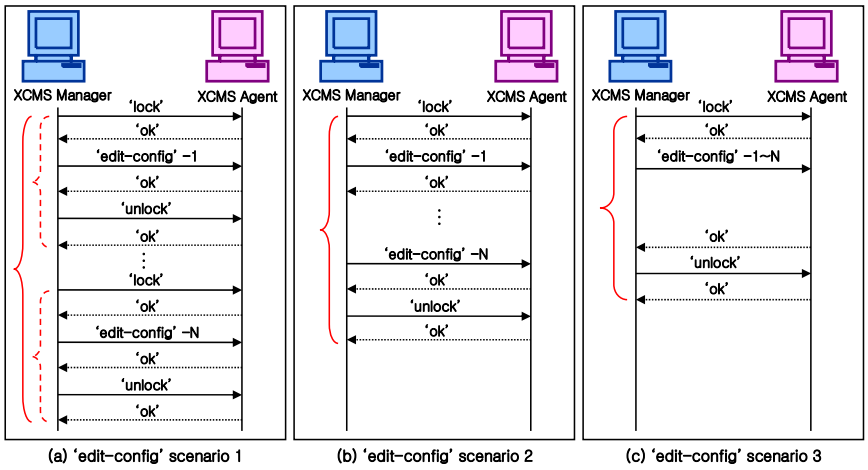


그림 11. ‘edit-config’수행 시나리오

본 연구는 ‘edit-config’ 오퍼레이션을 사용하여 구성관리를 수행하는 시나리오를 그림 11처럼 3종류로 구분하여 실험을 수행하였다.

먼저, 그림 11(a)는 ‘lock’과 한 개의 세부 오퍼레이션을 지닌 ‘edit-config’ 그리고 ‘unlock’을 매번 반복하는 방식이다. 그림 11(b)는 초기에 ‘lock’을 수행 한 뒤, 한 개의 세부 오퍼레이션을 지닌 ‘edit-config’작업들이 모두 끝나면 unlock을 수행하는 방식이다. 마지막으로 그림 11(c)는 ‘multiple operations’ 방식을 사용하는 경우이다.

우리는 ‘multiple operations’ 방식을 사용할 때의 ‘edit-config’의 수행 시간을 측정하기 위해서 세부 오퍼레이션의 개수를 증가시키며 실험을 수행하였다. 본 실험은 그림 10(a)의 메시지를 사용하였다. 또한 세부 오퍼레이션들 간의 성능 차이를 무시하기 위해서, 동일한 세부 오퍼레이션의 개수만을 증가시키며 수행하였다.

그림 12는 세 종류 시나리오의 응답 시간을 측정한 도표이다. 그림 12를 보면 ‘multiple operations’을 사용하는 ‘edit-config’의 응답 시간이 현저하게 짧은 것을 확인할 수 있다. 특히 오퍼레이션의 개수가 많아 질수록 차이는 더욱 커진다. 관리자가 많은 네트워크 장비를 관리해야 하는 경우에 ‘multiple operations’의 사용은 시간측면에서의 많은 이점을 가져다 준다.

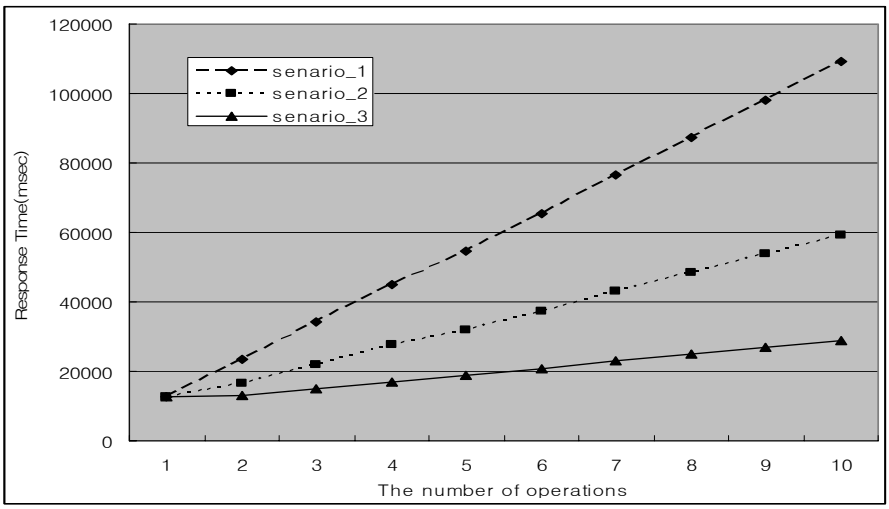


그림 12. ‘edit-config’ response time(msec)

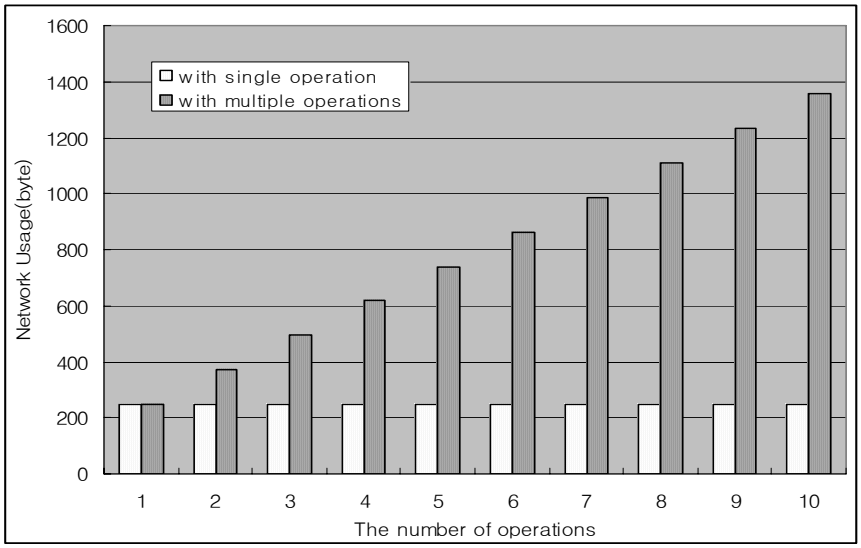


그림 13. 'edit-config' network usage

하지만, 'multiple operations'방식은 네트워크 사용량 측면에서 낮은 성능을 보일 수 있다. 해당 방식은 한 번에 많은 수행 내용을 전송하므로, 요청 메시지 크기는 클 수 있다. 하지만, 다른 두 방식은 여러 번에 나누어 전송하므로 한 번의 요청 메시지는 크지않다. 본 연구는 네트워크 사용량을 측정하는 실험을 수행하였다. 전송 횟수는 네트워크 사용량에 영향을 미치지 않으므로, 한 번에 전송되는 요청 메시지 크기를 측정하였다. 그림 13은 실험 결과를 도표로 나타낸 것이다. 'multiple operations' 방식은 위험성을 동반한다. 오퍼레이션 수행 중 단 한 개의 오류라도 발생하면, 전체 수행 내용이 잘못 될 수가 있다. 따라서, 해당 오퍼레이션 수행 중 단 한개의 오류라도 발생하면, 수행한 모든 내용을 취소하는 'rollback'기능이 제공되어야만 한다. 또한 'multiple operation'방식은 'rollback'후 정상적으로 수행 된 오퍼레이션까지 모두 재수행 해야 하는 반면에, 한 개의 세부 오퍼레이션을 여러 번 수행하는 방법은 오류가 발생한 특정 오퍼레이션만 재수행하면 된다. 해당 특징은 성능을 감소시키는 요소가 될 수 있다.

구성 정보를 설정하는 기능을 수행하는 NETCONF 프로토콜의 또 다른 오퍼레이션은 ‘copy-config’이다. ‘copy-config’는 ‘edit-config’와는 다르게 구성 정보의 특정 부분을 수정할 수 없다. 현재 ‘copy-config’는 오직 구성 정보단위로 수행된다. 해당 방식은 간단하므로 사용하기 쉽지만, 실제 장비에 적용시 다음과 같은 비효율성들이 발생한다.

첫째, 네트워크 장비의 구성 정보는 큰 규모일 수 있다. 큰 규모의 구성정보에서 아주 적은 부분만을 복사하고자 할 때, 전체를 복사하는 작업은 불필요한 오버헤드를 생성한다.

둘째, 관리자는 특정 네트워크 장비의 구성 정보를 다른 네트워크 장비로 복사하는 작업을 수행할 수 있다. 하지만, 구성 정보의 특정 부분만을 복사하고자 할 때 현재의 ‘copy-config’방식으로는 수행할 수 있는 방법이 없다. 즉, 구성 정보 전체를 복사하는 방법만이 존재한다.

셋째, ‘copy-config’는 <running>를 목표 대상으로 설정하고 수행할 수 없다. <running>에 반영하기 위해서는 <candidate>를 ‘commit’하는 방법만이 존재한다. 이 때, ‘commit’은 구성 정보 전체를 반영한다. 따라서, 불필요한 부분까지 반영하므로 오버헤드가 생긴다.

넷째, 여러대의 NETCONF 매니저가 동일한 NETCONF 에이전트에 대해서 구성 관리를 수행 할 수 있다. 어떤 매니저가 <candidate>에 수정 작업 수행 후 ‘commit’작업을 하지 않은 상태에서, 또 다른 매니저가 <candidate>에 수정 작업 수행 후 ‘commit’ 작업을 한다면 원하지 않은 구성 정보에까지 설정 작업이 이루어 질 수 있다.

본 연구에서는 위의 비 효율성들을 모두 해결하기 위한 방법으로, ‘copy-config’에 필터링 방식을 추가하는 새로운 방법을 제시한다. 필터링 방식은 ‘get-config’에서 사용하는 방식과 동일하다. 따라서, 필

터링을 사용하지 않은 경우에는 그림 14처럼 기존의 방식과 동일하게 구성 정보 전체를 복사한다. 하지만, 필터링을 사용한 경우에는 그림 15처럼 필터링 내용에 맞는 부분만 복사가 수행된다.

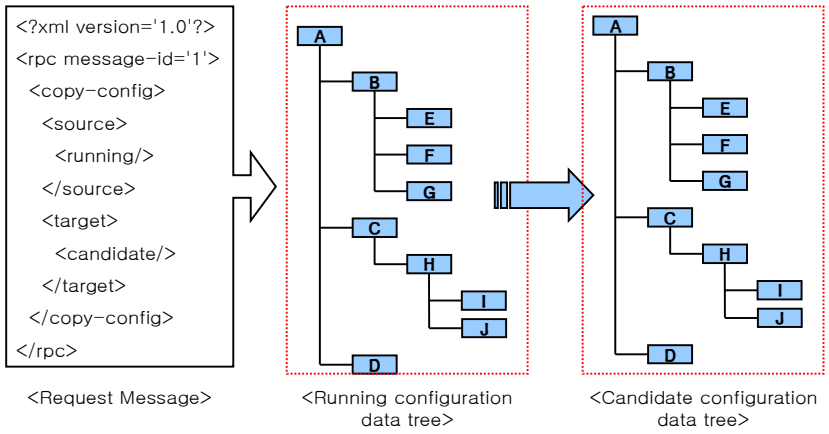


그림 14. 'copy-config' without filtering

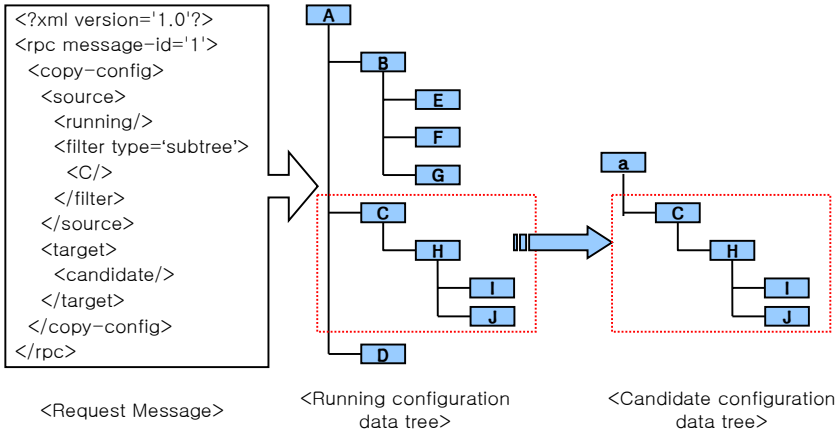


그림 15. 'copy-config' with filtering

'copy-config'에 필터링 방식을 적용시킴으로써, 여러가지 이점을 얻을 수 있다. 먼저, 구성 정보의 특정 부분만을 복사할 수 있다. 이는 구성 정보 전체를 복사하지 않아도 되므로 응답 시간이 줄어든다. 또한, 네트워크 장비들간의 구성 정보 전달 작업을 좀 더 효율적으로

수행할 수 있다. 뿐만 아니라, 그림 15의 그림처럼 구성 정보의 일부분만을 <candidate>에 복사함으로써, ‘commit’시 충돌을 방지 할 수 있으며 수행 시간도 줄일 수 있다.

3.1.4 Content Layer

NETCONF에서의 content 계층은 구성 정보들을 XML 형태로 표현하는 역할을 수행한다. 현재 관리되어야 하는 구성 정보들의 속성 및 구조는 아직 표준화가 되어 있지 않은 상태이며, NetMod[37]를 통해서 진행중에 있다. 따라서, 해당 구성 정보의 특징은 NETCONF 구현물에 의존하는 상태이다. 즉, 관리하려는 장치들마다 구성 정보의 구조는 동일하지 않을 수 있다. 이번 절에서는, NETCONF의 content 계층에서 수행되는 작업들의 효율성을 증명한다. 또한, 구성 관리를 수행하는데 있어서, content 계층에서의 비 효율성을 지적, 향상 시킬 수 있는 방안을 제시한다.

NETCONF는 구성 정보의 특정 부분에 대해서만 수행하는 방법으로 필터링 방식을 제시하고 있다. 특히, 구성 정보 값을 읽어 오는 ‘get’과 ‘get-config’ 오퍼레이션은 구성 정보에 대해서 두 종류의 필터링 방식을 제공한다. 해당 필터링 방식들중 하나는 NETCONF에서 기본으로 제시하고 있는 ‘subtree filtering’ 방식이고, 나머지 하나는 XML[26]의 기술인 ‘XPath[25]’ 방식이다. 두 방식은 서로 비슷하면서도 많은 차이점을 보인다. 먼저, ‘subtree filtering’은 NETCONF에서만 사용되고 있는 방식이다. 즉, NETCONF에서 만든 필터링 방식이다. ‘subtree filtering’의 규칙은 파악하기가 많이 어려우며, 의미가 정확하지 않은 부분들이 많다. 또한, 정확한 사용방법의 예시가 많이 부족하여서 구현시 어려움이 있으며, 각 구현물마다 방식이 다소 차이가 있을 수 있다. 반면에, ‘XPath’는 일반적인 ‘XPath’의 표현법을 그대로 사용한다. ‘XPath’의 표현만을 사용하면 되므로, 사용하는 방법은 간단하

지만, 해당 표현 방법이 다소 복잡하다. 이러한 기능적 장,단점 이외에 성능적 측면에서의 수치적 증명이 필요하다. 왜냐하면, NETCONF에서는 'XPath'가 수행 시간 및 메모리 측면에서 다소 부담이 되는 특징을 지녔다며, 'XPath'대신에 새로운 'subtree filtering'의 규칙들을 만들어서 제시하였다. 'XPath'는 수행 대상 메세지를 파싱하기 위해서, 해당 내용을 DOM[20]에 올려서 파싱 작업을 수행한다.

우리는 두 필터링 방식의 성능적 차이점을 알아보기 위해서 실험을 수행하였다. 'get-config' 오퍼레이션을 사용하였으며, 두 경우로 나누어서 실험을 수행하였다. 첫 번째 실험은 한 번의 파싱 작업이 아닌, 여러 경우의 파싱 작업을 수행한 후, 해당 결과를 합치는 방식이다. 즉, merge 작업을 수행할 때의 두 필터링 방식의 응답시간과 메모리 사용량을 측정하였다. 두 번재 실험은 간단한 파싱 작업을 수행할 때의 두 필터링 방식에 대해서 동일하게 응답시간과 메모리사용량를 측정하였다.

	Subtree Filtering	**XPath**
Request Message_1 (Using merge)	<filter type="subtree"> <interfaces> <iface> <name>eth0</name> </iface> <iface> <mtu>1000</mtu> <name/> </iface> </interfaces> </filter>	<filter type="xpath"> //name[.='eth0']|//mtu[.='1000'] </filter>
Processing Time	2.085 ms	1.901 ms
Memory Usage	1816 KB	1828 KB
Request Message_2 (Using simple)	<filter type="subtree"> <interfaces> <iface> <name/> </iface> </interfaces> </filter>	<filter type="xpath"> //name </filter>
Processing Time	1.716 ms	1.954 ms
Memory Usage	1812 KB	1828 KB

표 4. Subtree flitering/ XPath

표 4는 두 실험에서 사용한 NETCONF 요청 메시지와 실험 수행 결과 값들이다. 각 요청 메시지의 수행 결과 메세지는 두 필터링 방식에서 동일하다. 표 4에 나온 결과 값을 살펴보면, 두 실험의 성능 차이가 서로 다르다. 먼저, 첫 번째 실험인 merge 기능이 필요한 경우에는 'XPath'를 사용하는 경우가 수행 시간이 더 적게 걸리는 것을 확인할 수 있다. 이는, 모든 구성 정보를 DOM에 올려놓고 파싱을 수행하는 'XPath'에서는 merge작업과 단순한 작업에 있어서의 차이가 적을 것이다. 하지만, 요청 메시지의 내용과 구성 정보를 일대일로 매칭하면서 수행하는 'subtree filtering'은 여러 작업을 합쳐야 하는 작업에 대해서는 좀 더 많은 수행 시간을 요구한다. 반면에, 간단한 수행을 요구한 두번 째 실험에서는 'subtree filtering'이 더 적은 시간이 걸리는 것을 확인할 수 있다. 이는 단순한 작업 임에도 DOM에 모든 메시지를 올리고 수행 하는 'XPath'의 특징 때문이다. 마지막으로, 메모리 사용량에 있어서는, subtree는 작업을 수행함에 따라서 차이가 생기지만, 'XPath'는 작업의 복잡성 및 내용에 관계 없이 항상 동일한 메모리를 사용한다. 또한, 모든 데이터를 메모리에 올리고 사용하는 'XPath'가 더 많은 양을 사용하는 것을 확인할 수 있다.

NETCONF의 content들은 XML 메시지 형태이다. 즉, 구성 정보들은 트리 형태로 제공된다. NETCONF 프로토콜은 구성 정보 트리에 대해서 여러가지 구성 관리 오퍼레이션들을 수행 한다. 앞에서 실험한 'subtree filtering'이나 'XPath'를 이용하여 구성 정보를 모니터링 하거나 새로운 값으로 설정하려고 할 때, 구성 정보 트리 구조를 이용하여 수행 대상을 찾는다. 구성 관리의 대상인 네트워크 장비들에서는 구성 정보 트리의 규모가 클 수 있다. 일반적으로, 네트워크 장비에서 요구하는 구성 관리 대상인 구성 정보들은 다양하며 많다. 예를 들면, 해당 네트워크 장비에서 네트워크 관련 설정이 필요한 구성 정보는 interface, routing, VLAN, Dynamic Port 이외 에도 다양하며 많다. 또, SNMP 서비스를 제공하기 위해서 설정이 필요한 정보가 있으며,

사용자 관리를 하는 구성 정보 부분도 있을 것이다. 그 외에, 해당 네트워크 장비에서만 요구하는 구성 정보등 그 규모와 종류는 다양하다.

NETCONF 프로토콜을 이용한 구성 관리에서는 정확한 오퍼레이션들을 수행하기 위해 구성 정보 트리의 구조를 올바르게 이용해야 한다. 하지만, 앞에서 언급한 사실처럼 네트워크 디바이스는 많은 부분들로 이루어진 복잡하고 큰 구성 정보 트리 구조를 가질 수 있다. 또한, 네트워크 디바이스의 구성 정보 트리는 새로운 정보가 추가되고 삭제되면서 해당 구조가 변할 수 있다. 관리자가 이러한 구성 정보 구조를 모두 올바르게 파악하는데는 한계가 있다. 구성 관리 시스템에서 해당 구조를 따로 파악하고 저장해 놓고 관리자에게 알려주는 방법이나 관리자가 설명서를 보고 수행해야 하는 부가적인 노력이 필요하다. 현재 NETCONF 프로토콜에서 전체 구성 정보의 트리 구조를 알수 있는 방법으로 제공하고 있는 것은 필터링기능을 사용하지 않은 'get-config' 오퍼레이션이다. 하지만, 해당 오퍼레이션은 그림 16에서 보여주는 결과 메시지 처럼 전체 구성 정보 트리의 구조 뿐만 아니라, 모든 데이터 값까지 채워서 읽어오는 작업을 수행한다.

```
<?xml version="1.0"?>
<rpc-reply message-id="1">
 <data>
  <interfaces>
   <iface>
    <name>eth0</name>
    <address>141.223.82.1</address>
    <netmask>255.255.255.0</netmask>
    <bcast>255.255.255.255</bcast>
    <mtu>1500</mtu>
   </iface>
   <iface>
    <name>eth1</name>
    <address type="ipv4">141.223.82.3</address>
    <netmask>255.255.255.0</netmask>
    <bcast>255.255.255.255</bcast>
    <mtu>1000</mtu>
   </iface>
  </interfaces>
  <ng-mon ip="141.223.82.144">
   <admin name=" " email=" "/>
   <database user="root" password="********"/>
   <packetcapture>
    <device name="eth1"/>
   </packetcapture>
   <flowgenerator interval="2"/>
   <flowstore>
    <p2p file="p2p.xml"/>
   </flowstore>
  </ng-mon>
  <user>
   <name>root</name>
   <type>superuser</type>
   <full-name>Charlie Root</full-name>
   <company-info>
    <dept>1</dept>
    <id>1</id>
   </company-info>
  </user>
  <user test_name="root">
   <name>fred</name>
   <type>admin</type>
   <full-name>Fred Flintstone</full-name>
   <company-info>
    <dept>2</dept>
    <id>2</id>
   </company-info>
  </user>
  <user>
   <name>barney</name>
   <type>admin</type>
   <full-name>Barney Rubble</full-name>
   <company-info>
    <dept>2</dept>
    <id>3</id>
   </company-info>
  </user>
 </data>
</rpc-reply>
```

그림 16. 'get-config' 오퍼레이션 수행 결과

하지만, 위와 같은 응답 메시지는 NETCONF 에이전트가 네트워크 장비로부터 모든 구성 정보의 데이터 값을 읽어서 응답 메시지에 포함시킨 뒤 전달되기 때문에 처리 시간도 길고 응답 메시지 크기도 커진다. 또한, 구성 정보 구조를 알기 위한 작업은 구성 관리 초기 뿐만 아니라, 구성 관리 중에 여러 번 수행 될 수가 있다. 즉, 단순히 구성 정보 트리 구조를 알아내기 위해 수행하는 경우에는, 해당 오퍼레이션은 많은 오버헤드를 가지고 있는 단점이 있다. 본 논문에서는 이러한 비 효율성을 해결 하기 위해서, ‘get-config’ 오퍼레이션에 관리 정보 구조만을 읽어오는 ‘config-structure’기능 추가를 제시한다.

‘config-structure’ 기능은 구성 정보 트리에서 데이터 값을 제외한 해당 구조만을 모니터링 할 수 있는 기능이다. 데이터 값을 읽는 과정이 요구 되지 않으므로, 수행 시간을 줄일 수 있으며, 데이터 값이 포함되지 않기 때문에 메시지의 크기가 줄어들고 전송 시간이 줄어든다. 또, 구성 정보 트리에서 특정 부분에 대해서 수행하기 위해 구조를 모니터링 하는 경우에는, 더욱 불필요한 정보를 추가적으로 가져 오지 않아도 된다. 뿐만 아니라, 구성 관리를 수행하는 관리자 프로그램은 네트워크 장비의 종류에 관계없이 구성 관리를 수행 할 수 있다. 그림 17는 새로 제시하는 ‘config-structure’ 기능을 NETCONF 프로토콜 요청 메시지에 적용시킨 예와 해당 메시지에 대한 응답 메시지이다.

Request Message	Response Message
<?xml version='1.0'?> <rpc message-id='1'> <get-config> <source> <running/> </source> <config-structure/> </get-config> </rpc>	<?xml version="1.0"?> <rpc-reply message-id="1"> <config> <interfaces> <iface> <name/> <address/> <netmask/> <bcast/> <mtu/> </iface> </interfaces> <ng-mon> <admin/> <database/> <packetcapture> <device/> </packetcapture> <flowgenerator/> <flowstore> <p2p/> </flowstore> </ng-mon> <user> <name/> <type/> <full-name/> <company-info> <dept/> <id/> </company-info> </user> </config> </rpc-reply>

그림 17. 'config-structure' 요청/응답 메시지 예

우리는 'config-structure' 오퍼레이션을 이용하여 수행하는 방법의 효율성을 증명하기 위해서 실험을 수행하였다. 실험에 사용한 NETCONF 요청 메시지와 그에 대한 응답 메시지는 그림 16와 그림 17의 예제 메시지들이다. 먼저 'get-config' 오퍼레이션을 이용하여 수행한 그림 16의 경우에는 총 수행 시간이 69134msec이며, 'config-structure'를 사용하여 수행한 그림 17의 경우에는 3902msec이다. 해당 결과값은, 모든 구성 정보 데이터 값들을 읽어 저장하는데 걸리는 오버헤드와 큰 사이즈의 메시지를 전송하는 데 걸리는 오버헤드로 인해 생긴 차이다. 또한, 전송되는 메시지의 크기는 'get-config'를 사용한 경우가 1495byte이며, 'config-structure'를 사용하여 수행한 경우는 500byte였다. 'config-structure'의 사용은, 구성 정보 구조에 대한 작업의 수행 시간과 네트워크 사용량등의 효율성을 향상시키는 것을 확인할 수 있다.

4 XCMS 구조

본 논문의 3장에서는 NETCONF의 효율성을 향상 시키는 방법들을 제시하고 증명하였다. 이번 장에서는, 제시한 방법들이 적용된 XCMS의 새로운 전체 구조와 구현에 대해서 살펴본 후, 매니저와 에이전트를 세부적으로 살펴본다.

4.1 XCMS

이번 절에서는 XCMS의 전체적인 구조를 살펴본다. 그림 18은 XCMS의 전체적인 구조를 나타낸다. XCMS는 크게 매니저와 에이전트로 구성되어 있으며, 인터페이스로 웹 브라우저를 제공하고 있다. XCMS 매니저와 에이전트는 모두 SOAP over HTTP, BEEP 그리고 SSH 프로토콜을 제공하며, 서버와 클라이언트의 역할을 하고 있다. 우리는 XCMS 매니저와 에이전트의 SOAP[40] over HTTP 프로토콜 구현을 위해 gsoap[21] 툴킷을 사용하였다. 또 BEEP[39]은 roadrunner[38]라는 C로 구현된 BEEP 오픈소스 구현물을 이용하였으며, 마지막으로 SSH는 포트를 포워딩하는 방식을 사용하여 구현하였다. 또한, 두 시스템 모두 압축 모듈을 지니고 있어서 전송하는 메시지를 압축하고 전송받은 메시지의 압축을 해제하는 기능을 수행한다.

두 시스템에 있는 Repository는 두 시스템이 동작하는데 이용하는 파일들을 저장한다. 먼저, 매니저는 사용자와 관리되는 장비들의 정보를 XMLDB[24]에 저장하며, 웹 브라우저에서 해당 DB를 이용한다. 매니저에 저장되어 있는 configuratoin은 장비들에게 전송할 때 사용하는 구성 정보들이다. 반면에, 에이전트에 저장되어 있는 configuration은 NETCONF의 세 가지 구성 정보 데이터인 <startup>, <running>, <candidate>이다. 각 정보의 특징에 대해서는 2장 관련연구에서 상세히 설명하였으며, 에이전트는 요청 메시지에 맞추어서 적절

한 구성 정보 데이터에 대해서 구성 관리를 수행한다. 또한, 에이전트는 여러대의 매니저에 의해서 관리가 될 수 있는데 변화되는 정보를 log파일에 기록하며, 삭제되는 구성 정보 데이터는 백업해 놓는다. XCMS매니저와 에이전트는 SOAP over HTTP 프로토콜 사용시 이용되는 WSDL을 저장하고 있다. 해당 WSDL[34]은 NETCONF SOAP over HTTP 인터넷 초안에 나오는 WSDL[4]을 따르고 있다.

XCMS의 구성관리는 XCMS 매니저가 요청 메시지를 생성하여, 세 프로토콜 중 하나를 이용해서 에이전트에게 전송을 하면, 에이전트는 해당 메시지의 오퍼레이션을 수행 한 뒤, 결과를 메시지로 생성하여 매니저에게 전송하는 과정으로 이루어진다.

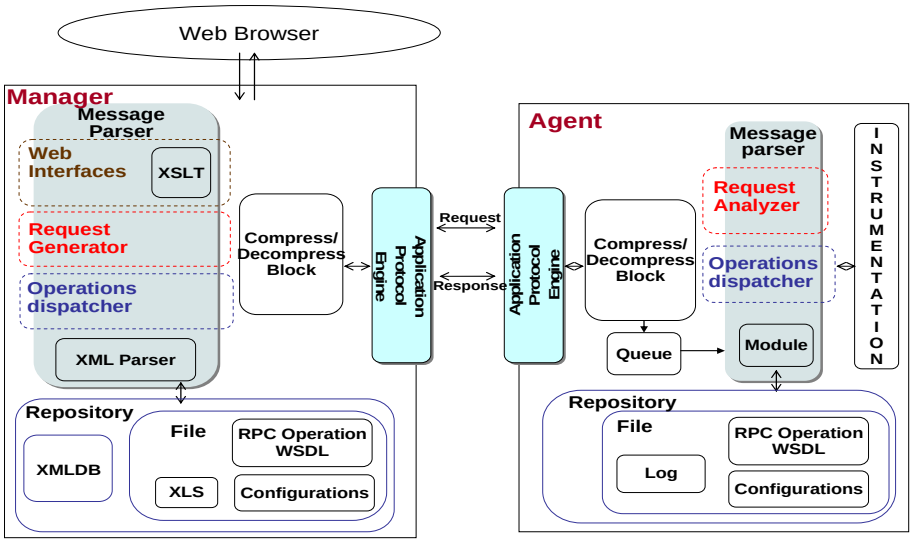


그림 18. XCMS architecture

4.2 XCMS Manager

XCMS 매니저는 두 종류의 인터페이스를 제공한다. NETCONF에서 기본적으로 요구하는 CLI를 제공하며, 웹 환경에서 동작하는 웹

인터페이스를 제공한다. XCMS 매니저는 구성 관리를 수행하려는 오퍼레이션을 NETCONF 요청 메시지 형태로 생성하여 에이전트에게 전송을 하며, 해당 에이전트로부터 전송받은 메시지를 화면에 출력을 하거나 관리자가 보기 편한 형태로 변환하여 웹에 출력하는 기능을 수행한다.

그림 19은 XCMS 매니저의 구조를 나타낸다. XCMS 매니저는 크게 서버, 압축 관련 모듈, 메시지 생성 모듈 그리고 메시지 처리 모듈로 구성되어 있다. 먼저, 서버는 에이전트 및 다른 시스템과 통신하기 위한 모듈이다. 앞서 언급했듯이, XCMS 매니저는 에이전트와 통신하기 위한 SOAP over HTTP, BEEP 그리고 SSH를 지원한다. 또한, 서버는 웹 환경과 통신하는 역할도 수행한다. 서버는 메시지를 전송하거나 메시지는 전송 받으면 프로파일 생성 모듈에 이벤트 메시지를 전송하여 시간 기록을 남긴다. XCMS 매니저는 구성 관리를 수행하기 위해서, 수행하려는 오퍼레이션, 내용 그리고 대상과 같은 파라메터들을 NETCONF 요청 메시지 형태로 생성한다. 생성한 메시지를 서버는 각 프로토콜의 payload에 적재 시켜서 전송을 수행한다. XCMS 매니저는 전송 받은 메시지를 CLI 화면에 출력 하거나, XSL[45]을 이용하여 해당 메시지를 관리자가 읽기 쉬운 형태로 웹에 출력한다.

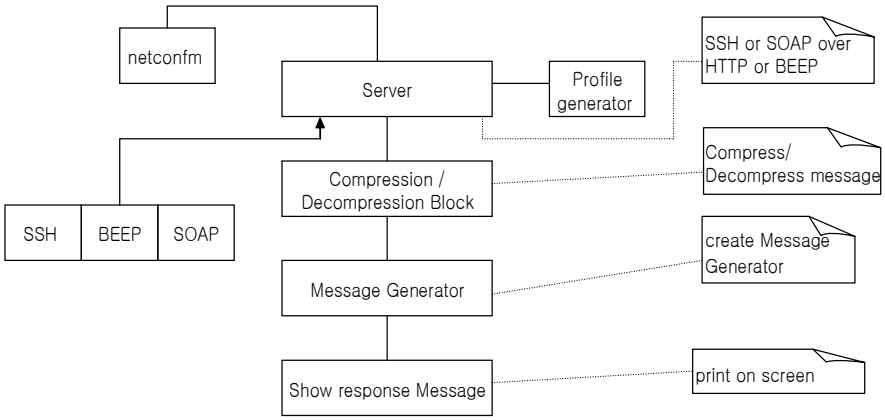


그림 19. XCMS Manager architecture

4.3 XCMS Agent

NETCONF 구현물인 XCMS 에이전트는 기본적인 NETCONF의 개념 및 기능들을 모두 제공하고 있으며, 본 논문에서 제시하고 있는 효율성을 향상시키는 방안들이 적용된 시스템이다. XCMS 에이전트는 구성 관리 대상인 장비에 장착 되어, 장비의 구성 관리 정보를 읽고 수정하며 동작한다. XCMS 에이전트는 적은 자원을 제공하는 임베디드 시스템에 적용되는 경우를 위해서, C 컴파일 언어로 구현되어 있으며 C 컴파일 언어로 구현된 전송 프로토콜들을 사용한다.

그림 20은 XCMS 에이전트의 구조를 나타낸다. XCMS 에이전트의 구조는 NETCONF의 계층에 따라 그림 20처럼 나누어진다.

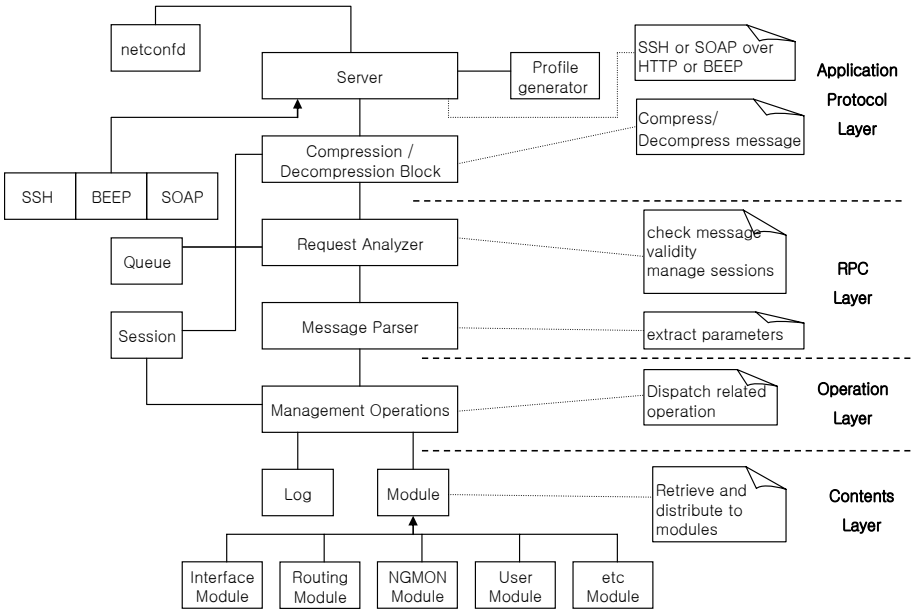


그림 20. XCMS Agent architecture

먼저, application protocol layer 부분에는 매니저와 통신하기 위한 서버가 존재한다. 서버는 3가지 전송 프로토콜을 모두 지원하며 시스템에 데몬형태로 실행된다. 본 논문에서는 application protocol layer에서

의 데이터 크기를 줄이기 위해 압축 기법의 사용을 제시하였다. 에이전트에서 Compression block이 데이터를 압축하고 압축을 해제하는 기능을 수행한다. RPC layer로부터 전송받은 메시지를 압축하여 서버에게 전송하고, 또는 서버로부터 전송받은 메시지를 압축 해제하여 RPC layer에게 전송하는 기능을 수행한다.

RPC layer 부분에는 request analyzer와 message parser가 있다. Request analyzer는 전송 받은 메시지가 NETCONF 요청 메시지인지 다른 메시지인지를 체크하는 역할을 수행한다. XCMS 에이전트는 pipelining 기능을 지원하기 위해서 queue 모듈을 두고 있다. Application protocol layer로부터 압축이 해제된 메시지는 queue에 저장된다. Request analyzer는 queue로부터 메시지를 읽어서 차례대로 수행한다. Message parser는 요청 메시지로부터 오퍼레이션 이름, 수행 대상, 수행 하려는 내용등과 같은 오퍼레이션 수행에 필요한 파라메터들을 추출하는 기능을 수행한다. RPC layer의 효율성을 향상시키기 위해 제시한 mulit command 방법은 message parser모듈에 의해서 지원된다.

Operation layer에서는 RPC layer로부터 데이터를 전송 받아서 수행한다. 해당 데이터는 message parser가 NETCONF 요청 메시지로부터 추출해 놓은 파라메터들을 모아놓은 것이다. NETCONF 오퍼레이션들의 기능들은 모두 다르므로, XCMS 에이전트는 오퍼레이션들을 모듈 단위로 나누어 수행한다. Operation manager는 RPC layer로부터 전송 받은 데이터를 적절한 오퍼레이션 모듈에 전송하는 기능을 수행한다. XCMS 에이전트는 다중 섹션을 지원한다. 섹션들을 관리하는 세션 모듈과 연관 관계를 가지며 수행한다. 특히, ‘lock’은 해당 세션만이 ‘unlock’을 수행할 수 있으므로, session 모듈과의 관계는 중요하다.

Contents layer에서는 장비의 구성 정보들을 처리하는 기능을 수행한다. XCMS 에이전트는 구성 정보에 대해 설정 작업 중 오류가 발생하면, 수행 하였던 모든 작업들을 취소하는 ‘roll back’기능을 지원한

다. 또한 <candidate>와 <startup>이 삭제되는 경우 백업 데이터를 저장한다. Log 모듈이 해당 기능들을 지원한다. 장비의 전체 구성 정보는 여러 단위로 나눌 수 있다. 예를 들면, routing에 관련된 정보들, 사용자 정보에 관련된 정보들, 네트워크에 관련된 정보들처럼 여러 큰 단위로 나눌 수 있다. 각 구성 정보 단위들은 요구하는 구성 관리 방법 및 구조가 다를 수 있다. 이에, XCMS 에이전트는 각 구성 정보 단위들을 모듈 단위로 나누어서 수행한다. Module 매니저는 operation layer로부터 데이터를 전송 받아서 해당 구성 정보 모듈에 데이터를 전송한다. 구성 정보 모듈은 구성 정보에 대해서 수행 한 뒤 결과 값을 operation layer에게 전송한다.

XCMS 에이전트는 NETCONF의 계층에 맞추어서, application protocol layer에서부터 contents layer까지 차례로 내려가면서 NETCONF 요청 메시지를 분석하고 오퍼레이션을 수행 한다. 수행이 완료 된 후에는, contents layer에서부터 application protocol layer가지 차례로 올라가면서 수행 결과를 NETCONF 응답 메시지로 만들어서 XCMS 매니저에게 전송한다.

5 XCMS 성능 측정

이번 장에서는 3장에서 제시한 방법들의 성능을 검증한다. 먼저, 성능 측정이 수행된 실험환경에 대해 설명한다. 그 뒤 NETCONF의 각 계층별로 제시한 방법들을 적용하기 전과 적용한 후의 성능을 측정하여 비교, 설명한다.

5.1 성능 측정 방법

이번 절에서는 성능 측정이 수행된 실험 환경에 대해서 설명한다. 해당 실험에서는 NETCONF 매니저와 NETCONF 에이전트로써 NETCONF 기반의 구성관리 시스템인 XCMS를 사용하였다. XCMS는 3장에서 제시한 방법론들을 적용한 구성 관리 시스템이다. 4장에서 제시한 구조로 이루어져 있으며, NETCONF의 세 전송 프로토콜인 SOAP over HTTP, BEEP 그리고 SSH를 지원한다. XCMS 매니저와 에이전트는 그림 21에 표현된 그림처럼 100Mbps Ethernet network으로 연결되어 있다.

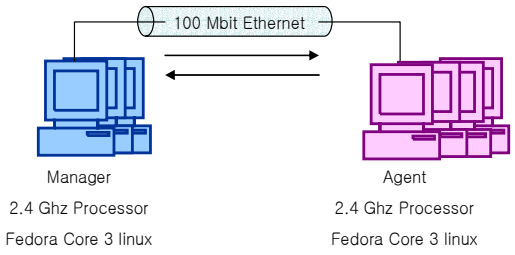


그림 21. Test setup

XCMS 에이전트는 프로토콜 기능들이 C언어로 구현되어 있으며, 전송프로토콜들은 gSOAP과 roadrunner를 사용하여 구현되어 있다. 테스트에 사용한 장치는 리눅스 시스템이며, 구성 정보는 리눅스 시스템 정보 중 네트워크 정보인 interface 정보를 사용하였다. 모든 실험

에서의 응답 시간을 좀 더 정확하게 측정 하기 위해 그림 22에 나온 그래프처럼 최소 1000번씩 측정을 하여 측정 값들의 평균 값을 이용하였다.

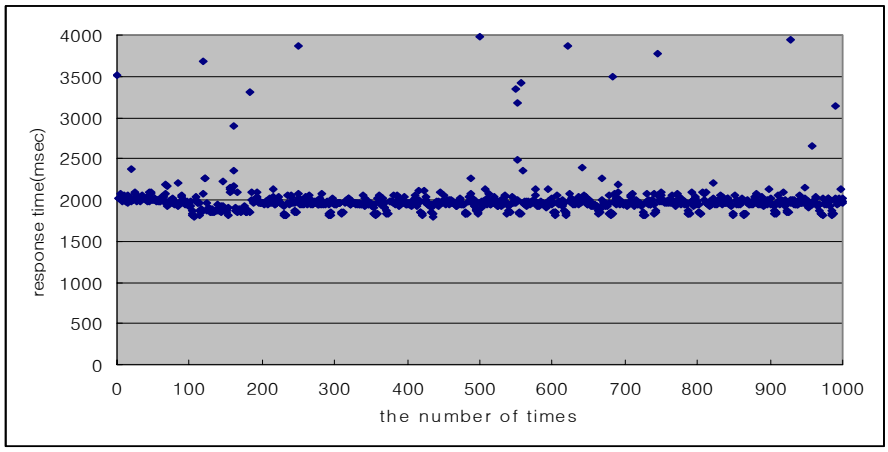


그림 22. Response Time 정확성

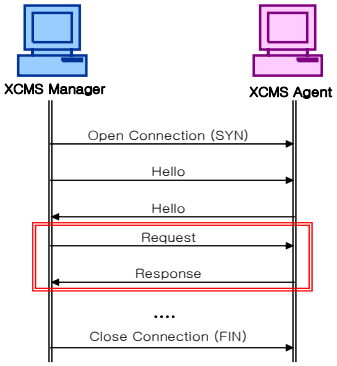


그림 23. NETCONF 통신 구조

NETCONF의 기본적인 통신 구조는 그림 23과 같다. NETCONF 매니저는 NETCONF 에이전트와 통신하기 위해서 메시지를 전송한 후, 서로간의 통신이 가능해지면, 서로의 capability 내용을 담은 Hello 메시지를 전송하면서 NETCONF 통신을 시작한다. 매니저는 NETCONF 요청 메시지를 전송하면서 구성관리를 수행한다. 우리는 NETCONF

매니저가 요청 메시지 전송 하는 부분과 응답 메시지를 전송 받는 부분들에 대해서 응답 시간과 네트워크 사용량을 측정하였다.

5.2 성능 측정

본 논문에서는 3장에서 NETCONF 프로토콜의 성능을 향상시킬 수 있는 방안들을 제시하였다. 이번 절에서는 XCMS의 성능을 측정함으로써 앞서 제시한 방법론들을 검증한다.

5.2.1 압축 방식

본 논문의 3장에서는 Application Protocol Layer에서의 성능을 향상시키기 위한 방법으로 압축 방법을 제시하였다. 이번 절에서는 XCMS를 이용하여 메시지의 크기를 변화시키며 압축을 사용시 얻는 효과를 측정한다. XML 메시지들의 특징은 데이터 값의 앞 뒤로 붙는 긴 태그들과 반복적으로 사용되는 단어들이다. 이 특징을 이용해서 XML 메시지를 압축하는 기술 중 하나인 Zlib[36]을 이용하여, 각 프로토콜의 payload 부분에 해당하는 값들에 압축 작업을 수행 후 전송하였다. 그림 24과 그림 25는 압축을 수행하지 않고 데이터를 전송하는 경우와 압축 수행 후 데이터를 전송하는 경우의 네트워크 사용량과 응답시간을 측정한 도표이다.

압축의 효과를 알아보기 위해서 다양한 크기의 데이터에 대해서 수행하였으며, 전송 프로토콜로는 XML 기술 중 대표적인 SOAP over HTTP를 사용하였다. 그림 24에서 보면, 작은 크기의 데이터를 압축시 눈에 띄는 큰 효과는 없다. 하지만, 데이터의 크기가 커질 수로 XML 메시지의 특징상 중복되는 데이터가 많아지게 되어 압축 방식을 사용하여 얻는 효과는 굉장히 크다. 또한, 우리는 압축을 수행함으로써 얻을 수 있는 오버헤드부분을 측정하였다. 그림 25의 결과에 따르

면 데이터의 크기가 적은 경우에는 압축 수행시 발생하는 오버헤드로 오히려 수행 시간이 길다. 하지만, 데이터의 크기가 커질수록 수행 결과는 다르다. 구성 관리에 관한 다른 연구 결과와 동일하게 압축 방법은 데이터의 크기가 클수록 네트워크 사용량과 응답 시간 모두에서 효과가 증가하는 것을 확인 할 수 있다. 따라서 매우 적은 데이터를 사용시에는 압축 방식이 오히려 단점으로 작용할 수 있지만, 많은 구성 정보를 관리시에는 압축 방법이 성능을 많이 향상 시킨다.

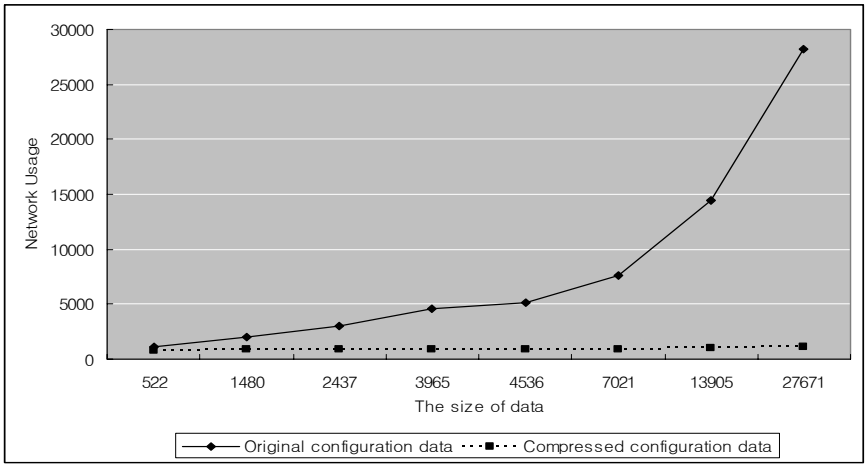


그림 24. compress/uncompress data size

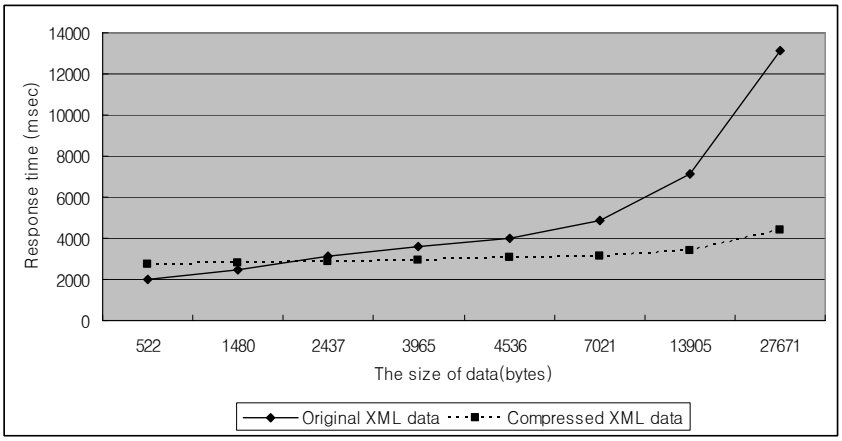
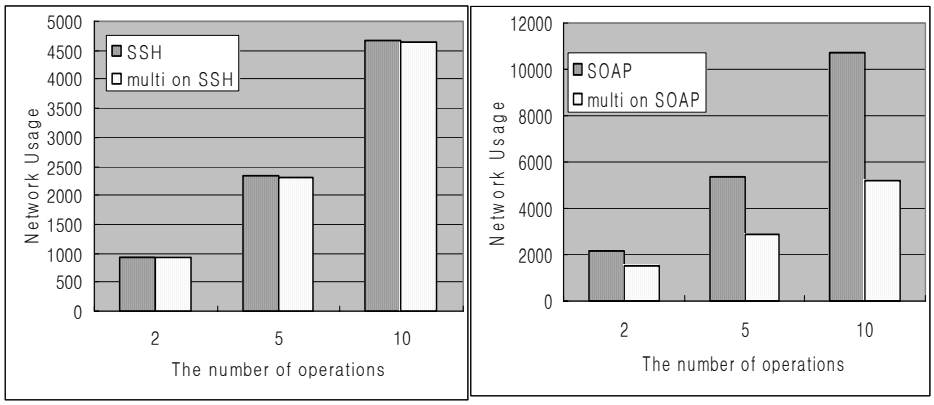


그림 25. compress/uncompress response time

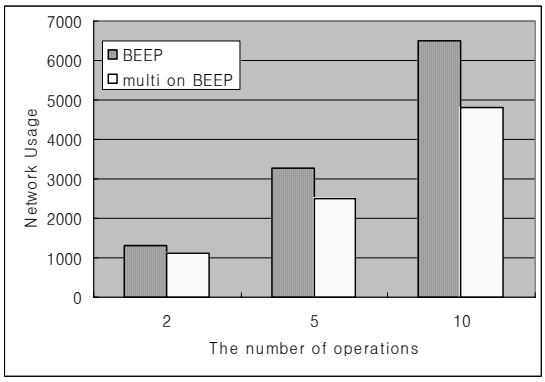
5.2.2 Multi Command

본 논문에서는 RPC Layer에서의 성능 향상 방법으로 'multi command' 방식을 제시하였다. 이번 절에서는 'Multi command' 방식으로 네트워크 사용량에서 얻는 이점을 증명하기 위해서 기존의 방식과 성능을 비교하는 실험을 수행한 결과를 설명한다. 효율성을 좀 더 공정하게 측정하기 위해서, 우리는 다양한 오퍼레이션의 개수에 따른 네트워크 사용량을 측정하였다. 데이터의 크기 변화를 쉽게 확인하기 위해서 'get-config'를 반복 사용하였다.



(a) SSH protocol

(b) SOAP over HTTP protocol



(c) BEEP protocol

그림 26. Multi command 사용시 Network Usage

그림 26은 ‘multi command’를 사용한 경우와 사용하지 않은 경우의 모든 전송 프로토콜에서의 네트워크 사용량을 측정한 결과이다. SSH같이 추가로 붙는 데이터가 적은 경우에는 ‘multi command’의 방식의 이점을 확인하기 힘들다. 응답 시간측면에서 본다면, 오히려 ‘pipelining’ 방식을 사용하는 것이 더 효율적일 수 있다. 하지만, 나머지 두 프로토콜에서는 ‘multi command’를 사용한 경우와 사용하지 않은 경우의 차이를 확연하게 보여준다. 특히, 많은 데이터가 추가로 붙는 SOAP의 경우에는 응답 시간 측면에서는 가장 좋은 성능을 보였던 반면에, 네트워크 사용량 부분에서 가장 낮은 성능을 보였었다. 이러한 SOAP의 단점을 해결할 수 있다. 또한 ‘multi command’는 ‘pipelining’과 비슷한 개념으로 통신을 수행한다. 그림 27에서 확인할 수 있듯이, 응답 시간에서는 ‘pipelining’과 비슷한 결과를 보여 준다.

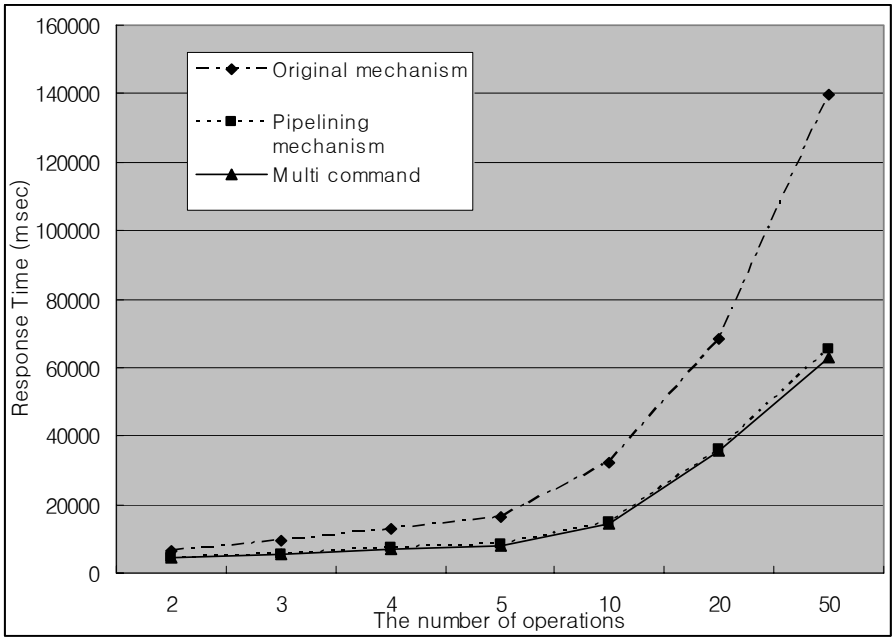


그림 27. Multi command Response Time (msec)

이와 같이, 사용하려는 프로토콜의 특징에 따라 ‘multi command’

방식의 사용은 네트워크 사용량과 응답 시간 모두에서 성능을 향상 시키는 것을 확인할 수 있다.

5.2.3 Detailed copy-config

본 논문에서는 Operations Layer에서의 성능 향상 방법으로 copy-config에 필터링을 추가한 detailed copy-config를 제시하였다. 본 절에서는 detailed copy-config를 사용한 경우와 사용하지 않은 경우의 응답 시간을 각각 측정 하였다.

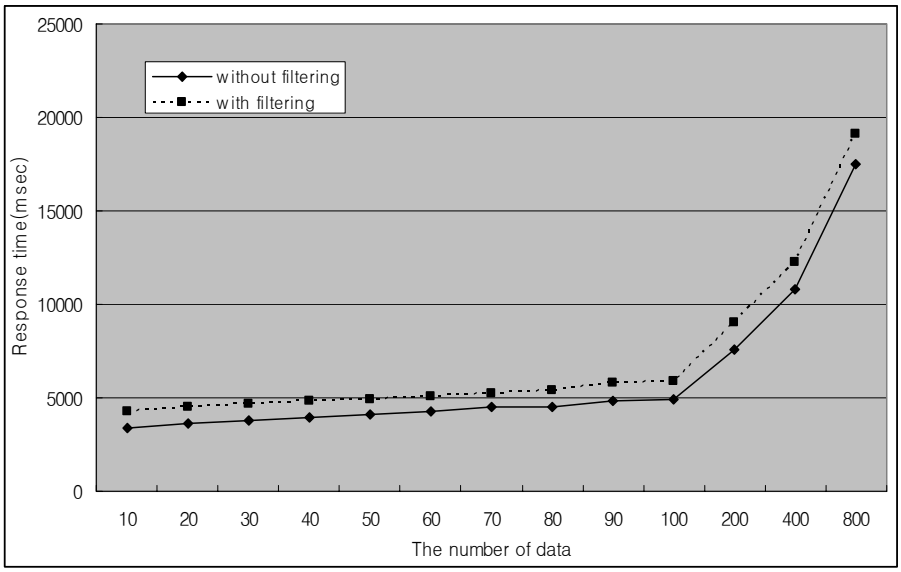


그림 28. 'copy-config' response time

그림 28은 필터링을 사용한 'copy-config'와 필터링을 사용하지 않은 'copy-config'의 응답 시간을 측정한 결과이다. 해당 실험은 복사하려는 데이터의 개수를 변화시키면서 수행하였다. 그림 28에서는 두 그래프의 차이를 비교하기 보다는 데이터의 크기 변화에 따른 응답시간의 변화량을 살펴보아야 한다. 필터링은 subtree filtering을 사용하였으며, 그림 15와 같은 간단한 필터링을 요구하였다. 그림 28에서 보면,

데이터의 개수가 증가할수록 응답 시간이 눈에띄게 증가하는 것을 확인할 수 있다. 이는 구성 정보 전체를 복사하는 작업에 비해서 특정 부분만을 복사하는 작업의 수행시간이 적게 걸리는 것을 확인 할 수 있다. 반면에, 필터링을 사용하는 경우가 동일한 데이터 개수 처리에서는 필터링을 사용하지 않는 경우보다 약간의 수행시간을 더 요구하는 것을 확인할 수 있다. 이는 대상 메시지를 파싱하는 시간이 추가가 되었기 때문이다. 따라서, 수행 시간만을 고려한다면 적은 구성 정보들에 대해서 수행시 필터링을 사용하지 않는 방식이 더 선호된다. 하지만, 앞서 언급한 것 처럼 수행 시간외에 다른 여러가지 장점들로 ‘copy-config’에 필터링의 추가는 요구된다.

6 결론 및 향후과제

NETCONF 표준화 작업은 여러 장치들간의 상호연동성을 높여 효율적인 구성관리가 가능하게 하고 있다. 하지만, NETCONF는 기능적인 측면에 비해 성능에 관한 연구결과 및 기술자료가 부족하다. 본 논문에서는 NETCONF의 효율성을 성능측면에서 계층별로 검증하며 그 효율성을 향상시킬 수 있는 방안들을 제시했다. 그리고 제시한 방안들을 적용한 구성 관리 시스템인 XCMS의 구조를 마지막으로 제시하였다.

본 논문에서는 성능 검증에서 응답시간, 네트워크 사용량 그리고 메모리 사용량들을 측정하였다. 계층별로 제시한 방안들은 다음과 같다. 첫째로 application 프로토콜 계층에서는 전송전에 데이터의 크기를 줄이는 압축기법을 제시하였다. 해당 방안은 적은 네트워크를 사용하고 빠른 관리가 가능하도록 한다. 두번째로는, RPC 계층에서는 하나의 RPC 메세지에 여러개의 오퍼레이션을 전송하는 multi command 방법을 제시하였다. Multi command 방법은 NETCONF 에이전트가 대기시간없이 오퍼레이션을 수행함으로써 응답시간을 단축하며, 네트워크 전체 사용량을 줄인다. 세번째로, operations 계층은 'copy-config'에 필터링 방식을 접목시켜 좀 더 효율적인 구성관리가 가능한 방안을 제시하였다. 마지막으로, contents 계층에서는 구성정보 구조를 전송하는 오퍼레이션을 제시하였다. 해당 오퍼레이션은 구현물에 의존적인 구성정보모델을 사용함으로써 생기는 비효율적인 점을 해결하기 위한 방안이다.

XCMS는 이번 연구로 성능과 기능이 모두 향상되었다. NETCONF 구현물인 XCMS는 실제 장치에 적용하여 구성 관리를 수행하고 다른 구현물들과 성능을 비교하는 연구가 향후 과제로 요구된다. 현재 NETCONF 구현물들은 에이전트의 위치를 알아야만 수행이

가능하다. 하지만 실제 장치들은 DHCP로 설정되는 경우가 존재할 수 있다. 또, firewall 뒤에서 장치들이 수행되는 경우도 있다. 이러한 장치들에 대해서도 수행할 수 있는 방법들에 대한 연구 또한 향후 과제로 요구된다.

참고문헌

[1] IETF, http://www.ietf.org/.

[2] IETF, “Network Configuration (Netconf)”, http://www.ietf.org/html.charters/netconf-charter.html.

[3] R. Enns, “NETCONF Configuration Protocol”, draft-ietf-netconf-prot-09, http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-09.txt,October 11, 2005.

[4] T. Goddard, “Using the Network Configuration Protocol (NETCONF) Over the Simple Object Access Protocol (SOAP)”, draft-ietf-netconf-soap-06, http://www.ietf.org/internet-drafts/draft-ietf-netconf-soap-06.txt, September 16, 2005.

[5] M. Wasserman, T. Goddard, “Using the NETCONF Configuration Protocol over Secure Shell (SSH)”, draft-ietf-netconf-ssh-05, http://www.ietf.org/internet-drafts/draft-ietf-netconf-ssh-05.txt, October 13, 2005.

[6] E. Lear, K. Crozier,“Using the NETCONF Protocol over Blocks Extensible Exchange Protocol(BEEP)”, draft-lear-netconf-beep-07, http://www.ietf.org/internet-drafts/draft-ietf-netconf-beep-07.txt, September 25, 2005.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, “Hypertext Transfer Protocol - HTTP/1.1”, RFC 2616, IETF HTTP WG, June 1999.

[8] Cisco Systems, Cisco Configuration Registrar, http://www.cisco.com/en/US/products/sw/netmgtsw/ps4618/index.html.

[9] P. Shafer and R. Enns, JUNOScript: An XML-based Network Management API, http://www.ietf.org/internet-drafts/draft-shafer-js-xml-api-00.txt, Aug. 27, 2002.

[10] YENCA, a Netconf agent for Linux implemented in C, http://madynes.loria.fr/Software_gb.htm.

[11] EnSuite, Extended Netconf SUITE, http://madynes.loria.fr/ensuite/index.html.

[12] D. Romascanu, S. Mazumdar, S. Adwankar, “Data Types for Netconf Data Models(netmod)”, http://ftp.ietf.org/internet-drafts/draft-romascanu-netconf-datatypes-01.txt, October 20, 2005.

[13] Hyoun-Mi Choi, Mi-Jung Choi, James W. Hong, "Design and Implementation of XML-based Configuration Management System for Distributed Systems", Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea, April 2004, pp. 831-844.

[14] Apache Group, “Apache”, http://www.apache.org/.

[15] Apache XML project, “Xerces Java parser”, http://xml.apache.org/xerces-j/.

[16] Apache XML project, “Xalan Java”, http://xml.apache.org/xalan-j/.

[17] Apache XML project, “Xindice”, http://xml.apache.org/xindice/.

[18] Apache XML project, “Axis”, http://xml.apache.org/axis/.

[19] Apache jakarta project, TOMCAT http://jakarta.apache.org/tomcat/

[20] W3C, “Document Object Model (DOM) Level 2 Core Specification”, W3C Recommendation, Nov. 2000.

[21] Robert A., “gSOAP: Generator Tools for Coding SOAP/XML Web Service and Client Applications in C and C++”, http://www.cs.fsu.edu/~engelen/soap.htm/.

[22] libxml, “The XML C parser and toolkit at Gnome”, http://www.xmlsoft.org/.

[23] Se-Hee Han, Myung-Sup Kim, Hong-Teak Ju and James W.Hong, “The Architecutre of NG-MON:A Passive Network Monitoring System”, Lecture Notes in Computer Science 2506, Edited by M. Feridun, P.Kropf and G.Babin, 13th IFIP/IEEE International Workshop on Distributed

Systems: Operations and Management (DSOM2002), Montreal, Canada, October, 2002, pp. 16-27.

[24] XML:DB, “XML:DB”, http://www.xmldb.org/.

[25] W3C, “XML Path Language (XPath) Version 2.0”, W3C Working Draft, November 2005.

[26] W3C, "Extensible Markup Language (XML) 1.0 3rd", W3C Recommendation, February 2004.

[27] Apostolos E. Nikolaidis et al, “Management Traffic in Emerging Remote Configuration Mechanisms for Residential Gateways and Home Devices”, IEEE Commun. Mag., Volume 43, Issue 5, May 2005, pp. 154-162

[28] A. Pras, T. Drevers, R. v.d. Meent and D. Quartel, “Comparing the Performance of SNMP and Web Services-Based Management,” IEEE eTNSM, Vol. 1, No. 2, December 2004, pp. 1-11.

[29] Mi-Jung Choi et al, "XML-based Configuration Management for IP Network Devices", IEEE Communications Magazine, Vol. 41, No. 7, July 2004. pp. 84-91.

[30] Sun-Mi Yoo, Hong-Taek Ju, James Won-Ki Hong, ",Web Services Based Configuration Management for IP Network Devices", 8th International Conference on Management of Multimedia Networks and Services (MMNS 2005), LNCS 3754, Barcelona, Spain, Oct., 2005, pp. 254-265.

[31] Vincent Cridlig, H. Abdelnur, J. Bourdellon, and Radu State, "A NetConf Network Management Suite:ENSUITE", 5th IEEE International Workshop on IP Operations and Management(IPOM 2005), LNCS 3751, Barcelona, Spain, Oct., 2005, pp. 152-161.

[32] YENCA, a Netconf agent for Linux implemented in C, http://mady-nes. loria.fr/Software_gb.htm

[33] EnSuite, Extended Netconf Suite, http://madynes.loria.fr/ensuite

[34] W3C, "Web Services Description Language (WSDL) Version 1.2" W3C Working Draft, July 2002.

[35] W3C, “XML Binary Characterization Working Group”, http://www.w3.org/2003/09/xmlap/xml-binary-wg-charter.html.

[36] ZLIB, “zlib 1.2.3”, http://www.zlib.net/.

[37] IETF, “Netconf Model (NetMod)”, draft-ietf-netconf-model-04, http://ftp.ietf.org/internet-drafts/draft-c hisholm-netconf-model-04.txt.

[38] RoadRunner, “RoadRunner BEEP framework”,http:// rr.codefactory.se.

[39] BEEP, “information on BEEP”, http://beepcore.org/

[40] W3C, "SOAP Version 1.2", W3C Working Draft, December 2001.

[41] OASIS, "Universal Description, Discovery and Integration (UDDI)", http://www.uddi.org/.

[42] Juniper, “Juniper Networks”, http://www.juniper.net/.

[43] Cisco, “Cisco Systems”, http://www.cisco.com/.

[44] J. Adiego, G. Navarro, and P. de la Fuente, “Lempel-Ziv Compression of Structured Text,” IEEE Proc. Data Compression Conf.

[45] W3C, "Extensible Stylesheet Language (XSL) Version 1.0", W3C Recommendation, October 2001.

[46] W3C, "XSL Transformations (XSLT) Version 1.0", W3C Recommendation, November 1999.

감사의 글

부족하기만한 저의 논문의 마지막 페이지를 적으며 마지막으로 그동안 저를 이끌어주시고 보살펴주신 여러 분들께 감사의 글을 올립니다.

입학 때부터 부족한 저를 믿어주시고 석사 2 년동안 많은 경험을 하게 해주시며 저의 연구를 이끌어 주신 홍원기 교수님께 큰 감사를 드립니다. 항상 세심함과 성실함으로 모범을 보이시며 매 순간순간 늘 최선을 다하시는 교수님의 모습을 본받아 어떤 분야에서든지 최선을 다하여 일하며 교수님의 이름에 걸맞는 제자가 되도록 노력하겠습니다. 더불어 저의 논문 심사를 위해 애써주신 서영주 교수님과 송황준 교수님께도 깊은 감사를 드립니다. 그리고 1 년차 시절부터 1 년반 가까이 먼 대구에서 이곳 포항까지 매주 오셔서 저의 연구를 지도해 주신 계명대 교수님이시자 연구실 선배님이신 주홍택 박사님께 감사를 드립니다. 또한 가까이서나 먼곳에서나 항상 큰 힘이 되어주시던 명섭 선배와 미정언니께 감사의 뜻을 전합니다. 그리고, 하나밖에 없는 연구실 컴공과 동기라는 이유만으로 귀찮은 일까지 모두 도와주며 항상 함께 했던 영준이, 힘든일 있을 때마다 옆에서 좋은 이야기로 위로해주고 기쁜일에는 항상 제일먼저 기뻐해주던 은희언니, 항상 뒤에서 나에게 말동무가 되어주고 나의 영어실력을 조금 향상시켜준 디팔리, 랩장이라는 이유만으로 많은 밥을 샀던 성철이, 항상 재미있게 해주시던 룡권아저씨, 내 일년후배임에도 많은 도움을 주었던 준명오빠, 랩 동기인 형조, 나만의 에디팅 담당이었던 소정이, 프로젝트할 때 많은 도움을 준 착한 창근이, 마지막으로 재밌는 사진을 보내주며 드디어 랩 후배가 된 병철이등 길다면 길고 짧다면 짧은 지난 2 년동안 기쁜일 슬픈일 힘든일 모두 함께 해준 연구실 모든 식구들께 헤어짐의 아쉬운 마음과 함께 고마움을 전합니다. DPNM 연구실은 제 인생에 있어서 가장 잊을 수 없는 곳이 될 것입니다. 싫은 소리도 잘하고 불만도 많은 저를 이쁘게 봐주고 항상 응원해주던 여러분들을 평생

잊지 못할 것이며, 여러분들이 항상 건강하기를 기도드리겠습니다.

대학생활 4 년에 이어서 이곳에 와서 지난 2 년까지 항상 내 고민, 수다 다 들어주고 기쁜일이 있으며 가장 먼저 기뻐해주고, 힘든일이 있으면 같이 고민해주며 슬퍼해주던 내 방순이면서 내 단짝 친구 주영이한테 큰 감사의 뜻을 전합니다. 주영아, 졸업축하하고 박사진학해서도 멋지게 해낼 것이라고 믿어. 이제 너 고민은 내가 들어주마. ^^ 먼곳에서도, 부족한 후배한테 항상 먼저 챙겨주시고 조언해주시며 때로는 웃음을 주셨던 중재오빠, 하소연과 장난 모두 받아주던 정호, 남철오빠, 석현이등 오즈 식구들께 고마움의 뜻을 전합니다. 멀리 있고 바쁘다는 핑계로 연락 잘 하지도 않는 친구한테 매번 먼저 연락해서 챙겨주던 내 고등학교 친구들 이정, 이수, 영주와도 이 기쁨을 같이 하고 싶습니다.

멀리서 항상 자식 걱정으로 늘 마음쓰시며 이 작은 결실에 누구보다 가장 먼저 기뻐하신 부모님께 깊은 감사를 드립니다. 또, 못난 언니를 항상 좋게 봐주며 응원해주던 하나밖에 없는 내 동생 미진이한테도 고마움을 전합니다.

마지막으로 저희 연구실이 지금까지 교수님과 선배들이 보여주었던 훌륭한 연구 실적과 끈끈한 인간애로서 더욱 발전하고 모범이 되는 연구실로 성장하기를 진심으로 바랍니다.

이 력 서

성 명 : 유 선 미

생 년 월 일 : 1981년 2월 3일

출 생 지 : 서울

주 소 : 서울시 동작구 사당4동 296-32호

학 력

1999.3 – 2004.2 : 숭실 대학교 컴퓨터학부 (B.S.)

2004.3 – 2006.2 : 포항공과대학교 컴퓨터공학과 (M.S.)

학 술 활 동

♦**Conference Papers**

- Sun-Mi Yoo, Hong-Taek Ju, James Won-Ki Hong, ",Web Services Based Configuration Management for IP Network Devices", 8th International Conference on Management of Multimedia Networks and Services (MMNS 2005), LNCS 3754, Barcelona, Spain, Oct., 2005, pp. 254-265. (SCIE)
- So-Jung Lee, Mi-Jung Choi, Sun-Mi Yoo, James W. Hong, Hee-Nam Cho, Chang-Won Ahn, Sung-In Jung, “Design of a WBEM-based Management System for Ubiquitous Computing Servers, EMW, Philadelphia, USA, September 12-15, 2004.
- 김동현, 유선미, 주홍택, 홍원기, "IP 네트워크 장비를 위한 웹 서비스 기반의 구성 관리", Proc. of KNOM 2005 Conference, Seoul, Korea, May 26-27, 2005, pp. 67-76.
- 이소정, 유선미, 주홍택, 홍원기, 안창원, "SNMP 게이트웨이를 이용한 WBEM 기반의 통합관리 시스템", Proc. of KNOM 2005 Conference, Seoul, Korea, May 26-27, 2005, pp. 151-158.

♦Journal **Papers**

- 유선미, 주홍택, 홍원기, "웹 서비스를 활용한 NETCONF 구성관리 시스템", KNOM Review, Vol. 8, No. 1, August 2005, pp. 1-13.

연구 활동

♦Projects

- Design and Implementation of WBEM-based System Framework (CIM/WBEM 기반 시스템 표준 프레임워크 설계 및 프로토타입 구현), ETRI Project, 2004
- Performance Evaluation Studies on WBEM Implementations (CIM 서버 성능 평가에 관한 연구), ETRI Project, 2005
- Development of XML-based Configuration Management System for IP Network Devices (XCMS) (네트워크 장비를 위한 XML 기반의 구성 관리 시스템 개발), DPNM Project, 2005
- Development of WebServices-based Configuration Management System for IP Network Devices(XCMS-WS)(네트워크 장비를 위한 웹서비스 기반의 구성관리 시스템 개발),DPNM Project, 2004
- Development of WBEM/SNMP Gateway(WBEM/SNMP 게이트웨이 개발), DPNM Project, 2004
- Performance of XML-based Network Management System (XML 기술을 기반으로 한 네트워크 관리 시스템 성능 연구), DPNM Project, 2004