**Network Configuration Protocol (NETCONF)**

**문서 이름**: **Request for Comments (RFC) 6241**

**발행일**: 2011년 6월

**ISSN**: 2070-1721

**카테고리**: **Standards Track**

**관련 RFC**: 이 문서는 이전의 **RFC 4741**을 대체(Obsolete)합니다.

**NETCONF 프로토콜의 목적 및 주요 특징**

1. **목적**:
   * **NETCONF**는 네트워크 장치의 \*\*구성(configuration)\*\*을 설치, 조작, 삭제하기 위한 메커니즘을 제공합니다.
2. **데이터 형식**:
   * 구성 데이터 및 프로토콜 메시지를 **XML(Extensible Markup Language)** 기반 데이터 인코딩을 사용하여 표현합니다.
3. **운영 방식**:
   * NETCONF 프로토콜은 \*\*원격 프로시저 호출(Remote Procedure Calls, RPC)\*\*로 실현됩니다. 이를 통해 네트워크 장치 간의 통신 및 제어가 이루어집니다.

**문서의 상태와 역할**

* 이 문서는 **Internet Standards Track** 문서로서, 표준화된 기술을 정의합니다.
* \*\*IETF(Internet Engineering Task Force)\*\*의 산출물이며, IETF 커뮤니티의 합의에 기반합니다.
* \*\*IESG(Internet Engineering Steering Group)\*\*의 검토 및 승인을 거쳐 공개되었습니다.

**추가 정보**

* 문서의 상태, 수정 사항(Errata), 그리고 피드백 제공 방법은 다음의 URL에서 확인할 수 있습니다:  
  <http://www.rfc-editor.org/info/rfc6241>.

**NETCONF가 대체한 RFC**

* 이 문서는 이전 표준인 **RFC 4741**을 대체(Obsoletes)하여 NETCONF의 새로운 버전을 정의합니다. 이는 기존 프로토콜의 개선 및 최신 요구 사항을 반영하기 위함입니다.

**요약**

RFC 6241은 네트워크 구성 관리 프로토콜인 **NETCONF**에 대한 표준 정의 문서입니다. 이 프로토콜은 XML 기반 데이터를 사용하며, RPC 메커니즘을 통해 네트워크 장치를 효율적으로 제어하고 관리할 수 있도록 설계되었습니다. 이는 IETF의 공식 표준 문서로, 네트워크 구성 및 관리의 핵심 기술 중 하나로 자리 잡고 있습니다.

이 문서는 IETF Trust 및 문서 작성자의 저작권 아래 보호됩니다.  
문서를 사용하는 사람은 BCP 78 및 관련 법적 조항을 준수해야 하며, 수정 및 파생 작업은 **IETF Standards Process** 내에서만 허용됩니다. 특히, 문서에서 제공된 코드는 Simplified BSD License 하에 제공되며, 보증이 없습니다.

# 목차

[목차 3](#_Toc189838549)

[1. Introduction 6](#_Toc189838550)

[1.1 Terminology 7](#_Toc189838551)

[1.2 Protocol Overview 9](#_Toc189838552)

[1.3 Capabilities 11](#_Toc189838553)

[1.4 Separation of Configuration and State Data 12](#_Toc189838554)

[2. Transport Protocol Requirements 15](#_Toc189838555)

[2.1 Connection-Oriented Operation 16](#_Toc189838556)

[2.2 Authentication, Integrity, and Confidentiality 17](#_Toc189838557)

[2.3 Mandatory Transport Protocol 18](#_Toc189838558)

[3. XML Considerations 19](#_Toc189838559)

[3.1 Namespace 19](#_Toc189838560)

[3.2 Document Type Declarations 19](#_Toc189838561)

[4. RPC Model 20](#_Toc189838562)

[4.1 <rpc> Element 20](#_Toc189838563)

[4.2 <rpc-reply> Element 22](#_Toc189838564)

[4.3 <rpc-error> Element 23](#_Toc189838565)

[4.4 <ok> Element 25](#_Toc189838566)

[4.5 Pipelining 26](#_Toc189838567)

[5. Configuration Model 28](#_Toc189838568)

[5.1 Configuration Datastores 29](#_Toc189838569)

[5.2 Data Modeling 31](#_Toc189838570)

[6. Subtree Filtering 34](#_Toc189838571)

[6.1 Overview 34](#_Toc189838572)

[6.2 Subtree Filter Components 37](#_Toc189838573)

[6.2.1 Namespace Selection 41](#_Toc189838574)

[6.2.2 Attribute Match Expressions 44](#_Toc189838575)

[6.2.3 Containment Nodes 46](#_Toc189838576)

[6.2.4 Selection Nodes 49](#_Toc189838577)

[6.2.5 Content Match Nodes 52](#_Toc189838578)

[6.3 Subtree Filter Processing 55](#_Toc189838579)

[6.4 Subtree Filtering Examples 58](#_Toc189838580)

[6.4.1 No Filter 58](#_Toc189838581)

[6.4.2 Empty Filter 59](#_Toc189838582)

[6.4.3 Select the Entire <users> Subtree 60](#_Toc189838583)

[6.4.4 Select All <name> Elements within the <users> Subtree 61](#_Toc189838584)

[6.4.5 One Specific <user> Entry 62](#_Toc189838585)

[6.4.6 Specific Elements from a Specific <user> Entry 64](#_Toc189838586)

[6.4.7 Multiple Subtrees 65](#_Toc189838587)

[6.4.8 Elements with Attribute Naming 67](#_Toc189838588)

[7. Protocol Operations 69](#_Toc189838589)

[7.1 <get-config> 70](#_Toc189838590)

[7.2 <edit-config> 74](#_Toc189838591)

[7.3 <copy-config> 77](#_Toc189838592)

[7.4 <delete-config> 79](#_Toc189838593)

[7.5 <lock> 80](#_Toc189838594)

[7.6 <unlock> 82](#_Toc189838595)

[7.7 <get> 84](#_Toc189838596)

[7.8 <close-session> 86](#_Toc189838597)

[7.9 <kill-session> 87](#_Toc189838598)

[8. Capabilities 89](#_Toc189838599)

[8.1 Capabilities Exchange 90](#_Toc189838600)

[8.2 Writable-Running Capability 91](#_Toc189838601)

[8.3 Candidate Configuration Capability 92](#_Toc189838602)

[8.4 Confirmed Commit Capabilities 94](#_Toc189838603)

[8.5 Rollback-on-Error Capability 96](#_Toc189838604)

[8.6 Validate Capability 97](#_Toc189838605)

[8.7 Distinct Startup Capability 99](#_Toc189838606)

[8.8 URL Capability 100](#_Toc189838607)

[8.9 XPath Capability 102](#_Toc189838608)

[9. Security Considerations 105](#_Toc189838609)

[10. IANA Considerations 107](#_Toc189838610)

[10.1 NETCONF XML Namespace 107](#_Toc189838611)

[10.2 NETCONF XML Schema 107](#_Toc189838612)

[10.3 NETCONF YANG Module 107](#_Toc189838613)

[10.4 NETCONF Capability URNs 107](#_Toc189838614)

[11. Contributors(기여자) 108](#_Toc189838615)

[12. Acknowledgements 108](#_Toc189838616)

[13. References 109](#_Toc189838617)

[13.1 Normative References 109](#_Toc189838618)

[13.2 Informative References 109](#_Toc189838619)

[Appendix A. NETCONF Error List 110](#_Toc189838620)

[Appendix B. XML Schema for NETCONF Messages Layer 110](#_Toc189838621)

[Appendix C. YANG Module for NETCONF Protocol Operations 110](#_Toc189838622)

[Appendix D. Capability Template 110](#_Toc189838623)

[Appendix E. Configuring Multiple Devices with NETCONF 111](#_Toc189838624)

[Appendix F. Changes from RFC 4741 111](#_Toc189838625)

# 1. Introduction

NETCONF 프로토콜은 네트워크 장치를 관리하고, 구성 데이터를 검색하며, 새로운 구성 데이터를 업로드하고 조작할 수 있도록 하는 간단한 메커니즘을 정의합니다. 이 프로토콜을 사용하면 장치가 완전한 형식의 애플리케이션 프로그래밍 인터페이스(API)를 노출할 수 있으며, 이를 통해 애플리케이션이 전체 또는 부분적인 구성 데이터 세트를 보내고 받을 수 있습니다.

NETCONF 프로토콜은 **원격 프로시저 호출(RPC, Remote Procedure Call)** 패러다임을 사용합니다.

* 클라이언트는 XML 형식으로 RPC 요청을 인코딩하여 서버로 전송합니다.
* 서버는 XML 형식의 응답을 반환합니다.
* 요청 및 응답의 내용은 XML DTD(문서 타입 정의, Document Type Definition) 또는 XML 스키마로 완전히 기술되며, 이를 통해 통신하는 양측이 교환되는 데이터의 구문(syntax) 제약 조건을 인식할 수 있습니다.

**NETCONF의 핵심 기능**

1. **기능 확장 지원**
   * NETCONF는 장치의 기본 기능을 관리 프로토콜에서 그대로 활용할 수 있도록 하여, 구현 비용을 줄이고 새로운 기능에 신속하게 접근할 수 있도록 합니다.
   * 또한, 애플리케이션은 장치의 네이티브 사용자 인터페이스(UI)의 구문 및 의미적 내용에 액세스할 수 있습니다.
2. **서버의 기능(캡터빌리티, Capabilities) 탐색 가능**
   * NETCONF 클라이언트는 서버가 지원하는 확장 기능(캡터빌리티)을 검색할 수 있습니다.
   * 이를 통해 클라이언트는 서버에서 제공하는 특정 기능을 활용할 수 있도록 동작을 조정할 수 있습니다.
   * 표준 및 비표준 기능을 엄격한 의미적, 구문적 기준을 바탕으로 정의할 수 있습니다.
3. **자동화된 구성 시스템과의 연계 가능**
   * NETCONF는 자동화된 네트워크 구성 시스템의 중요한 구성 요소로 작동할 수 있습니다.
   * XML을 기본 데이터 표현 방식으로 사용하며, XSLT(XML 스타일시트 변환 언어) 같은 XML 변환 기술과 함께 활용될 수 있습니다.
   * 이를 통해 벤더 독립적인 구성 데이터 모델을 특정 벤더, 제품, 운영 체제(OS), 소프트웨어 릴리즈에 맞춰 변환하고, 변환된 데이터를 NETCONF 프로토콜을 통해 장치에 전달할 수 있습니다.

[**RFC 2119**](#_13.1_Normative_References) **규칙**

이 문서에서 사용되는 다음 용어는 **RFC 2119**에 따라 해석해야 합니다.

* **MUST(반드시)**, **REQUIRED(필수적)**: 반드시 따라야 하는 요구 사항
* **SHALL(해야 한다)**, **SHALL NOT(해서는 안 된다)**: 엄격한 요구 사항
* **SHOULD(권장됨)**, **SHOULD NOT(권장되지 않음)**: 특정 상황에서 예외 가능
* **MAY(선택 사항)**, **OPTIONAL(선택적)**: 구현 여부가 자유로움

**설명 요약**

NETCONF는 네트워크 장치의 구성 및 관리를 위한 프로토콜로, XML 기반의 원격 프로시저 호출(RPC)을 사용하여 장치와 클라이언트 간의 통신을 수행합니다.

* 이 프로토콜을 사용하면 장치가 자체적인 API를 노출하고, 클라이언트는 이를 통해 설정을 가져오거나 변경할 수 있습니다.
* NETCONF는 기능 확장을 지원하며, 서버가 제공하는 기능을 동적으로 탐색할 수 있습니다.
* XML을 기반으로 하여 자동화된 네트워크 구성 시스템과 쉽게 연계될 수 있습니다.
* 문서 내에서 사용되는 주요 용어(MUST, SHALL, SHOULD 등)는 RFC 2119 규칙에 따라 엄격하게 해석되어야 합니다.

## 1.1 Terminology

**✅ 구성 데이터 저장소 (Configuration Datastore) 관련 용어**

1. **candidate configuration datastore (후보 구성 저장소)**
   * 현재 장치의 설정을 변경하지 않고 구성을 조작할 수 있는 저장소입니다.
   * 변경된 구성은 실행 구성 저장소(running configuration datastore)로 커밋(적용)할 수 있습니다.
   * 모든 장치가 이 기능을 지원하는 것은 아닙니다.
2. **running configuration datastore (실행 구성 저장소)**
   * 장치에서 현재 활성 상태인 구성 데이터를 저장하는 저장소입니다.
   * NETCONF에서는 항상 존재하는 필수적인 저장소입니다.
3. **startup configuration datastore (시작 구성 저장소)**
   * 장치가 부팅될 때 로드하는 초기 구성 데이터를 저장하는 저장소입니다.
   * 실행 구성 저장소와 별도로 운영되는 장치에서만 존재합니다.
4. **configuration datastore (구성 저장소)**
   * 장치를 초기 상태에서 원하는 작동 상태로 변환하는 데 필요한 **완전한 구성 데이터**를 보관하는 저장소입니다.
5. **datastore (데이터 저장소)**
   * 정보를 저장하고 액세스할 수 있는 **개념적인 저장 공간**을 의미합니다.
   * 예를 들어, 파일, 데이터베이스, 플래시 메모리 등의 형태로 구현될 수 있습니다.

**✅ 프로토콜 관련 용어**

1. **기능 확장 (capability)**
   * NETCONF의 기본 사양을 보완하는 추가적인 기능입니다.
2. **클라이언트 (client)**
   * 서버에서 제공하는 NETCONF 프로토콜 기능을 호출하는 역할을 합니다.
   * 또한, 서버에서 보내는 **알림(Notification)**을 구독할 수도 있습니다.
3. **서버 (server)**
   * 클라이언트가 요청한 NETCONF 프로토콜 작업을 실행하는 역할을 합니다.
   * 클라이언트에게 **알림(Notification)**을 보낼 수도 있습니다.
4. **세션 (session)**
   * 클라이언트와 서버가 보안이 보장된 연결을 통해 메시지를 주고받는 과정입니다.
5. **프로토콜 작업 (protocol operation)**
   * NETCONF 프로토콜에서 수행하는 특정 원격 프로시저 호출(RPC)입니다.
6. **원격 프로시저 호출 (remote procedure call, RPC)**
   * 클라이언트가 <rpc> 요청을 보내고, 서버가 이에 대한 <rpc-reply> 응답을 반환하는 방식으로 동작합니다.

**✅ 데이터 관련 용어**

1. **구성 데이터 (configuration data)**
   * 장치를 기본 상태에서 원하는 현재 상태로 변환하는 데 필요한 **설정 가능한 데이터**입니다.
2. **상태 데이터 (state data)**
   * 장치의 **운영 상태 정보를 나타내는 데이터**로, 예를 들어 읽기 전용의 장치 상태나 통계 데이터가 포함됩니다.
   * 구성 데이터와 달리, 사용자가 직접 수정할 수 없습니다.
3. **메시지 (message)**
   * 세션을 통해 전송되는 **NETCONF 프로토콜 요소**를 의미합니다.
   * 모든 메시지는 **XML 형식의 문서**로 구성됩니다.
4. **알림 (notification)**
   * 서버가 특정 이벤트를 감지했을 때 클라이언트에게 보내는 **서버 주도 메시지**입니다.

**✅ 사용자 관련 용어**

1. **사용자 (user)**
   * 클라이언트의 **인증된 ID**를 의미하며, 일반적으로 **NETCONF 사용자 이름**으로 표현됩니다.

**📌 요약**

NETCONF 프로토콜은 **구성 데이터(Configuration Data)와 상태 데이터(State Data)를 저장하는 다양한 저장소(Datastore)**를 사용하여 네트워크 장치를 관리합니다.

* **후보 구성 저장소 (Candidate Configuration Datastore)**: 변경 사항을 적용하기 전에 저장하는 임시 저장소 (모든 장치에서 지원하지는 않음).
* **실행 구성 저장소 (Running Configuration Datastore)**: 현재 실행 중인 구성 정보를 저장하는 저장소.
* **시작 구성 저장소 (Startup Configuration Datastore)**: 장치가 부팅될 때 로드하는 초기 구성 정보를 저장하는 저장소.
* **구성 데이터 (Configuration Data)**: 장치 설정을 변경하는 데이터.
* **상태 데이터 (State Data)**: 읽기 전용 정보 및 통계 데이터.

NETCONF는 **클라이언트-서버 모델**을 기반으로 작동하며, 클라이언트가 **RPC (Remote Procedure Call)** 요청을 보내면 서버가 이를 처리하고 응답합니다.

* **클라이언트(Client)**: 서버의 기능을 호출하고, 알림(Notification)을 받을 수 있음.
* **서버(Server)**: 클라이언트의 요청을 실행하고, 알림을 전송할 수 있음.
* **세션(Session)**: 클라이언트와 서버 간의 보안 연결.

## 1.2 Protocol Overview

NETCONF는 **RPC(Remote Procedure Call) 기반 통신 방식**을 사용하여 클라이언트와 서버 간의 상호작용을 단순화합니다.

* **클라이언트 (Client)**
  + 네트워크 관리자의 **스크립트 또는 애플리케이션**이 될 수 있습니다.
  + 일반적으로 네트워크 관리 소프트웨어의 일부로 실행됩니다.
* **서버 (Server)**
  + 네트워크 장치(Network Device)입니다.
  + 문서에서는 "서버"와 "장치"를 같은 의미로 사용합니다.
  + 클라이언트의 요청을 처리하고 응답을 반환합니다.

**✅ NETCONF 세션 (Session)**

* **NETCONF 세션**은 **네트워크 관리자 또는 네트워크 구성 애플리케이션과 네트워크 장치 간의 논리적 연결**입니다.
* 모든 장치는 **최소한 하나 이상의 NETCONF 세션을 지원해야 하며(MUST), 여러 개의 세션을 지원하는 것이 권장됩니다(SHOULD).**
* **세션의 속성**
  + **전역 속성(Global Attributes)**: 모든 세션에서 변경 사항이 즉시 반영됨.
  + **세션별 속성(Session-Specific Attributes)**: 해당 세션에서만 적용됨.

**✅ NETCONF 프로토콜 계층 구조**

NETCONF는 개념적으로 **4가지 계층**으로 나뉘며, 이는 **아래 그림** 과 같이 표현할 수 있습니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

**(1) 보안 전송 계층 (Secure Transport Layer)**

* **클라이언트와 서버 간의 안전한 통신 경로를 제공**합니다.
* NETCONF는 여러 보안 전송 프로토콜 위에서 동작할 수 있습니다.
  + 예) **SSH, TLS, BEEP/TLS, SOAP/HTTP/TLS** 등.
* NETCONF에서 요구하는 기본적인 전송 프로토콜 요건은 **2장에서 설명**됩니다.

**(2) 메시지 계층 (Messages Layer)**

* **RPC 요청과 응답, 알림(Notification)을 포함하는 메시지를 처리**합니다.
* 메시지는 **전송 프로토콜에 독립적인 방식**으로 구성됩니다.
* 관련 내용은 **4장에서 설명되며,** [**RFC5717**](#_13.1_Normative_References)**에서 알림(Notification) 메시지를 정의**합니다.

**(3) 작업 계층 (Operations Layer)**

* **NETCONF의 기본적인 프로토콜 작업(Operations)을 정의**합니다.
* XML을 이용하여 **RPC 메서드와 매개변수를 인코딩** 합니다.
* 주요 작업:
  + <edit-config>: 장치의 구성 데이터를 수정.
  + <get>: 장치에서 데이터를 조회.
  + <commit>: 후보 구성 저장소의 변경 사항을 실행 구성 저장소에 적용.
* **기본 프로토콜 작업 목록은 7장에서 자세히 설명됩니다.**

**(4) 콘텐츠 계층 (Content Layer)**

* NETCONF에서 **처리할 데이터(구성 데이터, 알림 데이터 등)**를 의미합니다.
* 이 문서의 범위를 벗어나지만, NETCONF 데이터를 표준화하기 위한 별도의 작업이 진행되고 있습니다.

**✅ YANG과 NETCONF**

* **YANG**은 NETCONF에서 사용할 **데이터 모델을 정의하는 언어**입니다.
* **YANG은 NETCONF의 "작업 계층(Operations Layer)"과 "콘텐츠 계층(Content Layer)"을 다룹니다.**
* [**RFC 6020**](https://datatracker.ietf.org/doc/html/rfc6020)에서 정의되었으며, **NETCONF의 데이터 모델링을 위한 표준**입니다.

**📌 요약**

NETCONF는 **RPC 기반 프로토콜**로, 네트워크 장치를 원격에서 구성하고 관리할 수 있도록 합니다.  
**4가지 계층 구조**를 기반으로 동작합니다.

1. **보안 전송 계층**: SSH, TLS 등 보안 통신을 제공.
2. **메시지 계층**: RPC 요청/응답과 알림 메시지를 처리.
3. **작업 계층**: <edit-config>, <get> 등 NETCONF의 주요 작업을 정의.
4. **콘텐츠 계층**: 구성 데이터 및 알림 데이터를 포함.

YANG은 NETCONF의 **데이터 모델을 정의하는 표준 언어**로, 주로 **작업 계층과 콘텐츠 계층을 지원**합니다.

## 1.3 Capabilities

**NETCONF의 기능 확장(Capability)이란?**

* **NETCONF 기본 사양을 보완하는 기능 집합**을 의미합니다.
* 기능 확장은 **URI(Uniform Resource Identifier,** [**RFC 3986**](#_13.1_Normative_References)**)에 의해 식별**됩니다.

**✅ 기능 확장의 역할**

기능 확장은 **장치의 기본 작업을 보완**하며, 다음과 같은 역할을 합니다.

1. **추가 작업(Operations) 제공**
   * 기본 NETCONF 명령 외에 추가적인 원격 프로시저 호출(RPC) 제공 가능.
   * 예: <validate>, <rollback-on-error> 등의 고급 작업.
2. **작업 내 허용되는 콘텐츠(Content) 정의**
   * 특정 작업에서 어떤 데이터나 매개변수가 허용되는지 결정.
3. **클라이언트가 서버 기능을 동적으로 인식(Discovery) 가능**
   * 클라이언트는 서버의 기능을 확인한 후, 해당 기능을 활용할 수 있음.

**✅ 기능 간의 종속성 (Capability Dependencies)**

* **일부 기능은 다른 기능을 필요로 할 수도 있음.**
* 특정 기능을 지원하려면 **해당 기능이 의존하는 모든 기능을 서버가 지원해야 함(MUST).**

**✅ 기능 교환 (Capabilities Exchange)**

* **클라이언트가 서버의 지원 기능을 확인하는 과정**입니다.
* **8장에서 기능 교환 방식과 NETCONF에서 정의된 기능 목록을 설명합니다.**

**✅ 기능 확장의 유연성**

NETCONF 기능 확장은 **확장성이 뛰어나며, 새로운 기능을 쉽게 추가할 수 있음.**

* **언제든지 외부 문서에서 새로운 기능을 정의할 수 있음.**
* **표준화 기구(Standards Bodies)**가 공식적인 기능을 정의할 수 있음.
* **벤더(Vendor)별 맞춤형 기능**도 정의 가능함.

⚠ **기능 URI는 중복되지 않도록 고유해야 함(MUST).**

* 이를 위해 **URI에 해당 기능을 정의한 기관(표준 기구 또는 벤더)의 명칭을 포함**해야 함.

**📌 요약**

* **NETCONF 기능 확장(Capability)** 은 **NETCONF의 기본 기능을 확장**하는 역할을 함.
* **추가 작업, 콘텐츠 정의, 클라이언트가 서버 기능을 인식하는 기능을 제공**함.
* 기능 간의 **종속성(Dependency)이 있을 수 있으며**, 서버는 **의존하는 모든 기능을 지원해야 함.**
* **클라이언트와 서버는 기능 교환(Capabilities Exchange)을 통해 지원 기능을 협상할 수 있음.**
* **새로운 기능은 표준 기구 또는 벤더에서 정의 가능하며, URI를 통해 고유하게 식별됨.**

## 1.4 Separation of Configuration and State Data

구성 데이터(Configuration Data)와 상태 데이터(State Data)의 구분

NETCONF 프로토콜에서는 **장치에서 검색할 수 있는 정보**를 두 가지 유형으로 구분합니다.

**✅ 1. 구성 데이터(Configuration Data)**

* **쓰기 가능한 데이터(Editable Data)**
* **장치가 초기 상태에서 현재 상태로 변경되는 데 필요한 정보**
* 예제: 네트워크 인터페이스 설정, 라우팅 테이블, 방화벽 규칙 등

**✅ 2. 상태 데이터(State Data)**

* **구성 데이터가 아닌 추가적인 정보**
* **읽기 전용(읽기만 가능, 수정 불가능) 상태 정보 및 통계 데이터 포함**
* 예제: CPU 사용량, 메모리 사용량, 패킷 전송량, 현재 네트워크 연결 상태 등

**✅ 왜 구성 데이터와 상태 데이터를 구분해야 할까?**

만약 NETCONF에서 **구성 데이터와 상태 데이터를 구분하지 않으면** 다음과 같은 문제가 발생할 수 있습니다.

1. **구성 데이터 비교 시 불필요한 요소 포함**
   * 예) 통계 데이터(트래픽 카운터, 메모리 사용량 등)가 포함되면 구성 변경 사항을 비교하기 어려워짐.
2. **읽기 전용 데이터를 변경하려는 비논리적 요청 발생**
   * 예) CPU 사용량을 수정하려는 요청은 의미가 없음.
3. **데이터 크기 증가로 인해 성능 저하 발생**
   * 상태 데이터가 포함되면 불필요하게 큰 데이터셋을 다뤄야 함.
4. **백업 및 복원 시 복잡성 증가**
   * 예) 백업 파일에 읽기 전용 데이터가 포함되면, 복원 시 불필요한 정보까지 복원해야 할 수도 있음.

**✅ NETCONF에서의 해결 방법**

NETCONF 프로토콜은 **구성 데이터와 상태 데이터를 별도로 처리할 수 있도록 설계됨**.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

📌 **즉, 상태 데이터가 필요하지 않다면 <get-config> 만 사용하면 됨.**  
📌 **장치의 전체 상태(구성 + 상태 데이터)를 조회하려면 <get> 사용.**

**✅ 구성 데이터 외의 기타 데이터는?**

* NETCONF는 기본적으로 **장치를 원하는 상태로 만들기 위한 정보에 초점을 맞춤**.
* 하지만, **장치에 영구적으로 저장되는 기타 데이터(예: 사용자 파일, 데이터베이스 등)**는 구성 데이터에 포함할지 여부가 **제조사 또는 벤더의 구현 방식에 따라 다름**.

📌 **예제:**

* 네트워크 장치에 **사용자 인증 데이터(예: 계정 정보, 비밀번호 해시 등)가 저장된 데이터베이스**가 있다면?
  + 어떤 장치는 이를 **구성 데이터로 취급하여 NETCONF를 통해 관리 가능**하도록 할 수도 있고,
  + 어떤 장치는 이를 **구성 데이터에서 제외하고 별도의 관리 방법을 제공할 수도 있음**.

**📌 요약**

✅ **구성 데이터 (Configuration Data)**: 장치를 원하는 상태로 변경하는 데 필요한 **쓰기 가능한 설정 데이터**  
✅ **상태 데이터 (State Data)**: **읽기 전용(변경 불가)** 상태 정보 및 통계 데이터

✅ **구성 데이터와 상태 데이터를 구분하는 이유:**

* 구성 변경 비교 시 **불필요한 데이터 방지**
* **읽기 전용 데이터를 변경하려는 비논리적 요청 방지**
* **데이터 크기 증가로 인한 성능 저하 방지**
* **백업 및 복원 시 복잡성 감소**

✅ **NETCONF는 두 개의 명령어로 데이터를 조회:**

* <get-config>: **구성 데이터만 조회**
* <get>: **구성 데이터 + 상태 데이터 조회**

✅ **구성 데이터 외의 추가적인 영구 데이터(예: 사용자 DB)는 벤더(제조사) 구현에 따라 포함될 수도 있고 아닐 수도 있음.**

# 2. Transport Protocol Requirements

NETCONF는 **RPC(Remote Procedure Call) 기반** 통신 방식으로 작동합니다. 즉, **클라이언트가 RPC 요청(Request) 메시지를 보내면, 서버가 해당 요청에 대한 응답(Reply) 메시지를 반환**하는 구조입니다.

**✅ NETCONF의 전송 프로토콜 특성**

* **NETCONF는 특정한 전송 프로토콜에 종속되지 않음**
  + 다양한 전송 프로토콜 위에서 동작할 수 있도록 설계됨.
  + 특정 프로토콜이 아니라, 필요 기능을 제공하는 프로토콜이라면 어떤 것이든 사용 가능.
  + 예: SSH, TLS, BEEP, SOAP/HTTP 등
* **NETCONF는 전송 프로토콜과 독립적으로 작동하지만, 적절한 매핑이 필요함**
  + NETCONF를 특정 전송 프로토콜 위에서 구현하려면, 해당 프로토콜과 NETCONF 간의 **매핑(mapping) 방법**이 정의되어야 함.
* **전송 프로토콜은 세션 유형(Session Type)을 식별할 수 있어야 함**
  + 즉, NETCONF 계층에서 **이 통신이 클라이언트인지 서버인지 구별할 수 있도록** 전송 프로토콜이 이를 제공해야 함.

**✅ 전송 프로토콜이 충족해야 할 요구사항**

* **신뢰할 수 있는 연결 제공** (Reliable Connection)
* **보안 기능 제공 (예: 암호화, 인증, 무결성 검증 등)**
* **메시지를 순서대로 전달할 것** (Ordered Message Delivery)
* **세션을 지속적으로 유지할 것** (Persistent Session)

**📌 요약**

✔️ NETCONF는 **RPC 기반 통신**을 사용하여 클라이언트-서버 간 요청/응답을 처리함.  
✔️ **특정한 전송 프로토콜에 종속되지 않으며**, 다양한 프로토콜 위에서 구현 가능.  
✔️ 전송 프로토콜은 **세션 유형(클라이언트/서버)을 식별할 수 있어야 함.**  
✔️ 전송 프로토콜은 **신뢰성, 보안성, 순차적 메시지 전달, 지속적 세션 유지** 기능을 제공해야 함.

**쉽게 말하면:**  
NETCONF는 **SSH, TLS 같은 여러 프로토콜 위에서 동작할 수 있으며**,  
전송 프로토콜은 **안전하고 신뢰할 수 있는 연결을 보장해야 한다**는 의미입니다. 🚀

## 2.1 Connection-Oriented Operation

NETCONF는 **연결 지향적(connection-oriented) 프로토콜**이며, 이를 위해 **지속적인(persistent) 연결**이 필요합니다.

**✅ NETCONF 연결의 주요 특징**

1. **지속적인 연결 유지 (Persistent Connection)**
   * NETCONF는 단순한 요청-응답 방식이 아니라, **하나의 연결을 장시간 유지하며** 여러 개의 프로토콜 작업을 수행할 수 있도록 설계됨.
   * 단순한 HTTP 요청처럼 연결이 끊어지는 방식이 아니라, **세션(session)이 지속적으로 유지**됨.
2. **신뢰할 수 있고 순차적인 데이터 전송 보장 (Reliable & Ordered Data Delivery)**
   * 전송되는 데이터는 **순서가 보장**되어야 하며,
   * **손실 없이 신뢰할 수 있는 방식으로 전송**되어야 함.
   * 이를 위해 SSH, TLS 같은 신뢰할 수 있는 프로토콜이 사용됨.
3. **자원 자동 해제 (Automatic Resource Cleanup)**
   * 특정 연결에서 요청된 **자원(Resource)** 은 **연결이 종료되면 자동으로 해제**됨.
   * 예를 들어, 클라이언트가 어떤 자원(예: 설정 잠금)을 요청했다면,
     + 클라이언트가 명시적으로 해제하기 전까지 유지되지만,
     + 만약 연결이 끊어진다면 서버가 이를 감지하고 자동으로 자원을 정리.
4. **장애 복구(Failure Recovery) 지원**
   * 클라이언트가 **잠금(lock)**을 요청한 후,
   * 클라이언트의 연결이 끊어지면,
   * 서버는 자동으로 해당 잠금을 해제하여 다른 클라이언트가 사용할 수 있도록 조치.

**✅ 예제: 설정 잠금 (Lock) 처리 방식**

* **클라이언트 A** 가 **서버의 설정을 수정하려고 <lock> 요청**을 보냄.
* 서버는 **클라이언트 A에게 설정 변경 권한을 부여하고 잠금(lock)을 유지**함.
* 만약 **클라이언트 A의 연결이 갑자기 끊어진다면**,
  + 서버는 **이 연결이 종료된 것을 감지하고 자동으로 잠금을 해제**하여 다른 클라이언트가 설정을 변경할 수 있도록 함.

🔹 **잠금(lock) 관련 기능은 NETCONF의 <lock> 연산(Section 7.5)에서 자세히 설명됨.**

**📌 요약**

✔️ NETCONF는 **지속적인 연결(Persistent Connection)을 유지**하며 작동하는 프로토콜.  
✔️ **신뢰할 수 있는(Reliable) & 순차적인(Ordered) 데이터 전송**이 필수.  
✔️ **연결이 종료되면 서버가 관련 자원을 자동으로 해제** (예: 설정 잠금 해제).  
✔️ **장애 복구 기능을 내장**하여 클라이언트 연결이 끊어져도 서버가 안전하게 조치.

**쉽게 말하면:**  
NETCONF는 단순한 요청-응답 방식이 아니라, **세션을 유지하면서 지속적으로 설정을 관리하는 프로토콜**입니다.  
그리고 **클라이언트가 연결이 끊어지면 서버가 자동으로 정리해주는 기능도 내장**되어 있어, 보다 안전한 네트워크 관리를 지원합니다. 🚀

## 2.2 Authentication, Integrity, and Confidentiality

인증(Authentication), 무결성(Integrity), 기밀성(Confidentiality)

NETCONF 프로토콜은 **보안이 필수적인 관리 프로토콜**로, 연결 시 **인증, 데이터 무결성, 기밀성 및 재전송 공격 방지 기능**을 보장해야 합니다.

**✅ NETCONF 연결에서 보장해야 할 보안 요소**

1. **인증(Authentication)**
   * **클라이언트(Client)와 서버(Server)는 반드시 인증을 거쳐야 함.**
   * NETCONF 자체가 아닌 **전송 프로토콜(SSH, TLS 등)이 인증을 담당**.
   * 예를 들어, SSH를 사용하면 **SSH 키 또는 패스워드를 이용한 사용자 인증**이 수행됨.
2. **데이터 무결성(Integrity)**
   * 전송 중에 데이터가 **손상되거나 변조되지 않도록 보호**해야 함.
   * 이를 위해 **TLS(SSL) 또는 SSH 같은 보안 프로토콜을 사용하여 데이터 암호화**.
3. **기밀성(Confidentiality)**
   * **전송되는 데이터는 반드시 암호화**되어야 함.
   * 이를 통해 **외부 공격자가 네트워크 트래픽을 감청하지 못하도록 방지**.
   * 예를 들어, TLS를 사용하면 **패킷을 암호화하여 보호**할 수 있음.
4. **재전송 공격 방지(Replay Protection)**
   * 공격자가 이전에 전송된 메시지를 가로채고 이를 재전송하는 공격 방지.
   * 전송 프로토콜에서 **세션 토큰(session token), 타임스탬프(timestamp), 난수(nonce) 등의 기법을 사용**하여 보장.

**✅ NETCONF에서 인증(Authorization) 방식**

* NETCONF는 **운영 장비의 기존 인증 시스템(RADIUS, TACACS+)을 활용할 수 있음.**
* 예를 들어, **RADIUS 기반 인증이 가능한 장비**라면 **NETCONF도 동일한 인증을 사용**하도록 구성할 수 있음.
* **인증 후, 서버는 클라이언트의 권한을 확인**하여 허용된 작업만 수행하도록 제어함.

**✅ NETCONF 사용자 이름 (NETCONF Username)**

* 인증이 완료되면 **NETCONF 서버는 클라이언트의 ID(사용자 이름)를 확인**함.
* 이 **사용자 이름(NETCONF username)** 은 NETCONF 세션에서 사용되며, **해당 사용자의 권한을 기반으로 접근 제어가 적용됨.**
* NETCONF 사용자의 접근 권한(permissions)은 **장비의 설정(Configuration)에 따라 결정됨.**
* 예를 들어, **읽기 전용(READ-ONLY) 계정과 관리자(ADMIN) 계정은 서로 다른 권한을 가짐.**

**📌 요약**

✔️ **NETCONF 연결은 반드시 인증(Authentication), 데이터 무결성(Integrity), 기밀성(Confidentiality), 재전송 공격 방지(Replay Protection)를 제공해야 함.**  
✔️ **보안 기능은 NETCONF 자체가 아닌 전송 프로토콜(SSH, TLS 등)에서 제공**해야 함.  
✔️ **NETCONF 인증 방식은 기존 네트워크 장비의 인증 시스템(RADIUS 등)을 활용 가능.**  
✔️ **사용자 이름(NETCONF username)을 기반으로 접근 제어(Access Control)가 적용됨.**

**쉽게 말하면:**  
NETCONF는 단순한 설정 변경 프로토콜이 아니라, **보안이 철저한 프로토콜**입니다.  
**암호화(Encryption)된 연결을 유지하면서, 사용자 인증을 철저히 하고, 권한이 있는 사용자만 설정을 변경할 수 있도록 보장합니다.** 🔐

## 2.3 Mandatory Transport Protocol

NETCONF을 구현할 때 **반드시 지원해야 하는 전송 프로토콜은 SSH(Secure Shell)** 입니다.

✔ [**RFC 6242**](#_13.1_Normative_References)에서 정의된 **SSH 기반 NETCONF 전송 프로토콜**을 지원해야 합니다.  
✔ SSH는 **보안 연결(암호화)과 인증을 제공**하며, NETCONF 메시지를 안전하게 주고받을 수 있도록 함.

**📌 왜 SSH가 필수인가?**

1. **보안성(Security)**
   * SSH는 **데이터를 암호화**하여 **중간자 공격(Man-in-the-Middle, MITM) 방지**
   * **사용자 인증(User Authentication)** 지원
   * **무결성(Integrity) 보장**: 데이터 변조 방지
2. **신뢰할 수 있는 연결(Reliability)**
   * SSH는 **TCP 기반**으로 **신뢰할 수 있는 데이터 전송** 제공
   * **세션(Session) 유지 기능**으로 NETCONF 작업 수행 중 연결이 끊어지는 것을 방지
3. **기존 네트워크 환경과 호환**
   * 대부분의 네트워크 장비(Cisco, Juniper 등)는 이미 **SSH를 기본적으로 지원**
   * 추가적인 설정 없이 바로 NETCONF를 사용할 수 있음

**📌 요약**

✔ **모든 NETCONF 구현은 SSH(**[**RFC 6242**](#_13.1_Normative_References)**)를 필수적으로 지원해야 함.**  
✔ SSH는 **암호화, 인증, 무결성 보장** 등의 보안 기능을 제공하여 NETCONF 메시지를 안전하게 전송함.  
✔ **NETCONF over SSH**는 네트워크 장비에서 기본적으로 지원하는 방식이므로 **호환성이 뛰어남**.

**쉽게 말하면:**  
NETCONF를 사용할 때 **반드시 SSH를 통해 안전한 연결을 유지해야 한다는 의미!**  
다른 전송 프로토콜(예: TLS, BEEP 등)도 사용할 수 있지만, **SSH 지원은 필수 조건**입니다. 🔐

# 3. XML Considerations

NETCONF은 **XML을 메시지 인코딩 포맷으로 사용**하여 **계층적 데이터 구조**를 표현할 수 있습니다.  
이를 통해 **텍스트 기반 도구뿐만 아니라 XML 전용 도구**를 사용해 쉽게 **읽고, 저장하고, 변환 가능**합니다.

**📌 NETCONF 메시지의 필수 조건**

✅ **반드시 UTF-8 인코딩을 사용해야 함** ([RFC 3629](#_13.1_Normative_References))  
✅ **모든 메시지는 올바른 XML 형식(Well-formed XML)이어야 함**  
✅ **잘못된 XML 메시지**를 받으면 **"malformed-message" 오류를 반환해야 함**  
✅ **오류 응답이 불가능한 경우 세션 종료**해야 함  
✅ **XML 선언(Optional) 가능** (예: <?xml version="1.0" encoding="UTF-8"?>)

## 3.1 Namespace

NETCONF의 모든 프로토콜 요소는 **다음 네임스페이스를 사용함**



✔ **NETCONF의 기능(Capabilities)은 반드시 URI 형식으로 정의해야 함** ([RFC 3986](#_13.1_Normative_References))  
✔ **NETCONF 기능(Capabilities)은 8장에서 다룸**

## 3.2 Document Type Declarations

🚫 **NETCONF 메시지에서 Document Type Declaration(DTD)은 사용할 수 없음!**  
🚫 예를 들어, 다음과 같은 선언이 **금지됨**



✔ DTD는 XML의 유효성 검증을 위해 사용되지만, NETCONF에서는 **제한된 XML 구조를 유지하기 위해 허용하지 않음**

**📌 요약**

✔ NETCONF 메시지는 **UTF-8 인코딩된 올바른 XML 형식이어야 함**  
✔ XML 네임스페이스는 **urn:ietf:params:xml:ns:netconf:base:1.0** 을 사용해야 함  
✔ **DTD 선언은 금지됨** (XML 문서 구조를 제한하기 위해)  
✔ **잘못된 XML 메시지를 받으면 오류 응답을 보내고, 불가능하면 세션 종료해야 함**

**쉽게 말하면:**  
**NETCONF 메시지는 "UTF-8 XML" 형식으로만 통신하고, 반드시 올바른 XML 구조여야 하며, DTD는 금지됨!** 🚀

# 4. RPC Model

NETCONF는 **RPC(Remote Procedure Call) 기반 통신 모델**을 사용합니다.  
즉, **클라이언트가 요청을 보내면(Server에게 원격 호출) 서버가 응답을 반환하는 구조**입니다.

**📌 RPC 구조**

NETCONF에서는 **다음 두 가지 XML 요소**를 사용하여 **요청과 응답을 주고받음**

1️ <rpc> **요청(Request) 메시지**  
2️ <rpc-reply> **응답(Response) 메시지**

✔ 이를 통해 **NETCONF는 특정 전송 프로토콜(SSH, TLS 등)과 독립적으로 동작 가능**  
✔ XML을 기반으로 요청과 응답을 표준화

**📌 RPC 메시지의 XML 정의**

🔹 **NETCONF의 RPC 메시지 문법과 XML 인코딩 방식은** [**Appendix B**](#_Appendix_B._)**에서 XML Schema로 정의**됨.  
🔹 이를 통해 **데이터 구조를 표준화하고, 상호운용성을 보장**할 수 있음.

**📌 요약**

✔ NETCONF는 **RPC 기반의 요청-응답 방식**을 사용함  
✔ <rpc> 요청과 <rpc-reply> 응답을 통해 클라이언트와 서버가 통신  
✔ **전송 프로토콜과 독립적으로 동작 가능**  
✔ RPC 메시지의 XML 형식은 [**Appendix B**](#_Appendix_B._)**의 XML Schema에서 정의됨**

**쉽게 말하면:**  
NETCONF는 **"요청-응답(RPC)" 방식**으로 작동하고, 요청과 응답을 **XML로 표준화**해서 주고받음 🚀

## 4.1 <rpc> Element

**📌 1. <rpc> 개요**

✅ <rpc> 요소는 **클라이언트 → 서버로 전송하는 NETCONF 요청을 감싸는 요소**  
✅ **반드시 포함해야 하는 속성:** "message-id"  
✅ 요청된 **메서드 이름과 매개변수(parameter)는 <rpc> 내부에서 XML 형식으로 표현**

**📌 2. message-id 속성**

🔹 "message-id"는 **요청을 식별하는 문자열** (일반적으로 증가하는 숫자로 설정됨)  
🔹 서버는 이 값을 해석하지 않고, 그대로 **응답 <rpc-reply>의 "message-id"로 반환**  
🔹 XML의 속성 값은 [W3C XML 표준](https://www.w3.org/TR/2000/REC-xml-20001006/)에 따라 **정규화(normalization)** 필요

📌 **예제:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**📌 3. <rpc>의 요청 구조**

📌 **기본 구조**  
1️ **메서드 이름** = <rpc> 내부 첫 번째 요소  
2️ **매개변수 (parameter)** = 메서드 내부의 하위 요소

**📌 4. 예제**

✅ **예제 1: <my-own-method> 호출 (파라미터 포함)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🔹 **메서드명:** <my-own-method>  
🔹 **매개변수:**

* <my-first-parameter> = 14
* <another-parameter> = "fred"

✅ **예제 2: <rock-the-house> 호출 (파라미터 포함)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🔹 **메서드명:** <rock-the-house>  
🔹 **매개변수:**

* <zip-code> = "27606-0100"

✅ **예제 3: <get> 호출 (매개변수 없음)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🔹 **메서드명:** <get> (매개변수 없음)

**📌 요약**

✔ **<rpc>는 NETCONF 요청을 감싸는 XML 요소**  
✔ **반드시 message-id 속성을 포함해야 함**  
✔ **요청 메서드는 <rpc> 내부에 XML 요소로 포함됨**  
✔ **매개변수는 메서드 요소 내부에서 XML 태그로 정의**  
✔ **서버는 요청을 처리한 후 <rpc-reply>로 응답** 🚀

## 4.2 <rpc-reply> Element

**📌 1. <rpc-reply> 개요**

✅ <rpc-reply> 요소는 **서버 → 클라이언트로 응답을 보낼 때 사용**  
✅ 요청했던 <rpc>와 **동일한 "message-id" 속성 값을 반환**  
✅ <rpc>에 포함된 **추가 속성(attributes)이 있으면 그대로 반환해야 함**  
✅ 응답 데이터는 <rpc-reply> 내부의 **자식 요소**로 포함됨

**📌 2. message-id 속성**

🔹 **반드시 포함해야 하는 속성:** "message-id"  
🔹 **클라이언트의 <rpc> 요청과 동일한 message-id 값을 반환해야 함**

**📌 3. 예제**

✅ **예제 1: <get> 요청 및 응답 (추가 속성 포함)**  
📌 **요청 (<rpc>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **응답 (<rpc-reply>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🔹 **클라이언트가 <get> 요청을 보냄**  
🔹 **"user-id"라는 추가 속성을 포함 → 응답에도 그대로 반환됨**  
🔹 **서버는 <data> 태그 안에 요청된 정보를 포함하여 응답**

**📌 4. 주요 규칙 정리**

✔ **<rpc-reply>는 <rpc> 요청에 대한 응답**  
✔ **"message-id" 속성은 요청과 동일하게 설정**  
✔ **요청에 포함된 추가 속성(attributes)은 그대로 반환해야 함**  
✔ **응답 데이터는 <rpc-reply>의 하위 요소로 제공됨** 🚀

## 4.3 <rpc-error> Element

**📌 1. <rpc-error> 개요**

✅ **클라이언트의 <rpc> 요청 처리 중 오류가 발생하면, <rpc-reply>에 <rpc-error> 요소를 포함하여 응답**  
✅ 하나의 <rpc-reply> 안에 **여러 개의 <rpc-error> 요소를 포함할 수도 있음**  
✅ 서버는 오류를 하나만 보고할 수도 있고, 여러 개를 동시에 보고할 수도 있음  
✅ 클라이언트가 **접근 권한이 없는 데이터 모델 관련 정보는 제공하면 안 됨**

**📌 2. <rpc-error> 내부 요소 설명**

🔹 **<error-type>**: 오류가 발생한 계층을 정의

* transport (보안 전송 계층, 예: SSH, TLS)
* rpc (메시지 계층)
* protocol (프로토콜 계층)
* application (애플리케이션/콘텐츠 계층)

🔹 **<error-tag>**: 오류 상태를 나타내는 문자열

* 예: missing-attribute, invalid-value, operation-failed
* 전체 목록은 [**Appendix A**](#_Appendix_A._NETCONF) **(표준 NETCONF 오류 목록)** 참조

🔹 **<error-severity>**: 오류 심각도

* error: 치명적 오류
* warning: 경고 (현재 정의된 <error-tag> 중 사용 사례 없음, 향후 사용 가능)

🔹 **<error-app-tag>** (선택 사항): 데이터 모델 또는 특정 구현에서 정의한 오류 코드

🔹 **<error-path>** (선택 사항): 오류 발생 위치를 XPath 표현식으로 지정

🔹 **<error-message>** (선택 사항): 사람이 이해할 수 있도록 설명하는 메시지

* "xml:lang" 속성을 포함 가능 (예: <error-message xml:lang="en">Invalid IP address</error-message>)

🔹 **<error-info>** (선택 사항): 추가적인 오류 세부 정보 제공

* 예: <bad-attribute>, <bad-element> 등

**📌 3. 예제**

✅ **예제 1: message-id 속성 누락 오류**  
📌 **클라이언트 요청 (<rpc>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **서버 응답 (<rpc-reply>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🔹 <rpc> 요청에서 "message-id" 속성이 빠져 있음 → **missing-attribute 오류 발생**  
🔹 error-info에서 "bad-attribute"와 "bad-element"를 포함하여 문제 요소를 명확히 전달

✅ **예제 2: MTU 값 범위 오류 및 잘못된 IP 주소 오류 (다중 오류 포함)**  
📌 **서버 응답 (<rpc-reply>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🔹 첫 번째 <rpc-error>: **MTU 값이 256~9192 범위를 벗어남**  
🔹 두 번째 <rpc-error>: **Ethernet1/0 인터페이스에 잘못된 IP 주소 입력**  
🔹 <error-path>를 사용해 정확한 오류 발생 위치를 XPath로 표현

**📌 4. 주요 규칙 정리**

✔ **오류 발생 시 <rpc-reply> 내부에 <rpc-error>를 포함해야 함**  
✔ **여러 개의 오류가 발생하면, 하나의 <rpc-reply>에 여러 <rpc-error>를 포함할 수 있음**  
✔ **error-type, error-tag, error-severity는 필수 요소**  
✔ **필요한 경우 error-message, error-path, error-info를 포함하여 추가 정보 제공 가능** 🚀

## 4.4 <ok> Element

**📌 1. <ok> 개요**

✅ **NETCONF <rpc-reply> 메시지에서 오류 없이 성공적으로 요청이 처리되었음을 나타냄**  
✅ **데이터를 반환할 필요가 없는 경우 사용됨**  
✅ 클라이언트가 <ok> 응답을 받으면 **해당 RPC 요청이 성공적으로 완료되었음을 의미**

**📌 2. <ok> 사용 예제**

✅ **예제 1: edit-config 요청 성공 응답**  
📌 **클라이언트 요청 (<rpc>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **서버 응답 (<rpc-reply>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🔹 클라이언트가 **MTU 값을 1500으로 설정하는 edit-config 요청을 보냄**  
🔹 서버가 해당 요청을 성공적으로 적용했으며 **오류 없이 완료되었음을 <ok> 응답으로 전달**

✅ **예제 2: delete-config 요청 성공 응답**  
📌 **클라이언트 요청 (<rpc>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **서버 응답 (<rpc-reply>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🔹 클라이언트가 **candidate 데이터스토어를 삭제하는 delete-config 요청을 보냄**  
🔹 **요청이 성공적으로 실행되었으며, 반환할 데이터가 없으므로 <ok>를 응답**

**📌 3. <ok> 응답을 받았을 때 클라이언트의 해석**

✔ <ok> 응답을 받으면 **해당 요청이 성공적으로 처리되었음을 의미**  
✔ 클라이언트는 **추가적인 오류 확인 없이 다음 작업을 수행 가능**  
✔ **데이터를 반환하는 요청(get, get-config)에서는 <ok>가 사용되지 않음** 🚀

## 4.5 Pipelining

**📌 1. NETCONF Pipelining 개요**

✅ **NETCONF에서 <rpc> 요청을 직렬(Serial)로 처리해야 함**  
✅ **클라이언트는 이전 요청이 완료되기 전에 추가 <rpc> 요청을 보낼 수 있음**  
✅ **서버는 <rpc> 요청을 받은 순서대로 응답을 반환해야 함**

**📌 2. Pipelining 동작 방식**

🔹 클라이언트가 여러 개의 <rpc> 요청을 빠르게 보낼 수 있음 (비동기 전송 가능)  
🔹 하지만 **서버는 요청을 순차적으로(직렬) 처리하고, 응답도 요청 순서대로 반환**  
🔹 따라서 응답을 병렬로 처리하지 않으며, **FIFO(First In, First Out) 방식**을 따름

**📌 3. Pipelining 예제**

📌 **클라이언트가 두 개의 <rpc> 요청을 연속적으로 보냄**

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

📌 **서버는 요청 순서대로 응답을 반환**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**📌 4. Pipelining에서 클라이언트 고려 사항**

✔ **응답 순서가 보장되므로, 클라이언트는 message-id로 응답을 추적 가능**  
✔ **대기 시간(Latency)이 줄어들어 성능이 향상될 수 있음**  
✔ **하지만 서버는 요청을 병렬로 처리하지 않으므로, 요청 처리 시간이 길면 응답 지연 발생 가능**

📢 **결론:**  
NETCONF Pipelining을 사용하면 클라이언트가 요청을 빠르게 보낼 수 있지만,  
서버는 **순차적으로 응답을 반환하므로 비효율적인 요청 설계는 응답 지연을 초래할 수 있음** 🚀

# 5. Configuration Model

**📌 1. NETCONF 구성 모델 개요**

✅ NETCONF는 **기본 운영(Operation) 집합과 확장 가능한 기능(Capability) 제공**  
✅ **세션 초기화 시 장치 간에 지원하는 기능(기능 목록)을 교환**  
✅ **기능 교환 과정은 8.1절에서 설명됨**

**📌 2. NETCONF의 주요 특징**

🔹 **구성(Configuration) 데이터와 상태(State) 데이터 분리**  
🔹 **구성 변경이 가능하며, 트랜잭션 기반의 원자적(Atomic) 실행 지원**  
🔹 **확장 가능한 기능(Capability)을 통해 다양한 벤더 지원 가능**

**📌 3. 구성 모델 예제**

📌 **구성 데이터를 가져오는 요청 (<get-config>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **서버의 응답 (<rpc-reply>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **구성을 변경하는 요청 (<edit-config>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **서버의 응답 (<ok>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**📌 4. NETCONF 구성 모델의 장점**

✔ **표준화된 방식으로 네트워크 장치의 설정 관리 가능**  
✔ **구성 데이터와 상태 데이터를 분리하여 일관된 관리 지원**  
✔ **원자적(Atomic) 트랜잭션 지원으로 일관성 보장**  
✔ **확장 기능(Capability) 교환을 통해 다양한 벤더 장치 간 호환 가능**

📢 **결론:**  
NETCONF는 **구성 변경, 상태 조회, 트랜잭션 처리** 등을 지원하는 강력한 네트워크 관리 프로토콜 💡🚀

## 5.1 Configuration Datastores

**📌 1. Configuration Datastore 개요**

✅ **NETCONF는 하나 이상의 구성 데이터 저장소(Datastore) 지원**  
✅ **구성 데이터 저장소란?**

* **초기 상태 → 원하는 동작 상태로 변경하기 위한 모든 구성 데이터 포함**
* **운영 명령(Executive Commands)이나 상태 데이터(State Data)는 포함되지 않음**

📌 2. 주요 구성 데이터 저장소 (Datastore Types)

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**📌 3. Running Configuration (<running>) 예제**

📌 **현재 실행 중인 구성 데이터를 가져오기 (<get-config>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **서버의 응답 (<rpc-reply>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **Running Configuration을 변경하기 (<edit-config>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **서버의 응답 (<ok>)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**📌 4. 추가 데이터 저장소 (Candidate & Startup)**

🔹 **Candidate (<candidate>)** → 변경 사항을 미리 저장하고 <commit> 명령으로 적용  
🔹 **Startup (<startup>)** → 부팅 시 적용될 초기 설정 관리

📌 **이 두 데이터 저장소는 기본 NETCONF 모델에는 포함되지 않으며, 추가 기능(Capability) 필요!**  
➡️ **Candidate 기능은 8.3절, Startup 기능은 8.7절에서 설명됨**

**📌 5. Configuration Datastore의 핵심 개념**

✔ **<running> 은 항상 존재하고 실시간으로 적용되는 구성 저장소**  
✔ **추가 기능(candidate, startup)을 사용하면 더 강력한 구성 관리 가능**  
✔ **구성 변경 시 <edit-config> 를 사용하여 적용 가능**

📢 **결론:**  
NETCONF의 구성 데이터 저장소는 네트워크 장치의 **구성 데이터 관리**를 체계적으로 할 수 있도록 지원! 🚀

## 5.2 Data Modeling

**📌 1. NETCONF에서 데이터 모델의 역할**

✅ **NETCONF 프로토콜 자체는 데이터 모델을 정의하지 않음**  
✅ **NETCONF는 구성 데이터를 <config> 요소 안에 포함하여 전송**  
✅ **장치는 "Capabilities" (기능 광고)를 통해 지원하는 데이터 모델을 선언**  
✅ **NETCONF는 데이터 모델을 "불투명한(opaque) 데이터"로 취급**  
➡️ **즉, 데이터 모델의 구조나 제약 사항을 NETCONF 자체가 해석하지 않음**

**📌 2. 데이터 모델링 관련 핵심 개념**

📌 **데이터 모델을 잘 알고 있어야 NETCONF를 올바르게 사용할 수 있음**  
📌 **장치와 관리 시스템이 동일한 데이터 모델을 이해하고 있어야 함**  
📌 **데이터 모델에는 다음과 같은 정보가 포함될 수 있음**

* **데이터 구조 (Layout)**
* **데이터 포함 관계 (Containment)**
* **키 및 조회 방식 (Keying & Lookup)**
* **데이터 변경 규칙 (Replacement & Management)**
* **기타 제약 조건 (Constraints, Validation Rules)**

**📌 3. 데이터 모델과 Capabilities**

🔹 **장치가 지원하는 데이터 모델은 "Capabilities”를 통해 광고됨**  
🔹 **예를 들어, NETCONF는 표준 모델(YANG)과 사유 모델(벤더별 XML Schema)을 모두 지원 가능**  
🔹 **데이터 모델을 지원하는 기능들은 <hello> 메시지를 통해 초기 교환됨**

📌 **예제: 장치가 YANG 데이터 모델을 지원한다고 광고하는 <hello> 메시지**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡️ **이 메시지를 통해 관리 시스템은 해당 장치가 "ietf-interfaces" YANG 모델을 지원하는 것을 알 수 있음**

**📌 4. 데이터 모델의 예시 (YANG 모델 적용 예시)**

📌 **YANG 데이터 모델 예제 (ietf-interfaces.yang)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **NETCONF에서 해당 YANG 모델을 사용하여 구성 데이터를 주고받는 방식**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 **이 메시지를 해석할 수 있는 이유는?**  
➡️ **장치와 클라이언트가 "ietf-interfaces" YANG 모델을 지원하고 있기 때문!**

**📌 5. 데이터 모델링의 핵심 정리**

✔ **NETCONF는 데이터 모델을 정의하지 않지만, 특정 데이터 모델을 다룰 수 있도록 설계됨**  
✔ **장치가 지원하는 데이터 모델을 "Capabilities"를 통해 광고하고 관리 시스템이 이를 활용**  
✔ **표준 데이터 모델 (예: YANG)과 벤더별 데이터 모델(XML Schema) 모두 사용 가능**  
✔ **데이터 모델을 이해해야 올바르게 NETCONF 요청을 생성하고 장치를 제어할 수 있음**

📢 **결론:**  
NETCONF는 **데이터 모델을 기반으로 동작**하며, YANG 같은 모델을 통해 **효율적인 네트워크 장치 관리**를 가능하게 함! 🚀

# 6. Subtree Filtering

## 6.1 Overview

**📌 1. Subtree Filtering 이란?**

✅ XML 서브트리 필터링은 **NETCONF의 <get> 및 <get-config> 요청에서 특정 XML 서브트리만 응답에 포함하도록 선택하는 기능**  
✅ **서버가 데이터 모델에 대한 특정한 의미론(Semantics)을 해석할 필요 없음** → **단순하고 중앙 집중적인 구현 가능**  
✅ **포함(inclusion), 정확한 값 일치(exact-match), 노드 선택(selection) 등의 기본적인 필터링 기능 제공**  
✅ **필터의 각 노드는 "포함 필터(inclusive filter)" 역할**

* 지정된 노드 및 그 하위 트리만 응답에 포함
* 요청 필터에 존재하지 않는 데이터는 응답에 포함되지 않음

**📌 2. Subtree Filtering 개념**

✅ **서브트리 필터는 0개 이상의 XML 요소로 구성**  
✅ **필터에서 지정된 경로와 일치하는 데이터만 반환됨**  
✅ **필터에서 지정한 특정한 leaf 노드는 응답에서 전체 서브트리로 확장될 수도 있음**

➡ **즉, 클라이언트는 필요한 데이터만 요청하고, 서버는 불필요한 데이터를 보내지 않음**

**📌 3. Subtree Filtering 예제**

**✅ 전체 데이터 요청 (<get-config> without filtering)**

클라이언트가 **인터페이스 정보 전체**를 요청하는 경우:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **서버는 전체 인터페이스 정보를 반환** (불필요한 데이터 포함 가능)

**✅ 특정 노드만 요청 (<get-config> with filtering)**

예를 들어, 클라이언트가 **GigabitEthernet0/1의 MTU 값만 요청**하는 경우:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **불필요한 데이터(예: enabled, address)는 응답에 포함되지 않음**

**✅ 값 일치 필터링 (Exact Match)**

**특정한 값을 가진 데이터만 요청**하는 경우, 예를 들어 **MTU 값이 9000인 인터페이스만 반환**하도록 요청:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **MTU 값이 9000인 GigabitEthernet0/2 인터페이스만 반환됨**

**📌 4. Subtree Filtering의 장점**

✅ **불필요한 데이터 제거 → 네트워크 트래픽 절감**  
✅ **단순한 XML 구조 유지 → 데이터 가독성 향상**  
✅ **서버가 복잡한 데이터 모델을 해석할 필요 없음 → 구현 용이**

📌 **5. Subtree Filtering 요약**

텍스트, 스크린샷, 폰트, 영수증이(가) 표시된 사진

자동 생성된 설명

📢 **결론:**  
Subtree Filtering을 사용하면 **네트워크 장치에서 필요한 데이터만 효율적으로 가져올 수 있어** NETCONF 운영을 최적화할 수 있음! 🚀

## 6.2 Subtree Filter Components

NETCONF의 **Subtree Filtering**은 **다섯 가지 주요 구성 요소**로 이루어져 있음:  
1️ **Namespace Selection (네임스페이스 선택)**  
2️ **Attribute Match Expressions (속성 일치 표현식)**  
3️ **Containment Nodes (포함 노드)**  
4️ **Selection Nodes (선택 노드)**  
5️ **Content Match Nodes (내용 일치 노드)**

각 요소가 **필터링 방식에 어떻게 작용하는지** 살펴보자. 👇

**1️ Namespace Selection (네임스페이스 선택)**

✅ NETCONF 메시지는 **여러 XML 네임스페이스를 사용할 수 있음**  
✅ 네임스페이스를 지정하지 않으면 **필터가 의도한 데이터 모델에 적용되지 않을 수 있음**  
✅ 네임스페이스는 <filter> 태그 내부에서 XML xmlns 속성을 사용하여 선언됨

**📌 예제: 네임스페이스를 지정한 필터**

**인터페이스 정보를 포함하는 네임스페이스를 필터에 명시**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **네임스페이스가 정확히 일치하는 데이터만 응답됨**

**2️ Attribute Match Expressions (속성 일치 표현식)**

✅ XML 속성 값을 기준으로 필터링 가능  
✅ 특정 속성을 가진 노드만 포함

**📌 예제: 특정 속성을 가진 인터페이스만 요청**

**운영 상태(enabled="true")인 인터페이스만 반환 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **enabled="false"인 인터페이스는 응답에서 제외됨**

**3️ Containment Nodes (포함 노드)**

✅ 데이터 구조 내에서 특정 **상위 요소(Parent Node)**를 포함한 상태에서 필터링 가능  
✅ 필터가 특정 부모 요소를 지정하면, 그 하위 데이터만 반환됨

**📌 예제: 특정 인터페이스의 서브트리 요청**

**"GigabitEthernet0/1"의 모든 서브트리 데이터를 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **GigabitEthernet0/1 인터페이스와 그 하위 노드만 포함됨**

**4️ Selection Nodes (선택 노드)**

✅ 특정한 노드만 선택하여 반환 가능  
✅ 포함(Containment)과 다르게, 하위 데이터를 전부 포함하지 않을 수도 있음

**📌 예제: 인터페이스 이름만 반환 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **이름(name) 정보만 반환됨** (MTU, 상태 정보 등은 제외)

**5️ Content Match Nodes (내용 일치 노드)**

✅ 특정 값과 일치하는 데이터를 포함하는 노드만 선택 가능  
✅ 값이 다르면 해당 노드는 필터링됨

**📌 예제: 특정 MTU 값을 가진 인터페이스 요청**

**MTU 값이 1500인 인터페이스만 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **MTU 값이 1500이 아닌 인터페이스는 응답에서 제외됨**

📌 정리: Subtree Filtering 핵심 요소

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

✅ **필요한 데이터만 효율적으로 가져올 수 있어 NETCONF 최적화 가능! 🚀**

### 6.2.1 Namespace Selection

✅ **NETCONF의 서브트리 필터링에서 네임스페이스(namespace)는 매우 중요!**  
✅ **필터가 적용되는 데이터 모델과 정확히 일치해야 올바른 데이터가 반환됨**

**📌 핵심 개념**

1️ **네임스페이스가 일치해야 필터가 적용됨**

* <filter> 내부의 XML 요소가 특정 네임스페이스를 가져야 해당 데이터만 필터링 됨
* 네임스페이스가 다르면 필터가 무시될 수 있음

2️ **네임스페이스만으로 필터링 불가**

* **적어도 하나 이상의 요소(element)가 필터 내에 존재해야 함**
* 네임스페이스만 지정하면 오류 발생 가능

3️ **네임스페이스 와일드카드 지원**

* <filter> 내부의 요소에 네임스페이스를 명시하지 않으면 **서버가 모든 네임스페이스를 평가함**
* 단, **XML 속성(attribute)에는 와일드카드 적용 불가**

4️ **XML Prefix는 비교 시 무시됨**

* 네임스페이스 비교는 **URI 값으로 이루어짐** (예: http://example.com/schema/1.2/config)
* XML 내에서 사용되는 prefix(예: ex:top에서 ex)는 비교 시 고려되지 않음

**📌 예제: 네임스페이스를 지정한 필터**

**🔹 특정 네임스페이스의 데이터만 가져오기**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **"**[**http://example.com/schema/1.2/config**](http://example.com/schema/1.2/config)**" 네임스페이스를 가진 <top> 노드만 반환됨**  
✅ **해당 네임스페이스를 가지지 않은 다른 <top> 노드는 무시됨**

🔹 네임스페이스를 생략한 경우 (와일드카드 적용)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **모든 네임스페이스에서 <top> 노드를 검색하여 반환함**  
✅ **네임스페이스를 지정하지 않으면 와일드카드 방식으로 동작**

**⚠ 주의할 점**

**❌ 네임스페이스만 선언하고 요소가 없으면 필터가 적용되지 않음**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**➡ 🚨 오류 발생: <filter> 내부에 적어도 하나의 요소가 필요함!**

**✅ 올바른 방식 (최소한 하나의 요소 포함)**

**텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명**

**📌 정리: Namespace Selection의 핵심 포인트**

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

➡ **정확한 네임스페이스를 설정하면 원하는 데이터만 효율적으로 가져올 수 있음! 🚀**

### 6.2.2 Attribute Match Expressions

✅ **NETCONF 서브트리 필터에서 특정 속성(attribute)을 기준으로 데이터를 필터링하는 기능**  
✅ **필터에 명시된 속성과 값이 정확히 일치하는 데이터만 포함됨**  
✅ **속성이 정의되지 않은 요소는 선택되지 않음**

**📌 핵심 개념**

1️ **속성 기반 필터링**

* 특정 **XML 속성(attribute) 값과 일치하는 데이터만 선택**
* 요소 내부 값이 아닌, 요소의 **속성값(attribute value)**을 기준으로 필터링

2️ **속성은 여러 개 설정 가능**

* 하나 이상의 속성을 지정할 수 있으며, **모든 속성이 일치해야 데이터가 선택됨**

3️ **네임스페이스와 함께 사용 가능**

* 속성이 포함된 요소가 특정 네임스페이스를 가지는 경우, 필터에서도 해당 네임스페이스를 지정해야 함

4️ **속성이 정의되지 않은 경우 선택되지 않음**

* 특정 속성을 포함하지 않는 요소는 필터에서 자동으로 제외됨

**📌 예제: 속성 매칭을 활용한 필터링**

**🔹 특정 인터페이스(ifName="eth0")만 가져오기**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **ifName="eth0"을 가진 인터페이스만 반환됨**  
✅ **다른 인터페이스(예: eth1, wlan0 등)는 응답에 포함되지 않음**

🔹 여러 속성을 사용한 필터링

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **ifName="eth0"이면서 state="up"인 인터페이스만 선택됨**  
✅ **속성 중 하나라도 불일치하면 해당 인터페이스는 응답에 포함되지 않음**

🔹 네임스페이스가 다른 경우 필터 미적용

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

❌ **잘못된 네임스페이스를 사용했기 때문에 아무 데이터도 반환되지 않음!**  
✅ **정확한 네임스페이스(http://example.com/schema/1.2/config)를 사용해야 함**

**⚠ 주의할 점**

❌ **속성이 없는 경우 필터링에서 제외됨**



➡ "ifName" 속성을 가지지 않은 <interface> 요소는 응답에 포함되지 않음 🚨

✅ **모든 인터페이스를 가져오려면 속성을 제외해야 함**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

📌 정리: Attribute Match Expressions의 핵심 포인트

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

➡ **속성 필터를 활용하면 원하는 데이터를 더 정확하게 가져올 수 있음! 🚀**

### 6.2.3 Containment Nodes

✅ **NETCONF 서브트리 필터에서 특정 구조의 데이터 계층을 유지하면서 필터링하는 방식**  
✅ **부모 요소(Containment Node) 아래의 모든 자식 요소가 포함됨**  
✅ **부모 요소(Containment Node) 자체는 필터링에서 제거되지 않음**

**📌 핵심 개념**

1️ **Containment Node란?**

* 다른 요소(자식 노드)를 포함하는 부모 노드
* 자식 노드를 정의하지 않으면 부모 노드의 모든 하위 요소가 반환됨
* **필터에서 계층 구조를 유지하는 역할**

2️ **네임스페이스(namespace), 요소 계층 구조, 속성 일치 여부가 중요**

* 올바른 네임스페이스가 적용되어야 함
* 서브트리 필터에 명시된 계층 구조와 정확히 일치하는 데이터만 포함됨

3️ **Containment Node는 여러 개 중첩될 수 있음**

* 즉, 계층 구조를 따라 원하는 데이터만 추출 가능

**📌 예제: Containment Node를 활용한 필터링**

**🔹 특정 상위 노드(top) 아래 모든 users 정보 가져오기**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **users 노드를 포함한 모든 사용자 정보가 반환됨**  
✅ **필터에서 users 요소에 자식 요소를 명시하지 않았기 때문에 모든 데이터가 포함됨**

🔹 특정 사용자의 정보(name=admin)만 가져오기

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **name=admin을 가진 사용자 정보만 반환됨**  
✅ **불필요한 다른 사용자 정보(guest 등)는 제외됨**

🔹 네임스페이스를 잘못 지정한 경우 (잘못된 필터)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

❌ **잘못된 네임스페이스를 사용했기 때문에 응답이 비어 있음!**  
✅ **올바른 네임스페이스(http://example.com/schema/1.2/config)를 사용해야 함**

**⚠ 주의할 점**

❌ **Containment Node를 지정하지 않으면 계층이 유지되지 않음**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🚨 user 노드가 어디에 속해 있는지 모호하므로 적용되지 않을 수 있음!

✅ **Containment Node를 사용해 정확한 계층 구조 유지**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **이렇게 해야 올바르게 작동함!**

📌 정리: Containment Nodes의 핵심 포인트

텍스트, 스크린샷, 폰트, 영수증이(가) 표시된 사진

자동 생성된 설명

➡ **Containment Node를 활용하면 원하는 데이터 계층을 정확하게 유지하면서 필터링 가능! 🚀**

### 6.2.4 Selection Nodes

✅ **선택 노드는 필터링 과정에서 특정 노드만 선택하도록 제한하는 역할**  
✅ **빈 리프(leaf) 노드로 사용되며, 명시적으로 해당 노드만 반환하도록 함**  
✅ **형식: <node/> 또는 <node></node> (내용 없이 선언)**  
✅ **동일한 부모(형제 노드)에서 특정 노드만 선택할 때 사용됨**

**📌 핵심 개념**

**1️ Selection Node란?**

* **데이터 모델에서 특정 노드만 선택하도록 제한하는 필터 방식**
* **빈 노드(Empty Leaf Node)로 선언**하여 해당 노드만 필터링 됨
* **선택 노드가 포함되면, 같은 부모를 가진 다른 형제 노드는 자동 포함되지 않음**

**2️ Containment Node와의 차이점**

* **Containment Node**: 자식 노드를 포함하는 부모 요소 (계층 구조 유지)
* **Selection Node**: 해당 노드만 선택하도록 필터링 (형제 노드 자동 포함 X)

**📌 예제: Selection Node를 활용한 필터링**

**🔹 특정 그룹(users)의 데이터만 가져오기**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **users 노드만 선택됨, interfaces, settings 등 다른 노드는 자동 포함되지 않음**

🔹 특정 사용자(admin)의 데이터만 가져오기

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **admin 사용자 정보만 선택됨**  
✅ **다른 사용자(guest 등)는 자동 제외됨**

🔹 특정 인터페이스(eth0)의 데이터만 가져오기

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **eth0 인터페이스 정보만 선택됨**  
✅ **다른 인터페이스(eth1, wlan0 등)는 자동 제외됨**

**⚠ 주의할 점**

❌ **Selection Node를 사용하지 않으면 형제 노드가 자동 포함됨**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🚨 user 노드를 비워둔 경우, **모든 사용자 정보가 반환됨!**  
✅ 특정 데이터를 선택하려면 **명확한 Selection Node를 지정해야 함**

✅ **올바른 Selection Node 사용 예제**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

➡ **이렇게 해야 admin 사용자만 선택됨!**

📌 정리: Selection Nodes의 핵심 포인트

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

➡ **Selection Node를 활용하면 원하는 데이터만 정확하게 선택 가능! 🚀**

### 6.2.5 Content Match Nodes

✅ **특정 값을 가진 데이터만 선택하는 필터링 기법**  
✅ **단순 텍스트 값을 포함하는 리프(leaf) 노드에서 사용됨**  
✅ **형식: <노드>값</노드> → 해당 값과 정확히 일치하는 데이터만 반환**  
✅ **여러 개의 Content Match Node가 있을 경우, AND 조건으로 평가됨**

**📌 핵심 개념**

**1️⃣ Content Match Node란?**

* **특정 값을 가진 데이터만 필터링**
* **단순 텍스트(leaf 노드)만 포함, 하위 요소 포함 불가**
* **필터링 시 정확히 일치하는 값만 선택**
* **여러 개의 Content Match Node가 있을 경우 "AND" 조건 적용**

**📌 예제: Content Match Nodes 활용**

**🔹 특정 사용자(fred)의 데이터만 가져오기**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **이름이 "fred"인 사용자 정보만 반환됨**  
✅ **다른 사용자(alice, bob) 정보는 자동 제외됨**

🔹 특정 사용자(fred) + 특정 역할(admin)을 가진 데이터만 가져오기

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **이름이 "fred"이고 역할이 "admin"인 데이터만 반환됨**  
✅ **"fred"지만 역할이 "user"인 데이터는 제외됨**  
✅ **"admin"이지만 이름이 "fred"가 아닌 데이터도 제외됨**  
✅ **여러 개의 Content Match Nodes가 있을 경우 "AND" 조건 적용됨**

🔹 특정 네트워크 인터페이스(eth0)의 IP 주소 가져오기

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

💡 **서버 응답:**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **ifName=eth0인 인터페이스 정보만 반환됨**  
✅ **다른 인터페이스(eth1, wlan0) 정보는 자동 제외됨**

**⚠ 주의할 점**

❌ **Content Match Node 내부에 하위 요소를 포함할 수 없음**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🚨 **리프(leaf) 노드 안에는 단순한 텍스트 값만 포함 가능**  
✅ **올바른 예제**

텍스트, 스크린샷, 시계, 폰트이(가) 표시된 사진

자동 생성된 설명

❌ **빈 Content Match Node는 Selection Node로 해석됨**



✅ **Content Match Node로 사용하려면 반드시 텍스트 값을 포함해야 함**



❌ **여러 개의 Content Match Node는 "OR"이 아닌 "AND" 조건으로 동작**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

🚨 **이름이 "fred"이면서 역할이 "admin"인 데이터만 반환됨**  
🚨 **"fred"이지만 역할이 "user"이면 제외됨**  
🚨 **"admin"이지만 이름이 "fred"가 아니면 제외됨**

📌 정리: Content Match Nodes의 핵심 포인트

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

➡ **Content Match Node를 활용하면 원하는 값과 정확히 일치하는 데이터만 필터링 가능! 🚀**

## 6.3 Subtree Filter Processing

✅ **서브트리 필터는 특정 데이터 모델의 일부를 선택하여 반환하는 방식**  
✅ **필터는 여러 개의 데이터 모델 조각(fragment)을 포함할 수 있음**  
✅ **서버는 필터에서 지정한 요소와 내부 데이터 모델을 비교하여 처리**  
✅ **각 노드는 "형제 세트(sibling set)" 단위로 처리됨**  
✅ **루트에서 리프까지 순차적으로 필터 적용**

**🔹 서브트리 필터 처리 순서**

1️ **초기 상태** → **필터 출력은 비어 있음**  
2️ **서브트리 필터를 구성하는 데이터 모델 조각 확인**  
3️ **서버가 지원하는 내부 데이터 모델과 비교**  
4️ **필터가 지정한 요소가 데이터 모델에 정확히 존재하면 해당 노드 및 하위 노드 포함**  
5️ **같은 부모를 가진 형제 노드(sibling set) 단위로 필터 처리 진행**  
6️ **각 노드 유형별로 포함/제외 결정 (containment, selection, attribute match, content match 등 적용)**  
7️ **선택된 노드의 하위 노드에 대해 재귀적으로 같은 과정 수행**  
8️ **모든 서브트리 필터가 처리될 때까지 반복**

**🔹 서브트리 필터 처리 예제**

**🔸 예제 1: 모든 사용자 목록 가져오기**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**🔹 처리 과정**

1️ **필터에서 <users> 요소 요청**  
2️ **서버는 내부 데이터 모델에서 <users> 노드 확인**  
3️ **모든 <user> 하위 노드를 포함한 상태로 반환**

🔹 서버 응답

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **모든 사용자 정보가 반환됨**

🔸 예제 2: 특정 사용자(fred)의 데이터만 가져오기

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**🔹 처리 과정**

1️ **서버는 <users> 노드를 확인**  
2️ **하위 노드 <user> 중 <name>이 fred인 데이터만 선택**  
3️ **선택된 노드의 모든 형제 요소(예: <role> 등) 포함하여 반환**

**🔹 서버 응답**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **이름이 "fred"인 사용자만 반환됨**  
✅ **"alice" 등의 다른 사용자는 제외됨**

🔸 예제 3: 특정 네트워크 인터페이스(eth0)의 IP 주소 가져오기

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**🔹 처리 과정**

1️ **서버는 <interfaces> 노드를 확인**  
2️ **하위 노드 <interface> 중 <ifName>이 eth0인 데이터만 선택**  
3️ **선택된 인터페이스의 모든 형제 요소(예: <ipAddress>, <status> 등) 포함하여 반환**

**🔹 서버 응답**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✅ **eth0 인터페이스에 대한 정보만 반환됨**  
✅ **다른 인터페이스(예: eth1, wlan0)는 자동 제외됨**

🔹 서브트리 필터 처리의 핵심 원칙

텍스트, 스크린샷, 폰트, 영수증이(가) 표시된 사진

자동 생성된 설명

**🔹 정리**

✅ **서브트리 필터는 원하는 데이터만 선택적으로 가져오는 강력한 방법**  
✅ **형제 노드 단위로 필터를 적용하여 불필요한 데이터를 제거**  
✅ **서버는 루트에서 리프까지 재귀적으로 필터링 수행**  
✅ **필터 조건이 정확히 일치하는 노드만 포함 (불일치 데이터는 제거됨)**  
✅ **속성 매칭, 콘텐츠 매칭, 선택 노드 등 다양한 필터링 방식과 결합 가능**

➡ **💡 서브트리 필터를 효과적으로 사용하면 필요한 데이터만 네트워크를 통해 전송하여 최적화된 성능을 제공할 수 있음! 🚀**

## 6.4 Subtree Filtering Examples

### 6.4.1 No Filter

NETCONF의 <get> 요청에서 **필터를 지정하지 않으면**, 장치의 전체 데이터 모델이 반환됩니다.

📌 요청 예제 (필터 없음)

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <get/> 요청만 포함되어 있으며, **필터가 적용되지 않음**
* 결과적으로 장치의 **모든 데이터**가 반환됨

📌 응답 예제 (전체 데이터 반환)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <data> 태그 안에 장치의 모든 설정 및 상태 정보가 포함됨
* 이 방식은 데이터의 양이 많을 경우 **비효율적**일 수 있음
* 원하는 데이터만 가져오기 위해 **서브트리 필터링**을 적용하는 것이 일반적임

✅ **결론:**  
필터를 적용하지 않으면 **장치의 모든 데이터를 가져오므로**, 필요한 데이터만 선택적으로 가져오려면 필터링이 필요합니다.

### 6.4.2 Empty Filter

NETCONF 요청에서 **빈 필터를 사용하면 아무런 데이터도 선택되지 않습니다.** 하지만 이는 **오류가 아니며**, 정상적인 동작입니다.

📌 요청 예제 (빈 필터)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <filter> 태그는 있지만 **내부에 아무런 노드도 지정되지 않음**
* 즉, **어떤 데이터도 요청하지 않는 상태**

📌 응답 예제 (빈 데이터 반환)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <data> 태그는 존재하지만 **아무 데이터도 포함되지 않음**
* **필터 조건이 없으므로** 서버가 반환할 데이터가 없음

**✅ 결론**

* **빈 필터를 사용하면 아무 데이터도 반환되지 않음**
* **오류가 발생하지 않으며**, 정상적인 동작으로 처리됨
* 특정 데이터를 선택적으로 가져오려면 **필터에 노드를 명시적으로 지정해야 함**

### 6.4.3 Select the Entire <users> Subtree

이 예제에서는 **필터에 <users> 노드를 명시적으로 지정**하여, **전체 사용자 목록을 선택**합니다.

📌 요청 예제 (전체 <users> 서브트리 선택)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <filter> 내에 <users>를 포함하여 **사용자 정보만 가져오도록 지정**
* xmlns="http://example.com/schema/1.2/config" 네임스페이스 내에서 **구성 데이터 모델**을 가져옴

📌 응답 예제 (<users> 서브트리 포함)

텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **<users> 서브트리에 속한 모든 사용자 정보가 반환됨**
* <company-info> 같은 부가 정보도 포함

📌 동일한 결과를 내는 또 다른 요청 예제

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **<users> 노드 아래 <user> 요소를 명시적으로 포함**
* 그러나 <users>가 **오직 <user> 자식 요소만 가지므로**, 이전 요청과 동일한 응답을 받음

**✅ 결론**

* <users>를 필터에 지정하면, 해당 서브트리 전체가 반환됨
* <user/>를 추가해도 결과는 동일 (자식 요소가 하나뿐이기 때문)
* 필요에 따라 필터를 더 구체적으로 설정 가능 (예: 특정 사용자만 가져오기)

### 6.4.4 Select All <name> Elements within the <users> Subtree

이 예제에서는 **사용자 목록(<users>) 내 각 사용자(<user>)의 이름(<name>)만 선택**하는 방법을 설명합니다.

📌 요청 예제 (<name> 요소만 선택)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <users> 아래 <user>를 감싸고 그 안에 <name>을 지정
* 결과적으로 **각 사용자(user)의 <name> 요소만 가져옴**

📌 응답 예제 (<name> 요소 포함)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <users> 아래 모든 <user> 요소에서 **<name> 값만 반환됨**
* <type>, <full-name>, <company-info> 같은 다른 정보는 **포함되지 않음**

**✅ 요점 정리**

1. **필터 구조**
   * <users> → <user> → <name>을 명시하면, <name> 요소만 선택됨
   * **선택 노드**: <name> (출력될 요소)
   * **포함 노드**: <users>, <user> (필터링을 위한 경로 설정)
2. **결과 데이터**
   * <name> 값만 반환되며, 다른 사용자 정보는 포함되지 않음
3. **활용 예시**
   * 사용자 목록을 간략히 가져올 때 유용
   * 특정 사용자 이름을 기반으로 추가 요청을 보낼 수도 있음

### 6.4.5 One Specific <user> Entry

이 예제는 **이름이 "fred"인 특정 사용자**만 선택하는 필터링 방법을 설명합니다. 필터에는 **두 개의 포함 노드**(<users>, <user>)와 **하나의 값 일치 노드**(<name>이 "fred"인 경우)를 사용하여 해당 항목을 찾습니다.

📌 요청 예제 (<name>이 "fred"인 <user>만 선택)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* 필터에서 <users>와 <user>가 포함되며, <name>이 "fred"와 일치하는 사용자만 선택
* 이 요청은 "fred"라는 이름을 가진 사용자만 반환하게 됩니다.

📌 응답 예제 ("fred" 사용자만 포함)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **이름이 "fred"인 사용자만 반환**
* "fred"의 상세 정보는 응답에 포함되며, 다른 사용자들은 제외됩니다.

**✅ 요점 정리**

1. **필터 구조**
   * <users> → <user> → <name>에서 name이 "fred"인 사용자만 반환됨
   * **값 일치 노드**: <name>fred</name> (특정 값을 기반으로 필터링)
2. **결과 데이터**
   * **이름이 "fred"인 사용자만** 응답 데이터로 포함되며, 해당 사용자의 다른 정보도 반환됨
3. **활용 예시**
   * 특정 사용자의 세부 정보를 가져오고자 할 때 유용
   * 예: 시스템 관리자가 특정 사용자의 정보만 조회할 때

### 6.4.6 Specific Elements from a Specific <user> Entry

이 필터는 \*\*<users>\*\*와 **<user>** 노드를 포함하며, \*\*<name>\*\*이 "fred"인 사용자의 \*\*<type>\*\*과 **<full-name>** 요소만 선택합니다. 이 필터에서는 **<company-info>** 요소가 포함되지 않는데, 이는 **형제 요소들 중에서 선택 노드만 선택**되기 때문입니다.

📌 요청 예제 (특정 <user>에서 <type>과 <full-name>만 선택)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* 필터에서는 <users>와 <user>, <name>을 기준으로 **"fred"인 사용자**를 찾고, 해당 사용자의 \*\*<type>\*\*과 **<full-name>** 요소만 반환합니다.
* \*\*<company-info>\*\*는 필터에서 선택하지 않으므로 포함되지 않습니다.

📌 응답 예제 (이름이 "fred"인 사용자와 해당하는 요소들만 포함)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* 응답에서 **"fred"인 사용자**의 \*\*<type>\*\*과 \*\*<full-name>\*\*만 반환되며, \*\*<company-info>\*\*는 제외됩니다.

**✅ 요점 정리**

1. **필터 구조**
   * **선택 노드**: <name>이 "fred"인 사용자에 대해 \*\*<type>\*\*과 \*\*<full-name>\*\*만 선택
   * **형제 요소** 간에 선택한 노드만 반환되며, <company-info>는 포함되지 않음
2. **결과 데이터**
   * **"fred" 사용자**의 \*\*<type>\*\*과 \*\*<full-name>\*\*만 응답으로 반환
   * **선택된 요소**만 포함된 결과
3. **활용 예시**
   * 사용자의 특정 정보를 가져올 때 유용 (예: 사용자 이름과 직무만 조회)
   * 불필요한 데이터(예: 회사 정보)를 제외하고 필요한 정보만 필터링할 수 있음

### 6.4.7 Multiple Subtrees

이 예제는 \*\*세 개의 서브트리 (root, fred, barney)\*\*를 포함하는 필터 요청입니다. 각 서브트리에는 다양한 **선택 노드**와 **내용 일치 노드**가 포함되어 있으며, 그에 따라 결과가 다르게 반환됩니다.

**📌 필터 설명**

1. **"root" 서브트리 필터**
   * 필터는 \*\*<users>\*\*와 **<user>** 노드를 포함하고, \*\*<name>\*\*이 "root"인 사용자를 선택하며, **<company-info>** 요소만 포함합니다.
   * 결과: **"root" 사용자**의 \*\*<company-info>\*\*만 반환됩니다.
2. **"fred" 서브트리 필터**
   * 필터는 **<users>**, **<user>**, **<company-info>** 노드를 포함하고, \*\*<name>\*\*이 "fred"인 사용자를 선택하며, **<id>** 요소만 선택합니다.
   * 결과: **"fred" 사용자**의 \*\*<company-info>\*\*에서 \*\*<id>\*\*만 반환됩니다.
3. **"barney" 서브트리 필터**
   * 필터는 **<users>**, **<user>**, **<company-info>** 노드를 포함하고, \*\*<name>\*\*이 "barney"인 사용자를 선택하며, \*\*<type>\*\*이 "superuser"인 경우에만 **<dept>** 요소를 선택합니다.
   * 결과: **"barney" 사용자**는 \*\*<type>\*\*이 "superuser"가 아니므로 서브트리에서 제외됩니다.

📌 요청 예제 (세 개의 서브트리 필터)

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **"root" 서브트리**: **<company-info>** 요소만 반환
* “**fred" 서브트리**: \*\*<company-info>\*\*에서 **<id>** 요소만 반환
* **"barney" 서브트리**: \*\*<type>\*\*이 "superuser"가 아니므로 **서브트리 제외**

📌 응답 예제 (필터링된 결과)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* “**root" 사용자**: \*\*<company-info>\*\*에서 \*\*<dept>\*\*와 **<id>** 요소가 포함된 정보 반환
* “**fred" 사용자**: \*\*<company-info>\*\*에서 \*\*<id>\*\*만 반환
* “**barney" 사용자**는 제외됨 (필터링 기준에 맞지 않음)

**✅ 요점 정리**

1. **여러 서브트리 선택**: 각 서브트리는 독립적으로 필터링되어 결과가 다르게 반환됩니다.
   * "root" 서브트리: \*\*<company-info>\*\*만 포함
   * "fred" 서브트리: \*\*<company-info>\*\*의 \*\*<id>\*\*만 포함
   * "barney" 서브트리: \*\*"superuser"\*\*가 아니므로 제외됨
2. **서브트리 필터**는 여러 개의 데이터를 선택적으로 추출하고, 조건에 맞지 않는 데이터는 제외할 수 있습니다.
3. **실제 사용 시**: 여러 사용자에 대한 **세부 정보**를 필터링할 때 유용하며, **조건에 맞는 항목만** 반환함으로써 효율적인 데이터 추출이 가능합니다.

### 6.4.8 Elements with Attribute Naming

이 예제에서는 **속성**을 사용하여 **특정 데이터를 선택**하는 방법을 다룹니다. 여기서는 **속성 매칭**을 통해 필터링을 수행하며, 해당 속성 값이 "eth0"인 <interface> 요소만 선택됩니다.

**📌 필터 설명**

1. **필터 내용**
   * 이 필터는 **<interfaces>** 컨테이너 내의 **<interface>** 요소를 포함하고, **ifName** 속성이 "eth0"인 경우를 선택합니다.
   * **ifName** 속성은 **네임스페이스** http://example.com/schema/1.2/stats에 포함되어 있으므로 해당 속성은 **t:ifName** 형태로 표기됩니다.
2. **응답 내용**
   * 서버는 \*\*"eth0"\*\*을 가지는 **<interface>** 요소와 그 하위 요소들을 반환합니다.
   * 이 경우 \*\*<ifInOctets>\*\*와 **<ifOutOctets>** 값들이 함께 포함됩니다.

📌 요청 예제 (속성 매칭 필터링)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* 이 필터는 **<interface>** 요소 중 **ifName="eth0"** 속성을 가진 요소를 찾습니다.

📌 응답 예제 (속성 매칭 결과)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* \*\*ifName="eth0"\*\*인 **<interface>** 요소만 선택되고, 그 하위에 \*\*<ifInOctets>\*\*와 **<ifOutOctets>** 값이 포함됩니다.

**📌 속성 대신 자식 요소 사용 시**

만약 \*\*ifName\*\*이 속성이 아니라 자식 요소라면, 동일한 결과를 얻을 수 있습니다. 이 경우 요청은 다음과 같습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* 이 요청은 **<ifName>** 자식 요소에 "eth0" 값이 있는 **<interface>** 요소를 선택합니다.

**✅ 요점 정리**

1. **속성 매칭**: <interface t:ifName="eth0"/>와 같은 형태로 속성 매칭을 통해 필터링합니다.
   * 속성은 **네임스페이스**를 명시하여 매칭합니다.
2. **속성 대신 자식 요소**: 속성 대신 **자식 요소**를 사용하여 비슷한 필터링을 할 수 있습니다. 예를 들어, <ifName>eth0</ifName>로 자식 요소를 매칭할 수 있습니다.
3. **효율적인 데이터 선택**: 특정 속성이나 자식 요소를 기준으로 데이터를 선택하는 방식은 매우 효율적이며, 복잡한 데이터 구조에서 원하는 정보만을 추출하는 데 유용합니다.

# 7. Protocol Operations

NETCONF(Network Configuration Protocol)는 네트워크 장치의 **구성(Configuration) 및 상태(State) 관리**를 위한 **저수준(low-level) 프로토콜 작업**을 제공합니다.

**1️ 기본 프로토콜 작업 (Base Protocol Operations)**

NETCONF의 기본 프로토콜은 다음과 같은 작업을 포함합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**2️ 주요 개념**

* **📍 데이터스토어 (Datastore)**
  + **NETCONF는 장치의 구성을 여러 개의 데이터스토어에서 관리**합니다.
  + 예: running, candidate, startup 데이터스토어.
  + get-config을 사용하여 특정 데이터스토어에서 정보를 가져올 수 있음.
* **📍 세션 관리 (Session Management)**
  + NETCONF는 **클라이언트-서버 모델**을 기반으로 동작.
  + 여러 사용자가 동시에 장치를 수정하는 경우 **lock/unlock**을 사용하여 충돌 방지.
* **📍 RPC (Remote Procedure Call)**
  + 모든 NETCONF 작업은 **RPC(Remote Procedure Call) 메시지 형식**을 사용.
  + **응답 메시지에서 오류 코드 확인 필요!**
  + 일부 장치는 특정 연산을 지원하지 않을 수도 있음 → "operation not supported" 오류 발생 가능.

**3️ 예제: get-config 요청**

다음 예제는 running 데이터스토어에서 **구성 정보를 가져오는** get-config 요청을 보여줍니다.

**🔹 요청 (Client → Device)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **<source>**: running 데이터스토어에서 구성을 가져옴.

🔹 응답 (Device → Client)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* eth0 인터페이스에 대한 **구성 정보 반환**.

**4️ 오류 처리**

**예제: 지원되지 않는 연산 요청 시 오류 응답**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* "operation-not-supported" 오류는 장치가 특정 연산을 지원하지 않을 때 발생.

**✅ 정리**

* NETCONF는 **네트워크 장치의 구성 및 상태를 관리**하는 프로토콜.
* 주요 연산: **조회(get, get-config), 수정(edit-config), 복사(copy-config), 삭제(delete-config), 세션 관리(lock/unlock, close-session, kill-session)**.
* **RPC 방식으로 요청 및 응답을 주고받음**.
* **연산 실패 가능성 존재 → 오류 응답(rpc-error)을 항상 확인해야 함**.

프로토콜 작업의 구문과 XML 인코딩은 공식적으로 [Appendix C](#_Appendix_C._)의 YANG 모듈에 정의되어 있습니다

## 7.1 <get-config>

**📌 <get-config> 연산 (Retrieve Configuration Data)**

NETCONF의 <get-config> 연산은 **특정 데이터스토어의 전체 또는 일부 구성을 검색**하는 데 사용됩니다.

1️ <get-config> 연산 개요

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**2️ 요청 인자 (Parameters)**

**📍 1. source (필수)**

* 검색할 **데이터스토어(Datastore)를 지정**.
* 주요 데이터스토어 유형:
  + **<running/>** → 현재 실행 중인 설정
  + **<candidate/>** → 편집 가능한 임시 설정
  + **<startup/>** → 부팅 시 로드되는 설정



**📍 2. filter (선택)**

* **검색 범위를 제한**하여 원하는 정보만 가져올 수 있음.
* 기본적으로 **"subtree" (서브트리) 필터링**을 사용.
* 장치가: xpath 기능을 지원하면 **"xpath" 필터**도 사용 가능.

**✅ 서브트리(Subtree) 필터**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <users> 하위 요소 전체를 가져옴.

✅ **XPath 필터 (: xpath 지원 시)**



* "root" 사용자의 정보만 검색.

**3️ <get-config> 요청 예제**

**✅ 1. 전체 <users> 서브트리 검색**

**📌 요청 (Client → Device)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

📌 응답 (Device → Client)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**✅ 2. 특정 사용자 (fred) 검색**

**📌 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

📌 응답

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**✅ 3. ifName="eth0"인 인터페이스 정보 검색**

**📌 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

📌 응답

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**4️ 오류 응답 예제**

**🚨 요청된 데이터스토어가 존재하지 않을 경우**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**✅ 정리**

* <get-config> 연산은 **장치의 특정 데이터스토어에서 구성 정보를 검색**하는 데 사용됨.
* **source 필수**, **filter 선택** (필터를 사용하면 원하는 데이터만 추출 가능).
* 기본적으로 **서브트리(subtree) 필터링**을 사용하지만, **XPath 필터링(:xpath 지원 시)** 도 가능.
* 응답은 <rpc-reply> 내 <data> 요소에 포함되어 반환됨.
* 오류 발생 시 <rpc-error> 요소에서 원인을 확인 가능.

## 7.2 <edit-config>

<edit-config>은 NETCONF 프로토콜에서 특정 대상 구성 저장소(Target Configuration Datastore)에 설정을 변경하는 데 사용됩니다. 기존 구성을 완전히 덮어쓰는 <copy-config>와 달리, <edit-config>는 필요한 부분만 수정할 수 있도록 설계되었습니다.

**1. 주요 기능**

* **부분적 구성 수정**: 전체 구성을 교체하는 것이 아니라, 특정 항목만 변경할 수 있습니다.
* **여러 방식의 구성 적용**: 새로운 구성은 로컬 파일, 원격 파일, 또는 인라인 XML을 통해 전달될 수 있습니다.
* **구성 저장소 자동 생성**: 대상 저장소가 존재하지 않는 경우 자동으로 생성됩니다.
* **기존 구성과 비교 후 적용**: 장치는 새로운 구성과 기존 구성을 비교한 후 필요한 변경 사항만 수행합니다.

**2. operation 속성 (선택적)**

<config> 내부 요소들은 operation 속성을 가질 수 있으며, 변경 방식을 지정할 수 있습니다.  
만약 operation 속성이 명시되지 않으면 기본값인 merge가 적용됩니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**3. 주요 파라미터**

**✅ <target> (필수)**

* 변경을 적용할 대상 구성 저장소를 지정합니다.  
  예: <running/>, <candidate/>

**✅ <default-operation> (선택)**

* 개별 요소에 operation 속성이 지정되지 않은 경우 적용될 기본 동작을 정의합니다.
  + merge (기본값): 새로운 데이터를 기존 구성에 병합
  + replace: 새로운 데이터를 기존 구성 전체에 적용
  + none: 기본적으로 변경하지 않으며, operation이 지정된 항목만 변경

**✅ <test-option> (선택, :validate:1.1 지원 필요)**

* 변경 사항 적용 전 유효성 검사를 수행할지 여부를 결정합니다.
  + test-then-set (기본값): 변경 적용 전 유효성 검사를 수행, 오류 발생 시 적용하지 않음
  + set: 유효성 검사 없이 변경 사항을 바로 적용
  + test-only: 유효성 검사만 수행하고 실제 적용은 하지 않음

**✅ <error-option> (선택)**

* 오류 발생 시 수행할 동작을 지정합니다.
  + stop-on-error (기본값): 오류 발생 시 즉시 중단
  + continue-on-error: 오류 발생해도 계속 진행하며 모든 오류를 기록
  + rollback-on-error: 오류 발생 시 적용된 변경 사항을 되돌림 (서버에서 지원해야 함)

**✅ <config> (필수)**

* 변경할 구성 데이터를 포함하는 XML 구조입니다.
* 반드시 장치의 데이터 모델과 일치하는 네임스페이스를 포함해야 합니다.

**4. 응답 형태**

* **성공 시**: <rpc-reply> 내 <ok/> 반환
* **실패 시**: <rpc-reply> 내 <rpc-error> 반환

**5. 예제**

**✅ MTU 값을 1500으로 변경 (merge 기본값 적용)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **인터페이스를 기존 설정을 대체하여 추가 (replace 적용)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **인터페이스 삭제 (delete 적용)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 요약**

* <edit-config>은 NETCONF에서 구성 요소를 부분적으로 수정할 때 사용됩니다.
* operation 속성을 통해 특정 방식(병합, 교체, 삭제 등)으로 변경할 수 있습니다.
* <default-operation>, <test-option>, <error-option>을 활용하여 적용 방식을 세밀하게 조정할 수 있습니다.
* 변경 사항이 성공적으로 적용되면 <ok/>, 실패하면 <rpc-error>가 반환됩니다.

## 7.3 <copy-config>

<copy-config>은 NETCONF 프로토콜에서 전체 구성 저장소(Configuration Datastore)를 교체하거나 새로 생성하는 데 사용됩니다. 이 명령은 기존 구성을 부분적으로 변경하는 <edit-config>와 달리, **완전히 새로운 구성으로 대체**하는 것이 특징입니다.

**1. 주요 기능**

* **전체 구성 교체**: 기존 저장소가 존재하면 **완전히 덮어씀**.
* **새로운 저장소 생성 가능**: 대상 저장소가 존재하지 않을 경우, 허용된다면 새롭게 생성.
* **다양한 소스 지원**: 로컬 저장소뿐만 아니라 URL을 통한 원격 파일도 가능.
* **running 저장소 제한 가능**: 일부 장치는 running 저장소를 대상으로 지정하는 것을 허용하지 않을 수 있음.
* **원격 간 복사 제한 가능**: 일부 장치는 두 개의 원격 저장소 간 복사를 허용하지 않을 수 있음.
* **동일한 source와 target 금지**: source와 target이 동일한 경우 오류 발생.

**2. 주요 파라미터**

**✅ <target> (필수)**

* 복사 대상이 될 구성 저장소를 지정합니다.
  + 예: <running/>, <candidate/>, <startup/>, <url>...<url/>

**✅ <source> (필수)**

* 복사할 원본 구성 저장소를 지정합니다.
  + 예: <running/>, <candidate/>, <startup/>, <url>...<url/>
  + <config> 요소를 사용하여 직접 XML 데이터를 포함할 수도 있음.

**3. 응답 형태**

* **성공 시**: <rpc-reply> 내 <ok/> 반환
* **실패 시**: <rpc-reply> 내 <rpc-error> 반환

**4. 예제**

**✅ 원격 URL에서 running 저장소로 구성 복사**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

응답 (성공 시)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **candidate 저장소에서 running 저장소로 복사**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **새로운 구성 데이터를 startup 저장소로 복사**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**5. 주요 제한 사항**

* 일부 장치는 running 저장소를 <target>으로 지정하는 것을 지원하지 않을 수 있음.
* 원격 저장소 간 (<url> → <url>) 복사는 허용되지 않을 수 있음.
* source와 target이 동일할 경우 "invalid-value" 오류 발생.

**6. 요약**

* <copy-config>은 전체 구성 저장소를 새로운 데이터로 교체할 때 사용됩니다.
* <edit-config>과 달리 기존 설정을 유지하지 않고 **완전히 덮어씌움**.
* 원본과 대상은 로컬 저장소(running, candidate, startup 등) 또는 원격 URL이 될 수 있음.
* 동일한 원본과 대상 간 복사는 허용되지 않으며, 일부 장치는 특정 대상(running)을 허용하지 않을 수도 있음.

## 7.4 <delete-config>

<delete-config>은 NETCONF 프로토콜에서 **특정 구성 저장소(Configuration Datastore)를 삭제**하는 데 사용됩니다.

**1. 기능 개요**

* **구성 저장소 삭제**: 지정된 저장소를 제거함.
* **running 저장소 삭제 불가**: running 저장소는 활성 구성 데이터이므로 삭제할 수 없음.
* **URL 지원 가능**: 장치가: url 기능을 지원하는 경우, 원격 저장소(url) 삭제 가능.

**2. 주요 파라미터**

**✅ <target> (필수)**

* 삭제할 구성 저장소를 지정합니다.
  + 예: <startup/>, <candidate/>, <url>...<url/>

**3. 응답 유형**

* **성공 (<ok/> 반환)**
  + 삭제 작업이 정상적으로 수행됨.
* **실패 (<rpc-error> 반환)**
  + 대상 저장소가 존재하지 않거나 삭제가 불가능한 경우 오류 발생.

**4. 예제**

**✅ startup 저장소 삭제**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

응답 (성공 시)

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **원격 URL에 저장된 구성 삭제**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**5. 주요 제한 사항**

* **running 저장소는 삭제 불가**
* **장치가: url 기능을 지원하지 않을 수 있음**
* **삭제할 대상이 존재하지 않으면 오류 발생 가능**

**6. 요약**

* <delete-config>은 특정 구성 저장소를 삭제하는 기능.
* running 저장소는 삭제할 수 없음.
* startup, candidate, 또는 원격 URL(:url)을 대상으로 삭제 가능.
* 성공 시 <ok/>, 실패 시 <rpc-error> 응답.

## 7.5 <lock>

<lock>은 NETCONF 프로토콜에서 **구성 저장소(Configuration Datastore)에 대한 잠금을 설정**하여, 다른 클라이언트 또는 관리 인터페이스(SNMP, CLI 등)가 변경하지 못하도록 보호하는 기능입니다.

**1. 기능 개요**

* **구성 저장소에 대한 독점적 잠금 설정**
* **다른 NETCONF 세션, SNMP, CLI 등의 변경 요청 차단**
* **잠금 유지 조건**
  + **클라이언트가 잠금을 해제할 때까지 유지**
  + **NETCONF 세션이 종료될 경우 자동 해제** (네트워크 장애, 시간 초과 등)

**2. 주요 파라미터**

**✅ <target> (필수)**

* 잠금을 설정할 구성 저장소를 지정.
  + 예: <running/>, <candidate/>

**3. 응답 유형**

* **성공 (<ok/> 반환)**
  + 정상적으로 잠금이 설정됨.
* **실패 (<rpc-error> 반환)**
  + 이미 다른 세션이 잠금을 보유한 경우.
  + candidate 저장소가 수정되었으나, commit 또는 rollback이 되지 않은 경우.
  + running 저장소에 대해 다른 세션이 confirmed commit을 수행 중인 경우.

**4. 예제**

**✅ running 저장소 잠금 성공**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

응답 (성공 시)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **이미 잠긴 저장소에 대한 잠금 요청 (실패 사례)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**응답 (실패 시)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* session-id가 454인 세션이 이미 잠금을 보유하고 있음.

**5. 주요 제한 사항**

* **기존 잠금이 있는 경우 요청 실패** (**lock-denied** 오류)
* **candidate 저장소는 수정 후 commit 또는 rollback되지 않으면 잠금 불가**
* **running 저장소는 다른 세션이 confirmed commit을 수행 중이면 잠금 불가**
* **잠금 강제 해제 방법 (<kill-session>)**
  + <kill-session>을 사용하여 특정 세션을 강제 종료하면 해당 세션의 잠금이 해제됨.

**6. 요약**

* <lock>은 특정 구성 저장소를 보호하는 기능.
* **다른 NETCONF 세션, SNMP, CLI 등의 변경을 차단**.
* **기존 잠금이 있으면 실패 (lock-denied)**.
* **잠금은 해제될 때까지 유지되며, 세션이 종료되면 자동 해제됨**.
* **다른 세션이 잠금을 보유한 경우 <kill-session>을 통해 강제 해제 가능**.

## 7.6 <unlock>

<unlock>은 NETCONF 프로토콜에서 **이전에 <lock>을 통해 설정된 구성 저장소의 잠금을 해제**하는 기능입니다.

**1. 기능 개요**

* **잠금(lock)이 설정된 구성 저장소를 해제**
* **잠금을 설정한 동일한 세션에서만 해제 가능**
* **다른 세션에서 해제 요청 시 거부됨 (rpc-error 반환)**

**2. 주요 파라미터**

**✅ <target> (필수)**

* 잠금을 해제할 구성 저장소 지정
  + 예: <running/>, <candidate/>

**3. 응답 유형**

* **성공 (<ok/> 반환)**
  + 정상적으로 잠금 해제됨.
* **실패 (<rpc-error> 반환)**
  + **잠금이 설정되지 않은 경우** (존재하지 않는 잠금 해제 요청)
  + **잠금을 설정한 세션이 아닌 다른 세션에서 요청한 경우**

**4. 예제**

**✅ running 저장소 잠금 해제 성공**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

응답 (성공 시)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **잠금되지 않은 저장소에 대한 해제 요청 (실패 사례)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**응답 (실패 시)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <error-message>: "Unlock failed, no active lock found" → **잠금이 설정되지 않은 경우**

✅ **다른 세션에서 잠금 해제 요청 (실패 사례)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

응답 (실패 시)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* <error-message>: "Unlock failed, lock held by another session" → **잠금을 설정한 세션이 아닌 경우**

**5. 주요 제한 사항**

* **NETCONF 클라이언트는 자신이 설정한 잠금만 해제 가능**
* **다른 세션이 설정한 잠금은 <unlock>으로 해제할 수 없음**
  + 이 경우, **<kill-session>을 사용하여 해당 세션을 강제 종료**해야 함.

**6. 요약**

* <unlock>은 <lock>을 통해 설정된 잠금을 해제하는 기능.
* **잠금을 설정한 동일한 세션에서만 해제 가능**.
* **잠금이 없는 저장소에 대한 요청은 실패** (operation-failed).
* **다른 세션이 설정한 잠금을 해제하려면 <kill-session>을 사용해야 함**.

## 7.7 <get>

<get>은 NETCONF 프로토콜에서 **장치의 실행 중인 구성(running configuration) 및 상태 정보를 검색**하는 기능입니다.

**1. 기능 개요**

* **실행 중인 구성 및 장치 상태 데이터를 조회**
* **필터(filter)를 사용하여 특정 데이터만 선택적으로 조회 가능**
* **필터가 없을 경우, 전체 데이터를 반환**

**2. 주요 파라미터**

**✅ <filter> (선택 사항)**

* 검색할 데이터 범위를 지정하는 선택적 필터
* 필터가 없으면 **전체 구성 및 상태 정보를 조회**
* **type 속성을 사용하여 필터링 방식 지정 가능**
  + "subtree" (기본값) → **서브트리 필터링**
  + "xpath" → **XPath 표현식을 사용한 필터링** (NETCONF 장치가 지원할 경우)

**3. 응답 유형**

* **성공 시 (<rpc-reply> + <data> 반환)**
  + 요청한 데이터가 <data> 요소 안에 포함됨.
* **실패 시 (<rpc-error> 반환)**
  + 잘못된 요청이거나 장치가 응답할 수 없는 경우

**4. 예제**

**✅ 특정 인터페이스(eth0) 상태 정보 조회 (서브트리 필터링)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

응답 (성공 시)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **요청:** eth0 인터페이스의 데이터 조회
* **응답:** eth0의 수신(ifInOctets) 및 송신(ifOutOctets) 바이트 정보 포함

✅ **XPath를 이용한 필터링 예제 (xpath 필터)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **요청:** XPath 필터를 사용하여 eth0 인터페이스 정보 조회
* **조건:** 장치가: xpath 기능을 지원해야 함

**5. 주요 제한 사항**

* **필터를 사용하지 않으면 전체 데이터를 반환하므로, 대량 데이터 조회 시 주의 필요**
* **XPath 필터링은 지원 여부가 장치에 따라 다름**
* **<get>은 읽기 전용이며, 설정을 변경하지 않음**

**6. 요약**

* <get>은 **현재 실행 중인 구성 및 장치 상태 데이터를 검색**하는 NETCONF 명령
* **필터링(subtree, xpath)을 사용하여 특정 정보만 조회 가능**
* **성공 시 <data> 요소에 요청한 데이터 반환**
* **실패 시 <rpc-error> 반환**
* **구성을 변경하는 <edit-config>와 다르게, <get>은 읽기 전용**

## 7.8 <close-session>

<close-session>은 **NETCONF 세션을 정상적으로 종료**하는 기능입니다.

**1. 기능 개요**

* **현재 NETCONF 세션을 종료 요청**
* **서버가 세션 종료 전에 모든 리소스와 락을 해제**
* **이후 해당 세션으로부터의 모든 요청을 무시**

**2. 응답 유형**

* **성공 시 (<rpc-reply> + <ok> 반환)**
  + 정상적으로 세션이 종료됨
* **실패 시 (<rpc-error> 반환)**
  + 세션 종료가 불가능한 경우 (예: 서버 문제)

**3. 예제**

**✅ 세션 정상 종료 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

응답 (성공 시)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **요청:** 현재 NETCONF 세션을 정상적으로 종료
* **응답:** <ok/>이 반환되면 성공적으로 종료됨

**4. 주요 특징 및 제한 사항**

* <close-session>을 실행하면 **현재 세션에서 획득한 모든 리소스 및 락이 자동 해제됨**
* **서버는 요청을 받은 후 즉시 해당 세션과의 연결을 종료**
* **이후 해당 세션에서 오는 추가 요청은 무시됨**
* **비정상적인 세션 종료 시 <kill-session>을 사용할 수 있음** (강제 종료)

**5. <close-session> vs <kill-session> 비교**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 요약**

* <close-session>은 **NETCONF 세션을 정상적으로 종료**하는 명령
* **서버는 모든 리소스와 락을 해제한 후 연결 종료**
* **성공 시 <ok/> 반환, 실패 시 <rpc-error> 반환**
* **비정상적인 종료가 필요할 경우 <kill-session> 사용 가능**

## 7.9 <kill-session>

<kill-session>은 **NETCONF 세션을 강제 종료**하는 기능입니다.

**1. 기능 개요**

* **현재 열린 NETCONF 세션을 강제로 종료**
* **진행 중인 모든 작업을 즉시 중단**
* **세션이 보유한 리소스 및 락 해제 후 연결 종료**
* **강제 종료 시에도 기존의 설정 변경 내용은 롤백되지 않음**
  + 단, **confirmed commit이 진행 중이면 이전 상태로 복원**

**2. 응답 유형**

* **성공 시 (<rpc-reply> + <ok> 반환)**
  + 정상적으로 세션이 강제 종료됨
* **실패 시 (<rpc-error> 반환)**
  + 요청한 session-id가 현재 세션과 동일한 경우 (invalid-value 오류)
  + 존재하지 않는 세션을 종료하려는 경우

**3. 예제**

**✅ 세션 강제 종료 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

응답 (성공 시)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **요청:** session-id가 4인 세션을 강제 종료
* **응답:** <ok/>이 반환되면 성공적으로 종료됨

**4. 주요 특징 및 제한 사항**

* <kill-session>은 **비정상적이거나 응답하지 않는 NETCONF 세션을 종료하는 데 사용**
* **본인이 사용 중인 세션 (session-id)을 종료할 수 없음** (invalid-value 오류 발생)
* **강제 종료 시, 해당 세션이 보유한 모든 락과 리소스는 자동으로 해제됨**
* **confirmed commit이 진행 중이라면, 해당 변경 사항은 자동 롤백됨**
* **그 외의 변경 사항은 롤백되지 않음** → 필요하면 <discard-changes> 사용

**5. 요약**

* <kill-session>은 **NETCONF 세션을 강제로 종료**하는 명령
* **해당 세션의 모든 작업을 중단하고, 락 및 리소스를 해제한 후 연결 종료**
* **confirmed commit이 진행 중이면 롤백되지만, 일반적인 설정 변경은 유지됨**
* **현재 세션을 종료할 수 없으며, 시도 시 invalid-value 오류 발생**
* **정상 종료하려면 <close-session>을 사용**

# 8. Capabilities

Capabilities는 **NETCONF 클라이언트와 서버가 지원하는 기능을 광고하는 메커니즘**입니다.

**1. Capabilities 개념**

* **클라이언트와 서버가 초기 연결 시 자신의 지원 기능을 교환**
* **각 피어(peer)는 필요한 기능만 사용하고, 이해할 수 없는 기능은 무시해야 함**
* **새로운 기능은 표준 기구 또는 벤더가 추가로 정의 가능**

**2. Capabilities의 표현 형식**

* **각 Capability는 URI 형식으로 식별됨**
* 기본 형식: urn:ietf:params:netconf:capability:{name}:1.x
  + {name} -> 기능 이름
  + 1.x -> 버전 정보
* 간단한 표기법 (shorthand)
  + :{name} 또는 :{name}:{version}으로 줄여서 표현 가능
  + 예시:
    - urn:ietf:params:netconf:capability:foo:1.0 → :foo
    - urn:ietf:params:netconf:capability:bar:1.1 → :bar:1.1
* 중요!
  + 프로토콜 내에서는 반드시 URI 형식을 사용해야 하며, shorthand 표기법은 사용 금지

**3. Capabilities 예시**

**✅ 서버가 지원하는 Capabilities 광고**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **urn:ietf:params:netconf:base:1.1** → NETCONF 1.1 지원
* **urn:ietf:params:netconf:capability:candidate:1.0** → candidate 데이터스토어 지원
* **urn:ietf:params:netconf:capability:rollback-on-error:1.0** → 오류 발생 시 롤백 기능 지원

**4. Capabilities 추가 및 확장**

* **추가적인 Capabilities는 표준 기구 또는 벤더가 정의 가능.** [**Appendix D**](#_Appendix_D._)**.**
* [**RFC 3553**](#_13.1_Normative_References)**에 따라 URN을 사용하여 새로운 기능을 등록할 수 있음**
* **추가된 Capability는 기존 NETCONF 클라이언트/서버가 이해하지 못하더라도 무시해야 함**

**5. 정리**

* **NETCONF Capabilities는 클라이언트와 서버가 지원하는 기능을 교환하는 메커니즘**
* **각 Capability는 urn:ietf:params:netconf:capability:{name}:1.x 형식의 URI로 표현**
* **초기 hello 메시지를 통해 Capabilities를 광고하고 교환**
* **이해할 수 없는 Capabilities는 반드시 무시해야 함**
* **shorthand 표기법은 프로토콜 내부에서는 사용할 수 없음**

## 8.1 Capabilities Exchange

NETCONF **클라이언트와 서버는 세션을 시작할 때 자신이 지원하는 기능(Capabilities)을 광고**합니다.

**1. Capabilities 교환 과정**

1. **NETCONF 세션이 열리면, 클라이언트와 서버는 <hello> 메시지를 주고받음**
2. **각 피어(peer)는 <hello> 메시지 안에 자신이 지원하는 기능을 <capabilities> 리스트로 포함**
3. **최소한 기본 NETCONF 기능(urn:ietf:params:netconf:base:1.1)을 포함해야 함**
4. **이전 버전의 NETCONF도 지원 가능하며, 이를 명시할 수 있음**
5. **클라이언트와 서버는 공통적으로 지원하는 프로토콜 버전을 확인해야 함**
   * **공통된 버전이 없으면 세션을 종료해야 함**
   * **공통된 버전이 여러 개라면, 가장 최신 버전을 사용해야 함**

**2. 세션 ID(Session ID)**

* **서버는 <hello> 메시지에 <session-id>를 포함해야 함**
* **클라이언트는 <hello> 메시지에 <session-id>를 포함하면 안 됨**
* **서버가 클라이언트의 <hello> 메시지에서 <session-id>를 발견하면 세션을 종료해야 함**
* **클라이언트는 서버의 <hello> 메시지에서 <session-id>가 없으면 세션을 종료해야 함**

**3. Capabilities 교환 예시**

**✅ 서버가 광고하는 <hello> 메시지**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **urn:ietf:params:netconf:base:1.1** → NETCONF 1.1 프로토콜 지원
* **urn:ietf:params:netconf:capability:startup:1.0** → startup 기능 지원
* **http://example.net/router/2.3/myfeature** → 특정 벤더가 정의한 기능
* **<session-id>**

**4. Capabilities 교환 규칙**

✅ **서버와 클라이언트는 <hello> 메시지를 동시에 보내야 하며, 상대방의 메시지를 기다리면 안 됨**  
✅ **공통적인 프로토콜 버전이 없으면 세션을 중단해야 함**  
✅ **가장 높은 버전의 공통 프로토콜을 사용해야 함**

**5. 정리**

* **NETCONF 세션이 시작되면 클라이언트와 서버는 <hello> 메시지를 주고받으며 Capabilities를 교환**
* **각 피어는 자신이 지원하는 기능을 <capabilities> 리스트로 제공**
* **최소한 urn:ietf:params:netconf:base:1.1을 포함해야 함**
* **공통된 프로토콜 버전이 없으면 세션을 종료해야 함**
* **서버는 <session-id>를 포함해야 하고, 클라이언트는 포함하면 안 됨**

## 8.2 Writable-Running Capability

**1. 기능 설명 (Description)**

:writable-running **기능을 지원하는 장치는**

* **<running> 데이터스토어에 직접 쓰기(write) 가능**
* **즉, <edit-config> 및 <copy-config> 연산에서 <running>을 대상으로 사용할 수 있음**

**2. 의존성 (Dependencies)**

* **없음 (None)**

**3. Capability 식별자 (Identifier)**

* **urn:ietf:params:netconf:capability:writable-running:1.0**

**4. 새로운 연산 (New Operations)**

* **없음 (None)**

**5. 기존 연산의 수정 (Modifications to Existing Operations)**

**5.1. <edit-config> 연산 변경**

* <edit-config>에서 **<running>을 <target>으로 설정할 수 있음**
* 즉, <running> 데이터스토어를 직접 수정 가능

✅ **예제 (<edit-config> 사용)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**5.2. <copy-config> 연산 변경**

* <copy-config>에서 **<running>을 <target>으로 설정할 수 있음**
* 즉, **다른 데이터스토어의 설정을 <running>에 복사 가능**

✅ **예제 (<copy-config> 사용)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* 위 예제는 **<startup> 설정을 <running>에 복사**하는 동작을 수행

**6. 정리**

✅ **:writable-running을 지원하면 <running> 데이터스토어를 직접 수정할 수 있음**  
✅ **<edit-config> 및 <copy-config>에서 <running>을 <target>으로 설정 가능**  
✅ **기본적으로 NETCONF 서버는 <running>을 직접 수정할 수 없지만, 이 기능을 지원하면 가능**

## 8.3 Candidate Configuration Capability

**1. 기능 설명 (Description)**

:candidate **기능을 지원하는 장치는**

* **임시 저장소인 <candidate> 데이터스토어를 제공**
* **구성을 변경하더라도 즉시 장치에 적용되지 않음**
* **<commit> 연산을 실행해야 <running> 데이터스토어에 반영됨**
* **<discard-changes> 연산을 실행하면 변경 사항을 취소 가능**
* **다중 세션이 <candidate>를 공유할 수 있으므로, 수정 전 <lock>을 권장**

**2. 의존성 (Dependencies)**

* **없음 (None)**

**3. Capability 식별자 (Identifier)**

* **urn:ietf:params:netconf:capability:candidate:1.0**

**4. 새로운 연산 (New Operations)**

**4.1. <commit> 연산**

* <candidate> 데이터스토어의 변경 사항을 <running> 데이터스토어로 반영
* **완전한 변경 사항 적용이 실패하면 <running>은 변경되지 않음**
* **다른 세션이 <running> 또는 <candidate>를 잠그고 있으면 <commit>은 실패 (error-tag: in-use)**
* **:candidate 기능이 없는 시스템에서는 <commit> 사용 불가**

✅ **예제 (<commit> 실행)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**4.2. <discard-changes> 연산**

* **적용되지 않은 변경 사항을 취소하고 <candidate>를 <running>과 동일하게 복원**

✅ **예제 (<discard-changes> 실행)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**5. 기존 연산의 수정 (Modifications to Existing Operations)**

**5.1. <get-config>, <edit-config>, <copy-config>, <validate> 연산**

* **<candidate>를 <source> 또는 <target>으로 설정 가능**

✅ **예제 (<get-config>에서 <candidate> 사용)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**5.2. <lock> 및 <unlock> 연산**

* <candidate> 데이터스토어를 **잠그거나 (<lock>) 해제 (<unlock>) 가능**
* **세션이 중단되면 자동으로 변경 사항이 폐기됨**

✅ **예제 (<lock> 실행)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **예제 (<unlock> 실행)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 정리**

✅ **:candidate를 지원하면 <candidate> 데이터스토어에서 변경 작업을 수행 후 <commit>으로 적용 가능**  
✅ **변경 사항이 적용되기 전에는 운영 설정에 영향을 주지 않음**  
✅ **수정 중 변경 사항을 취소하려면 <discard-changes> 사용**  
✅ **다른 세션이 <candidate>를 수정할 수 있으므로 <lock> 권장**

## 8.4 Confirmed Commit Capabilities

**1. 기능 설명 (Description)**

:confirmed-commit:1.1 **기능을 지원하는 장치는**

* <commit> 연산에 **<confirmed>, <confirm-timeout>, <persist>, <persist-id> 매개변수를 추가 지원**
* <cancel-commit> 연산을 통해 **확정되지 않은 commit을 취소 가능**
* **기본적으로 600초(10분) 이내에 확정(commit)되지 않으면 자동으로 롤백됨**
* **재부팅 또는 세션 종료 시에도 자동 롤백**
* **<persist> 옵션을 사용하면 세션이 종료되어도 롤백되지 않으며, 다른 세션에서도 commit 확정 가능**
* **:startup 기능이 있는 경우 <copy-config>를 실행해야 <startup>에도 반영됨**
* **공유 데이터스토어 사용 시 <lock>을 함께 사용하는 것이 권장됨**

**2. 의존성 (Dependencies)**

* **:candidate 기능이 필수**

**3. Capability 식별자 (Identifier)**

* **urn:ietf:params:netconf:capability:confirmed-commit:1.1**

**4. 새로운 연산 (New Operations)**

**4.1. <cancel-commit> 연산**

* **확정되지 않은 commit을 취소하고 변경 사항을 롤백**
* **<persist-id>를 지정하면 다른 세션에서도 취소 가능**

✅ **예제 (<cancel-commit> 실행)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**5. 기존 연산의 수정 (Modifications to Existing Operations)**

**5.1. <commit> 연산의 추가 매개변수**

* **<confirmed>**: **확정되지 않은 commit을 수행**
* **<confirm-timeout>**: **확정되지 않으면 자동 롤백되는 시간 설정 (기본 600초)**
* **<persist>**: **세션 종료 후에도 유지되는 commit 설정**
* **<persist-id>**: **다른 세션에서 확정 commit을 진행할 때 사용**

✅ **예제 (<commit> 실행, 120초 타임아웃 설정)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **예제 (<commit> 실행, <persist> 옵션 사용)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **예제 (다른 세션에서 <commit> 확정, <persist-id> 사용)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 정리**

✅ **<confirmed> 옵션을 사용하면 일정 시간 내 확정되지 않으면 자동 롤백됨**  
✅ **<persist> 옵션을 사용하면 세션 종료 후에도 유지됨**  
✅ **<cancel-commit>을 통해 즉시 롤백 가능**  
✅ **<persist-id>를 사용하면 다른 세션에서도 확정 commit을 진행 가능**  
✅ **공유 데이터스토어에서는 <lock>과 함께 사용하는 것이 안전함**

## 8.5 Rollback-on-Error Capability

**1. 기능 설명 (Description)**

:rollback-on-error **기능을 지원하는 장치는**

* <edit-config> 연산의 <error-option> 매개변수에 "rollback-on-error" 값을 지원
* **설정 중 오류 발생 시 전체 변경 사항을 자동으로 롤백**
* **공유 데이터스토어 사용 시 <lock>을 사용하여 충돌 방지하는 것이 권장됨**

**2. 의존성 (Dependencies)**

* **의존성 없음 (None)**

**3. Capability 식별자 (Identifier)**

* **urn:ietf:params:netconf:capability:rollback-on-error:1.0**

**4. 새로운 연산 (New Operations)**

* **새로운 연산 없음 (None)**

**5. 기존 연산의 수정 (Modifications to Existing Operations)**

**5.1. <edit-config> 연산의 <error-option> 매개변수 추가**

* **rollback-on-error** :
  + **설정 적용 중 오류가 발생하면 변경 사항 전체를 롤백**
  + **기본적으로 부분 적용을 허용하는 stop-on-error 또는 continue-on-error보다 더 안전한 옵션**

✅ **예제 (rollback-on-error 옵션 적용한 <edit-config> 실행)**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 정리**

✅ **설정 적용 중 오류 발생 시 자동으로 롤백**  
✅ **rollback-on-error 옵션을 사용하여 데이터 무결성 유지**  
✅ **공유 데이터스토어에서는 <lock>을 사용하여 안전하게 적용**  
✅ **다른 <error-option>(stop-on-error, continue-on-error)보다 안정적인 설정 방식**

## 8.6 Validate Capability

**1. 기능 설명 (Description)**

✅ **설정이 적용되기 전, 구문(Syntax) 및 의미(Semantic) 오류를 검사하는 기능**  
✅ **이 기능을 지원하는 장치는 <validate> 연산을 제공하며 최소한 구문 오류를 검사**  
✅ **<edit-config> 연산에 <test-option> 매개변수를 추가 지원**  
✅ **버전 1.1에서는 새로운 <test-option> 값인 "test-only" 추가**

**2. 의존성 (Dependencies)**

* **없음 (None)**

**3. Capability 식별자 (Identifier)**

* **urn:ietf:params:netconf:capability:validate:1.1**

**4. 새로운 연산 (New Operations)**

**4.1. <validate> 연산 추가**

**✅ <validate> 연산은 특정 구성 데이터를 검증하는 역할을 수행**

🔹 **파라미터 (<source>)**

* **검증할 데이터 저장소 지정 (<candidate>, <config> 등)**

🔹 **응답 유형**

* ✅ **성공 시 (<ok>)**
* ❌ **오류 발생 시 (<rpc-error>)**
  + 구문 오류(Syntax Error)
  + 누락된 필수 매개변수
  + 정의되지 않은 데이터 참조
  + 데이터 모델 규칙 위반

✅ **예제: <candidate> 데이터스토어에 대한 검증 요청**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**5. 기존 연산의 수정 (Modifications to Existing Operations)**

**5.1. <edit-config> 연산 수정 (<test-option> 추가)**

**✅ <edit-config> 연산에서 <test-option> 매개변수를 지원**

🔹 **<test-option> 매개변수 값**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✅ **예제: test-only 옵션을 사용하여 설정을 검증만 수행**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 정리**

✅ **설정 변경 전에 오류를 검사하여 안정성을 높임**  
✅ **<validate> 연산을 사용하여 독립적인 설정 검증 가능**  
✅ **<edit-config>의 <test-option>을 통해 설정의 검증 수준을 조절 가능**  
✅ **버전 1.1에서는 "test-only" 옵션 추가로 설정을 안전하게 검증할 수 있음**

## 8.7 Distinct Startup Capability

**1. 기능 설명 (Description)**

✅ **장치가 running과 startup 구성 저장소를 별도로 유지**  
✅ **부팅 시 startup 구성을 로드하며, running 구성 변경은 startup에 자동 반영되지 않음**  
✅ **running → startup 동기화는 명시적으로 <copy-config> 연산을 사용해야 함**  
✅ **NETCONF 프로토콜에서 <startup> 요소를 사용하여 startup 데이터스토어를 참조**

**2. 의존성 (Dependencies)**

* **없음 (None)**

**3. Capability 식별자 (Identifier)**

* **urn:ietf:params:netconf:capability:startup:1.0**

**4. 새로운 연산 (New Operations)**

* **없음 (None)**

**5. 기존 연산의 수정 (Modifications to Existing Operations)**

✅ **이 기능(:startup)은 <startup/> 데이터스토어를 여러 NETCONF 연산에서 사용 가능하도록 확장**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 주요 예제**

**6.1. running → startup 구성 동기화 (<copy-config>)**

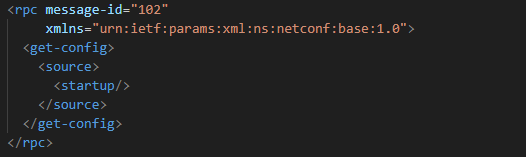
✅ **변경된 running 설정을 startup에 저장하는 예제**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6.2. startup 데이터스토어 내용 가져오기 (<get-config>)**

✅ **부팅 시 로드될 설정 확인**



**6.3. startup 구성 삭제 (<delete-config>) → 공장 초기화**

✅ **공장 초기화(Factor Reset) 수행**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

⚠ **주의:** startup 삭제 시 장치는 **공장 초기화 상태로 복귀**

**7. 정리**

✅ **running과 startup을 분리하여 설정 관리 가능**  
✅ **변경 사항이 부팅 후 유지되려면 <copy-config>를 사용해야 함**  
✅ **<delete-config>를 통해 startup 삭제 시 장치가 초기 설정으로 재설정됨**  
✅ **NETCONF 연산에서 <startup>을 지원하는 기능 확장됨**

## 8.8 URL Capability

**1. 기능 설명 (Description)**

✅ **NETCONF 피어가 <source> 및 <target> 매개변수에 <url> 요소를 허용**  
✅ **URL 스킴(scheme)에 따라 다양한 원격 파일 저장소(HTTP, FTP, FILE 등)에서 설정 관리 가능**  
✅ **이 기능이 활성화된 경우, <edit-config>, <copy-config>, <delete-config>, <validate> 연산이 <url>을 지원**

**2. 의존성 (Dependencies)**

* **없음 (None)**

**3. Capability 식별자 (Identifier)**

* **기본 URI 형식:** urn:ietf:params:netconf:capability:url:1.0?scheme={name,...}
* **예제:** urn:ietf:params:netconf:capability:url:1.0?scheme=http,ftp,file  
  → 이 경우, **HTTP, FTP, FILE** 스킴을 지원함을 의미

**4. 새로운 연산 (New Operations)**

* **없음 (None)**

**5. 기존 연산의 수정 (Modifications to Existing Operations)**

✅ **:url 기능이 활성화되면, 아래 연산에서 <url> 요소 사용 가능**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 주요 예제**

**6.1. 원격 파일에서 설정 적용 (<edit-config>)**

✅ **HTTP 서버에 저장된 설정 파일을 로드하여 running 구성 수정**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

📌 **설정 파일 (config.xml)은 XML 형식이어야 하며 <config> 루트 요소 포함**

**6.2. 원격 파일에서 구성 복사 (<copy-config>)**

✅ **FTP 서버에서 startup 구성으로 복사**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6.3. 원격 구성 삭제 (<delete-config>)**

✅ **파일 시스템에서 특정 구성 파일 삭제**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6.4. 원격 구성 검증 (<validate>)**

✅ **원격 구성 파일이 유효한지 확인**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**7. 정리**

✅ **NETCONF에서 <url>을 지원하면 외부 파일을 설정 소스로 활용 가능**  
✅ **HTTP, FTP, FILE 등 다양한 스킴을 통해 설정 데이터를 로드 및 관리**  
✅ **구성 편집(<edit-config>), 복사(<copy-config>), 삭제(<delete-config>), 검증(<validate>)에서 활용 가능**  
✅ **파일 형식은 반드시 <config> 루트 요소를 포함한 XML이어야 함**

## 8.9 XPath Capability

**1. 기능 설명 (Description)**

✅ **NETCONF 피어가 <filter> 요소에서 XPath 표현식을 지원함**  
✅ **XPath 1.0 (**[**W3C.REC-xpath-19991116**](https://www.w3.org/TR/1999/REC-xpath-19991116/)**) 기반**  
✅ **데이터 필터링을 위해 사용되며, 결과는 반드시 "node-set" 형식이어야 함**  
✅ **<get> 및 <get-config> 연산에서 XPath를 필터링 방식으로 활용 가능**

**2. 의존성 (Dependencies)**

* **없음 (None)**

**3. Capability 식별자 (Identifier)**

* **기본 URI 형식:** urn:ietf:params:netconf:capability:xpath:1.0

**4. 새로운 연산 (New Operations)**

* **없음 (None)**

**5. 기존 연산의 수정 (Modifications to Existing Operations)**

✅ **:xpath 기능이 활성화되면, <get> 및 <get-config> 연산에서 XPath 필터 사용 가능**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**6. 주요 예제**

**6.1. 특정 사용자 정보 조회 (<get-config>)**

✅ **사용자 중에서 이름이 "fred"인 사용자만 필터링하여 가져오기**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✔ **XPath 표현식: /t:top/t:users/t:user[t:name='fred']**  
✔ **필터를 통해 "fred"라는 이름을 가진 사용자만 조회**

✅ **서버 응답 예시**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

📌 **"fred"의 정보만 반환되며, 계층적 구조가 유지됨**

**6.2. 특정 VLAN 정보 조회 (<get-config>)**

✅ **VLAN ID가 100인 VLAN 설정 정보만 조회**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

✔ **XPath 표현식: /t:network/t:vlans/t:vlan[t:id='100']**  
✔ **VLAN ID가 100인 항목만 필터링**

✅ **서버 응답 예시**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**7. 정리**

✅ **NETCONF에서 XPath 필터링을 지원하면 더 정밀한 데이터 조회 가능**  
✅ **<get> 및 <get-config>에서 type="xpath" 필터링 방식 사용**  
✅ **XPath 표현식이 반드시 "node-set"을 반환해야 함**  
✅ **필터링된 데이터는 원본 계층 구조를 유지한 채 반환됨**  
✅ **사용 예시: 특정 사용자 정보 조회, VLAN 설정 조회 등**

# 9. Security Considerations

이 섹션은 NETCONF의 보안 측면을 다루며, 인증, 권한 부여, 데이터 무결성 및 잠재적 위험에 대해 설명합니다.

**1. 일반 보안 모델**

🔹 **NETCONF 보안은 계층적으로 적용됨**

* 보안 고려사항은 **세 가지 수준**에서 적용됩니다:  
  1️ **전송 계층** (예: SSH, TLS, BEEP)  
  2️ **NETCONF 메시지 계층**  
  3️ **NETCONF 데이터 모델 계층**

🔹 **NETCONF 내에서는 권한 부여가 정의되지 않음**

* 구현자는 **강력한 권한 부여 모델**을 제공해야 합니다.
* 권한 부여는 외부 메커니즘(예: **AAA, SSH 키**)을 통해 실행될 수 있습니다.

🔹 **NETCONF 사용자와 다른 인터페이스의 매핑은 항상 1:1이 아님**

* 데이터 모델과 인증 방법이 다를 수 있습니다.
* 권한 부여는 **보안 전송 메커니즘**(예: SSH, BEEP)을 통해 시행될 수 있습니다.

**2. 위험 및 위협**

**2.1. 무단 접근**

* **구성 정보**는 민감하며, 전송 시 **암호화**되지 않거나 **무결성 검사가** 없으면 **도청**이나 **위조 공격**에 취약할 수 있습니다.
* 구성 정보에는 **비밀번호, 사용자 이름, 서비스 설명, 토폴로지 정보** 등이 포함될 수 있습니다.

**2.2. 권한 없는 구성 변경**

* **구성 변경 작업**은 잠재적으로 시스템에 큰 영향을 미칠 수 있습니다.
* 예를 들어, **부분적으로 완료된 접근 제어 목록**(ACL)이 **잘못된 권한**으로 실행되면 **시스템에 보안 위험**을 초래할 수 있습니다.

**3. 보안 요구 사항**

**3.1. 최소 요구 사항**

🔹 **기밀성 및 인증 옵션을 지원**

* **전송 계층**(예: SSH, BEEP 등)은 **기밀성 및 인증**을 제공해야 하며, **NETCONF** 자체는 이를 보장하지 않습니다.

🔹 **NETCONF 세션의 양측은 인증 정보**를 교환해야 하며, 이를 통해 각 요청에 대한 **권한을 확인**할 수 있습니다.

**3.2. 민감한 작업에 대한 권한 부여**

* 예를 들어, \*\*<copy-config>\*\*와 같은 작업은 중요하며, 해당 작업을 실행할 권한이 없는 사용자가 이를 실행하지 않도록 해야 합니다.
* \*\*<edit-config>\*\*와 같은 일반적인 구성 변경 작업도 **적절한 권한**을 가진 사용자만 실행할 수 있도록 제한해야 합니다.

**3.3. 락(lock) 기능과 그 위험성**

* **<lock>** 작업은 시스템이 **부분적으로 잠길 수 있는 기능**을 제공하지만, 이를 잘못 사용하면 **보안 위험**을 초래할 수 있습니다.
* 예를 들어, 다른 사용자가 설정한 **구성 잠금을 해제**할 수 있는 방법이 필요할 수 있습니다. 이를 해결하기 위한 방법으로는 **세션 관리**와 **인증 서버**에서의 처리가 고려될 수 있습니다.

NETCONF의 보안을 구현할 때는 **기밀성, 인증, 권한 부여**와 관련된 모든 면을 충분히 고려하고, 외부 보안 메커니즘과의 통합을 통해 **안전한 운영**이 이루어질 수 있도록 해야 합니다.

# 10. IANA Considerations

## 10.1 NETCONF XML Namespace

이 문서는 IETF XML 레지스트리 [[RFC3688](#_13.1_Normative_References)]에 NETCONF XML 네임스페이스 URI를 등록합니다.  
IANA는 다음 URI를 참조하도록 업데이트하였습니다:

* **URI**: urn:ietf:params:xml:ns:netconf:base:1.0
* **등록자 연락처**: IESG
* **XML**: 해당 URI는 XML 네임스페이스이므로 XML 파일은 없음.

## 10.2 NETCONF XML Schema

이 문서는 IETF XML 레지스트리 [[RFC3688](#_13.1_Normative_References)]에 NETCONF XML 스키마 URI를 등록합니다.  
IANA는 다음 URI를 참조하도록 업데이트하였습니다:

* **URI**: urn:ietf:params:xml:schema:netconf
* **등록자 연락처**: IESG
* **XML**: 이 문서의 [Appendix B](#_Appendix_B._)에 XML 스키마 제공.

## 10.3 NETCONF YANG Module

이 문서는 YANG 모듈을 YANG 모듈 이름 레지스트리 [[RFC6020](#_13.1_Normative_References)]에 등록합니다:

* **이름**: ietf-netconf
* **네임스페이스**: urn:ietf:params:xml:ns:netconf:base:1.0
* **접두사**: nc
* **참조**: RFC 6241(본 문서)

## 10.4 NETCONF Capability URNs

IANA는 "네트워크 구성 프로토콜 (NETCONF) 기능 URN" 레지스트리를 생성하여 NETCONF 기능 식별자를 관리합니다. 레지스트리 추가는 IETF 표준 동작에 의해 요구됩니다.  
IANA는 이 문서의 참조를 업데이트하여 다음과 같은 기능 URN을 추가했습니다:

**기존 기능들**:

* **:writable-running**: urn:ietf:params:netconf:capability:writable-running:1.0
* **:candidate**: urn:ietf:params:netconf:capability:candidate:1.0
* **:rollback-on-error**: urn:ietf:params:netconf:capability:rollback-on-error:1.0
* **:startup**: urn:ietf:params:netconf:capability:startup:1.0
* **:url**: urn:ietf:params:netconf:capability:url:1.0
* **:xpath**: urn:ietf:params:netconf:capability:xpath:1.0

**새로 추가된 기능들**:

* **:base:1.1**: urn:ietf:params:netconf:base:1.1
* **:confirmed-commit:1.1**: urn:ietf:params:netconf:capability:confirmed-commit:1.1
* **:validate:1.1**: urn:ietf:params:netconf:capability:validate:1.1

# 11. Contributors(기여자)

편집자 외에도 이 문서는 다음 기여자들에 의해 작성되었습니다:

* **Ken Crozier**, Cisco Systems
* **Ted Goddard**, IceSoft
* **Eliot Lear**, Cisco Systems
* **Phil Shafer**, Juniper Networks
* **Steve Waldbusser**
* **Margaret Wasserman**, Painless Security, LLC

# 12. Acknowledgements

감사의 말.

# 13. References

## 13.1 Normative References

[RFC 2119](https://datatracker.ietf.org/doc/html/rfc2119) Key words for use in RFCs to Indicate Requirement Levels

[RFC 3553](https://datatracker.ietf.org/doc/html/rfc3553) An IETF URN Sub-namespace for Registered Protocol Parameters

[RFC 3629](https://datatracker.ietf.org/doc/html/rfc3629) UTF-8, a transformation format of ISO 10646

[RFC 3688](https://datatracker.ietf.org/doc/html/rfc3688) The IETF XML Registry

[RFC 3986](https://datatracker.ietf.org/doc/html/rfc3986) Uniform Resource Identifier (URI): Generic Syntax

[RFC 5717](https://datatracker.ietf.org/doc/html/rfc5717) Partial Lock Remote Procedure Call (RPC) for NETCONF

[RFC 6020](https://datatracker.ietf.org/doc/html/rfc6020) YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)

[RFC 6021](https://datatracker.ietf.org/doc/html/rfc6021) Common YANG Data Types

[RFC 6242](https://datatracker.ietf.org/doc/html/rfc6242) Using the NETCONF Protocol over Secure Shell (SSH)

[W3C.REC-xml-20001006](https://www.w3.org/TR/xml/) Extensible Markup Language (XML) 1.0 (Fifth Edition)

[W3C.REC-xpath-19991116](https://www.w3.org/TR/1999/REC-xpath-19991116/) XML Path Language (XPath) Version 1.0

## 13.2 Informative References

[RFC 2865](https://datatracker.ietf.org/doc/html/rfc2865) Remote Authentication Dial In User Service (RADIUS)

[RFC 3470](https://datatracker.ietf.org/doc/html/rfc3470) Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols

[RFC 4251](https://datatracker.ietf.org/doc/html/rfc4251) The Secure Shell (SSH) Protocol Architecture

[RFC 4741](https://datatracker.ietf.org/doc/html/rfc4741) NETCONF Configuration Protocol

[RFC 5246](https://datatracker.ietf.org/doc/html/rfc5246) The Transport Layer Security (TLS) Protocol

[W3C.REC-xslt-19991116](https://www.w3.org/TR/1999/REC-xslt-19991116) XSL Transformations (XSLT) Version 1.0

# Appendix A. NETCONF Error List

각 오류 태그에 대해 유효한 오류 유형과 오류 심각도 값이 나열되며, 필수 오류 정보가 있을 경우 이를 제공합니다.

* **오류 태그:** in-use  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청이 이미 사용 중인 리소스를 요구합니다.
* **오류 태그:** invalid-value  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청이 하나 이상의 매개변수에 대해 허용되지 않는 값을 지정했습니다.
* **오류 태그:** too-big  
  **오류 유형:** 전송, RPC, 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청 또는 응답(생성될 수 있는 응답)이 구현에서 처리할 수 없을 정도로 큽니다.
* **오류 태그:** missing-attribute  
  **오류 유형:** rpc, 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:**<bad-attribute>: 누락된 속성의 이름  
  <bad-element>: 누락된 속성이 포함되어야 하는 요소의 이름  
  **설명:** 예상된 속성이 누락되었습니다.
* **오류 태그:** bad-attribute  
  **오류 유형:** rpc, 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:**<bad-attribute>: 잘못된 값을 가진 속성의 이름  
  <bad-element>: 잘못된 값을 가진 속성이 포함된 요소의 이름  
  **설명:** 속성 값이 올바르지 않습니다; 예: 잘못된 유형, 범위 초과, 패턴 불일치 등.
* **오류 태그:** unknown-attribute  
  **오류 유형:** rpc, 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:**<bad-attribute>: 예상하지 않은 속성의 이름  
  <bad-element>: 예상하지 않은 속성이 포함된 요소의 이름  
  **설명:** 예상하지 않은 속성이 존재합니다.
* **오류 태그:** missing-element  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** <bad-element>: 누락된 요소의 이름  
  **설명:** 예상된 요소가 누락되었습니다.
* **오류 태그:** bad-element  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** <bad-element>: 잘못된 값을 가진 요소의 이름  
  **설명:** 요소 값이 올바르지 않습니다; 예: 잘못된 유형, 범위 초과, 패턴 불일치 등.
* **오류 태그:** unknown-element  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** <bad-element>: 예상하지 않은 요소의 이름  
  **설명:** 예상하지 않은 요소가 존재합니다.
* **오류 태그:** unknown-namespace  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:**<bad-element>: 예상하지 않은 네임스페이스를 포함한 요소의 이름  
  <bad-namespace>: 예상하지 않은 네임스페이스의 이름  
  **설명:** 예상하지 않은 네임스페이스가 존재합니다.
* **오류 태그:** access-denied  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청된 프로토콜 작업 또는 데이터 모델에 대한 액세스가 거부되었습니다. 이는 인증 실패 때문입니다.
* **오류 태그:** lock-denied  
  **오류 유형:** 프로토콜  
  **오류 심각도:** 오류  
  **오류 정보:** <session-id>: 요청된 잠금을 보유한 세션 ID, NETCONF가 아닌 엔터티가 잠금을 보유하는 경우 0을 지정  
  **설명:** 요청된 잠금에 대한 액세스가 다른 엔터티가 잠금을 보유하고 있기 때문에 거부되었습니다.
* **오류 태그:** resource-denied  
  **오류 유형:** 전송, RPC, 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청을 완료할 수 없었습니다. 이는 리소스 부족 때문입니다.
* **오류 태그:** rollback-failed  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 어떤 구성을 롤백하려는 요청(rollback-on-error 또는 <discard-changes> 작업)이 어떤 이유로 인해 완료되지 않았습니다.
* **오류 태그:** data-exists  
  **오류 유형:** 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청을 완료할 수 없었습니다. 관련 데이터 모델 콘텐츠가 이미 존재하기 때문입니다. 예를 들어, "create" 작업이 이미 존재하는 데이터에 대해 시도되었습니다.
* **오류 태그:** data-missing  
  **오류 유형:** 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청을 완료할 수 없었습니다. 관련 데이터 모델 콘텐츠가 존재하지 않기 때문입니다. 예를 들어, "delete" 작업이 존재하지 않는 데이터에 대해 시도되었습니다.
* **오류 태그:** operation-not-supported  
  **오류 유형:** 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청을 완료할 수 없었습니다. 요청된 작업이 이 구현에서 지원되지 않기 때문입니다.
* **오류 태그:** operation-failed  
  **오류 유형:** rpc, 프로토콜, 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 요청을 완료할 수 없었습니다. 요청된 작업이 어떤 이유로 실패했습니다. 이 오류는 다른 오류 조건에 의해 다루어지지 않은 경우입니다.
* **오류 태그:** partial-operation  
  **오류 유형:** 애플리케이션  
  **오류 심각도:** 오류  
  **오류 정보:**<ok-element>: 요청된 작업이 해당 노드 및 모든 하위 노드에 대해 완료된 데이터 모델의 요소를 식별합니다. 이 요소는 <error-info> 컨테이너에 0개 이상 포함될 수 있습니다.  
  <err-element>: 요청된 작업이 실패한 데이터 모델의 요소를 식별합니다. 이 요소는 <error-info> 컨테이너에 0개 이상 포함될 수 있습니다.  
  <noop-element>: 요청된 작업이 해당 노드 및 모든 하위 노드에 대해 시도되지 않은 데이터 모델의 요소를 식별합니다. 이 요소는 <error-info> 컨테이너에 0개 이상 포함될 수 있습니다.  
  **설명:** 이 오류 태그는 더 이상 사용되지 않으며, 이 문서에 따라 NETCONF 서버에서 전송되지 않아야 합니다.  
  요청된 작업의 일부가 실패하거나 시도되지 않은 이유로 전체 정리가 수행되지 않았습니다(예: 롤백이 지원되지 않음). <error-info> 컨테이너는 요청된 작업이 성공한 요소(<ok-element>), 실패한 요소(<err-element>), 또는 시도되지 않은 요소(<noop-element>)를 식별하는 데 사용됩니다.
* **오류 태그:** malformed-message  
  **오류 유형:** rpc  
  **오류 심각도:** 오류  
  **오류 정보:** 없음  
  **설명:** 메시지가 잘못 형성되어 처리할 수 없었습니다. 예를 들어, 메시지가 잘못된 XML 형식이거나 잘못된 문자 집합을 사용하고 있습니다.  
  이 오류 태그는 :base:1.1에서 새로 추가되었으며, 이전 클라이언트에게 전송되지 않아야 합니다.

# Appendix B. XML Schema for NETCONF Messages Layer

이 섹션은 **표준(normative)** 입니다.

NETCONF 메시지 계층의 XML 스키마는 **NETCONF 프로토콜의 메시지 형식을 정의**하며, **hello**, **rpc**, **rpc-reply** 등의 메시지를 포함합니다.

**XML 스키마 개요**

이 스키마는 **RFC에서 정의된 표준 NETCONF 메시지 구조**를 따릅니다. 주된 목적은 NETCONF 프로토콜의 메시지 형식을 엄격하게 정의하여, **클라이언트와 서버 간 통신이 일관되게 이루어지도록 하는 것**입니다.

**주요 요소**

* **hello**: NETCONF 세션 초기화 시 교환되는 메시지
* **rpc**: 클라이언트가 서버에 요청을 보낼 때 사용
* **rpc-reply**: 서버가 클라이언트 요청에 응답할 때 사용

**XML 스키마 예제 (netconf.xsd)**

다음은 \*\*NETCONF 메시지를 정의하는 XML 스키마(netconf.xsd)\*\*의 주요 내용입니다:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**설명**

1. **NETCONF hello 메시지**
   * **capabilities**: 클라이언트 또는 서버의 지원 기능을 나타냄
   * **session-id**: NETCONF 세션 ID(서버에서 제공)
2. **NETCONF rpc 요청**
   * 클라이언트가 서버에 명령을 보낼 때 사용
   * message-id 속성을 포함하여 요청을 추적할 수 있음
3. **NETCONF rpc-reply 응답**
   * ok 요소: 성공적인 응답
   * rpc-error 요소: 오류 응답 포함 가능
   * rpcResponse: 실제 응답 데이터 포함 가능
4. **NETCONF rpc-error 오류 메시지**
   * error-type: 오류 유형 (예: transport, rpc, protocol, application)
   * error-tag: 특정 오류 코드 (예: invalid-value, missing-attribute)
   * error-severity: 심각도 (error 또는 warning)
   * error-message: 오류 설명 메시지 (선택적)

**결론**

NETCONF 메시지 계층을 위한 XML 스키마는 **클라이언트와 서버 간의 표준화된 메시지 구조를 정의**합니다. 이를 통해 **프로토콜의 일관성을 유지하고, 오류를 명확하게 처리할 수 있도록 보장**합니다.

이 XML 스키마(netconf.xsd)는 **RFC 6241**에서 정의된 NETCONF 프로토콜과 연계되며, 다양한 네트워크 장비 및 관리 시스템에서 **NETCONF 구현의 기본 기준**이 됩니다.

<**netconf.xsd**>

.xsd 확장자는 **XML Schema Definition**(XML 스키마 정의) 파일을 나타냅니다. XML 스키마는 XML 문서의 구조와 내용에 대한 규칙을 정의하는 데 사용됩니다. 즉, XML 데이터가 특정 형식이나 구조를 따르도록 강제하는 메타데이터 파일입니다.

XML Schema는 XML 문서가 어떻게 구성되어야 하는지 규정합니다. 스키마는 XML 문서의 요소와 속성, 데이터 유형, 필수 여부 등을 정의하여 데이터의 유효성을 검사할 수 있게 합니다. XML 스키마는 \*\*XSD (XML Schema Definition)\*\*로도 불립니다.

**XSD 파일의 주요 목적**

1. **유효성 검사**: XML 문서가 특정 스키마를 준수하는지 검사할 수 있습니다. 즉, XML 문서가 올바른 형식인지, 필수 데이터가 포함되어 있는지 등을 검사합니다.
2. **구조 정의**: XML 문서에서 어떤 요소가 존재할 수 있는지, 그 요소들이 어떤 데이터를 포함할 수 있는지, 그리고 그 데이터의 데이터 유형을 정의합니다.
3. **데이터 타입 정의**: XSD는 XML 문서의 데이터를 다양한 데이터 타입 (예: string, integer, boolean, date, decimal 등)으로 정의할 수 있습니다.

**XSD 파일 검증**

XML 문서가 XSD 스키마를 준수하는지 검증하려면, XML 파서와 XSD 검증 도구를 사용할 수 있습니다. 예를 들어, xmllint라는 도구를 사용하여 XML 문서를 XSD 파일과 비교하여 유효성을 검사할 수 있습니다.

XSD 파일은 XML 문서를 정확하게 정의하고 검증할 수 있는 강력한 도구로, XML을 많이 사용하는 시스템이나 웹 서비스에서 필수적인 요소입니다.

# Appendix C. YANG Module for NETCONF Protocol Operations

Appendix C는 **NETCONF 프로토콜 작업을 위한 YANG 모듈**에 대해 설명합니다. YANG(또는 Yet Another Next Generation)은 네트워크 프로토콜에서 사용되는 구성 및 상태 데이터를 모델링하는 데 사용되는 **데이터 모델링 언어**입니다. NETCONF(Network Configuration Protocol)는 YANG 모델을 통해 네트워크 장치의 구성과 상태를 관리합니다.

**ietf-netconf YANG 모듈**은 NETCONF 프로토콜 작업에 필요한 데이터 구조를 정의합니다. YANG 모듈은 다양한 시스템에서 일관된 방식으로 구성을 표현하고 다룰 수 있도록 돕습니다.

**YANG 모듈 개요**

* **ietf-netconf YANG 모듈**은 hello, rpc, rpc-reply, edit-config와 같은 NETCONF 프로토콜 작업에 필요한 **데이터 구조를 정의**합니다.
* YANG 모듈을 사용하면 네트워크 장치에서 수행된 작업과 구성이 일관되게 표현되고 조작될 수 있습니다.

**RFC 6021에서 타입 정의 가져오기**

**ietf-netconf YANG 모듈**은 **RFC 6021**에서 타입 정의를 가져옵니다. RFC 6021은 YANG에서 사용하는 기본 데이터 유형과 표준 정의를 제공하며, 이는 YANG 모듈 내에서 재사용됩니다. 이를 통해 YANG 모듈은 다른 YANG 모듈과 호환되도록 보장할 수 있습니다.

**YANG 모듈의 주요 구성 요소**

1. **모듈 헤더**
   * YANG 모듈은 메타데이터를 포함하는 모듈 헤더로 시작하며, 여기에는 모듈의 이름, 개정 사항, 네임스페이스 및 조직 등이 포함됩니다.  
     예시:  
     텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

     AI가 생성한 콘텐츠는 부정확할 수 있습니다.
2. **데이터 구조**
   * YANG은 데이터를 트리 구조로 모델링하며, rpc, rpc-reply, edit-config와 같은 요소들은 노드로 표현됩니다.
   * 여기에는 사용자가 구성하는 데이터와 장치가 유지하는 상태 데이터가 정의됩니다.  
     예시:  
     텍스트, 스크린샷, 폰트이(가) 표시된 사진

     AI가 생성한 콘텐츠는 부정확할 수 있습니다.
3. **작업(Operation)과 응답(Response)**
   * **ietf-netconf** YANG 모듈은 rpc, rpc-reply, edit-config와 같은 NETCONF 작업을 컨테이너, 리스트, 리프(leaf) 형태로 모델링합니다.
   * 작업은 입력 파라미터와 예상 출력 결과를 정의합니다.
   * 응답은 작업이 성공했는지 실패했는지를 모델링합니다.
4. **Typedefs(타입 정의) 및 Imports(가져오기)**
   * YANG 모듈은 공통 데이터 유형을 다시 정의하지 않고, 다른 RFC에서 정의된 유형을 가져와 사용할 수 있습니다. 이를 통해 YANG 모듈 간 호환성을 유지할 수 있습니다.  
     예시:  
     텍스트, 스크린샷, 폰트이(가) 표시된 사진

     AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**NETCONF 작업을 위한 YANG 예시 (기본 골격)**

다음은 **ietf-netconf YANG 모듈**이 어떻게 NETCONF 작업을 모델링하는지에 대한 기본적인 골격 예시입니다:

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

위 예시에서 YANG 모듈은 rpc와 rpc-reply 작업을 모델링하고, 각 작업에 대한 입력 및 출력 파라미터를 정의합니다. rpc는 NETCONF 프로토콜 작업을 나타내고, rpc-reply는 해당 작업의 응답을 모델링합니다.

**결론**

**ietf-netconf YANG 모듈**은 NETCONF 프로토콜을 구현하는 시스템에서 필요한 데이터와 작업을 모델링하는 데 중요한 역할을 합니다. YANG을 사용하면 구성 및 상태 데이터를 일관되게 표현하고 관리할 수 있으며, 이를 통해 다양한 네트워크 장치 간의 상호 운용성을 보장할 수 있습니다.

# Appendix D. Capability Template

이 비규범적(non-normative) 부록에서는 프로토콜 기능을 정의할 때 사용할 수 있는 템플릿을 제공합니다.  
일반적으로 YANG으로 작성된 데이터 모델은 별도로 프로토콜 기능을 정의할 필요가 없습니다. 이유는 YANG을 사용하면 데이터 모델과 선택적 기능(이른바 "features")을 자동으로 발표하는 기능이 제공되기 때문입니다.

그러나 YANG의 기능만으로 충분하지 않은 경우(예: 매개변수가 필요한 기능을 처리해야 하는 경우) 또는 YANG이 아닌 다른 데이터 모델링 언어를 사용할 때, 이 기능 템플릿을 활용할 수 있습니다.

**D.1. 기능명 (Capability Name) 템플릿**

**D.1.1. 개요 (Overview)**

해당 기능의 개요를 설명하는 섹션입니다.

**D.1.2. 종속성 (Dependencies)**

이 기능이 의존하는 다른 기능이나 요구사항을 명시하는 섹션입니다.

**D.1.3. 기능 식별자 (Capability Identifier)**

해당 기능은 다음과 같은 기능 문자열(capability string)을 사용하여 식별됩니다:



**D.1.4. 새로운 연산 (New Operations)**

이 기능이 새롭게 추가하는 연산(operation)을 정의하는 섹션입니다.

**D.1.4.1. <op-name>**

새로운 연산의 세부 사항을 설명하는 하위 섹션입니다.

**D.1.5. 기존 연산의 수정 (Modifications to Existing Operations)**

이 기능이 기존 연산을 변경하는 경우 해당 변경 사항을 설명하는 섹션입니다.

**D.1.5.1. <op-name>**

수정된 기존 연산의 세부 사항을 설명하는 하위 섹션입니다.

※ 이 기능이 기존 연산을 수정하지 않는다면 이 섹션을 생략할 수 있습니다.

**D.1.6. 다른 기능과의 상호작용 (Interactions with Other Capabilities)**

이 기능이 다른 기능과 어떻게 상호작용하는지를 설명하는 섹션입니다.

※ 이 기능이 다른 기능과 상호작용하지 않는다면 이 섹션을 생략할 수 있습니다.

**정리**

이 템플릿은 프로토콜 기능을 문서화할 때 사용할 수 있도록 제공됩니다.  
특히 YANG의 기본 기능으로는 충분하지 않은 경우(예: 매개변수가 필요한 기능을 정의할 때)나 YANG이 아닌 다른 데이터 모델링 언어를 사용할 때 활용할 수 있습니다.

**각 섹션의 역할**

* **개요 및 종속성**: 기능의 목적과 필요 조건 설명
* **식별자**: 기능을 나타내는 URI 제공
* **새로운 연산 및 기존 연산의 변경**: 기능이 추가하거나 변경하는 연산 명시
* **다른 기능과의 상호작용**: 기능 간의 관계 설명 (필요 시)

이렇게 정의된 기능 템플릿을 활용하면 네트워크 관리 프로토콜에서 새로운 기능을 체계적으로 문서화하고 구현할 수 있습니다.

# Appendix E. Configuring Multiple Devices with NETCONF

이 부록은 비규범적(non-normative) 내용으로, NETCONF를 사용하여 여러 네트워크 장치를 구성하는 방법을 설명합니다.

**E.1. 개별 장치에 대한 연산**

단일 장치에서 구성 변경을 수행하는 과정에는 여러 단계가 필요합니다.  
구성을 변경할 때 애플리케이션은 변경이 올바르게 적용되었는지 확인해야 하며, 장치의 정상적인 동작에 영향을 미치지 않아야 합니다.

**개별 장치를 보호하는 주요 단계**

1. **구성 잠금 설정** (Acquiring the Configuration Lock)
2. **현재 실행 중인 구성의 체크포인트 저장** (Checkpointing the Running Configuration)
3. **새로운 구성 로드 및 검증** (Loading and Validating the Incoming Configuration)
4. **실행 중인 구성 변경** (Changing the Running Configuration)
5. **새로운 구성 테스트** (Testing the New Configuration)
6. **구성 변경 영구 저장** (Making the Change Permanent)
7. **구성 잠금 해제** (Releasing the Configuration Lock)

이제 각 단계를 자세히 살펴보겠습니다.

**E.1.1. 구성 잠금 설정 (Acquiring the Configuration Lock)**

NETCONF를 통해 여러 소스로부터 동시에 구성이 변경될 수 있으므로,  
변경 중 충돌을 방지하기 위해 구성 잠금을 설정해야 합니다.

<lock> 연산을 사용하여 실행 중인 구성을 잠급니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

장치가 :candidate 기능을 지원하는 경우,  
실행 구성 대신 후보(candidate) 구성을 잠글 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**E.1.2. 현재 실행 중인 구성의 체크포인트 저장**

구성을 변경하기 전에 현재 실행 중인 구성을 백업(체크포인트)해야 합니다.  
이를 통해 변경이 실패하면 원래 상태로 복구할 수 있습니다.

<copy-config> 연산을 사용하여 실행 중인 구성을 파일로 저장할 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

구성을 복원하려면 <source>와 <target>을 반대로 설정하면 됩니다.

**E.1.3. 새로운 구성 로드 및 검증**

:candidate 기능이 지원되는 경우,  
새로운 구성을 실행 환경에 영향을 주지 않고 먼저 로드할 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

장치가 :validate:1.1 기능을 지원하면,  
후보 구성(candidate)이 자동으로 검증됩니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**E.1.4. 실행 중인 구성 변경**

구성이 안전하게 로드되고 검증되면 실행 환경에 반영할 수 있습니다.

:candidate 기능을 지원하는 경우 <commit> 연산을 사용하여 실행 구성으로 적용할 수 있습니다.  
자동 복구 기능을 추가하려면 <confirmed> 속성을 사용합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

:candidate 기능이 없는 경우,  
구성을 직접 실행 환경(running)에 적용해야 합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**E.1.5. 새로운 구성 테스트**

구성이 적용된 후, 장치가 올바르게 동작하는지 확인해야 합니다.  
구체적인 테스트 방법은 변경 내용에 따라 다르며, 몇 가지 예시는 다음과 같습니다.

* **핑(Ping) 테스트**: 장치가 네트워크에서 도달 가능한지 확인
* **라우팅 테이블 검사**: 네트워크 가용성이 변경되었는지 확인
* **특정 프로토콜 검사**: 예를 들어, BGP 피어가 정상적으로 설정되었는지 확인

**E.1.6. 구성 변경 영구 저장**

구성이 정상적으로 적용되었음을 확인한 후, 변경 사항을 영구적으로 저장해야 합니다.

:startup 기능을 지원하는 장치에서는 <copy-config> 연산을 사용하여 실행 구성을 시작(startup) 구성으로 복사할 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

:candidate 기능을 지원하고 확인된 커밋을 수행한 경우,  
타임아웃 전에 최종 <commit>을 보내야 합니다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**E.1.7. 구성 잠금 해제**

구성 변경이 완료되면, 잠금을 해제하여 다른 애플리케이션이 접근할 수 있도록 해야 합니다.

<unlock> 연산을 사용하여 실행 중인 구성의 잠금을 해제합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

:candidate 기능을 지원하는 경우,  
후보 구성(candidate)의 잠금도 해제해야 합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**정리**

이 문서는 NETCONF를 사용하여 단일 네트워크 장치에서 안전하게 구성을 변경하는 표준적인 방법을 설명합니다.  
이러한 절차를 따르면 충돌을 방지하고, 실패 시 복구할 수 있으며, 변경 사항을 신뢰할 수 있습니다.

**E.2. 다중 장치에서의 연산 (Operations on Multiple Devices)**

네트워크 구성 변경이 여러 장치에 걸쳐 수행되어야 할 경우,  
트랜잭션(transaction) 개념을 적용하여 신뢰성을 보장하는 것이 중요합니다.  
NETCONF 프로토콜은 이러한 트랜잭션 중심의 작업을 구축할 수 있는 충분한 기본 연산을 제공합니다.

완전한 다중 장치 트랜잭션을 구현하는 것은 비용이 많이 들지만,  
실패가 발생할 수 있는 가능성을 줄이고, 이를 효율적으로 관리할 수 있습니다.

**E.2.1. 다중 장치 연산의 두 가지 유형**

다중 장치 구성 변경에는 두 가지 주요 유형이 있습니다.

**1. 개별 장치의 실패를 허용하는 연산**

이 유형의 연산에서는 일부 장치에서 실패해도 전체 네트워크 구성이 원래 상태로 돌아갈 필요는 없습니다.  
실패한 장치는 이후 다시 시도하거나, 사용자에게 오류를 보고하면 됩니다.

🔹 **예시**

* NTP 서버 추가
  + 일부 장치에서 실패하더라도 나머지 장치에서는 정상적으로 업데이트 가능
  + 실패한 장치는 나중에 다시 시도할 수 있음

✅ **실패 방지 및 복구 방법**

* 개별 장치의 복구 절차 수행
* 실패 로그를 기록하고 관리자에게 알림
* 이후 다시 시도할 수 있도록 스케줄링

**2. 전체 네트워크 상태를 유지해야 하는 연산**

이 유형의 연산은 모든 장치에서 변경이 성공적으로 적용되거나,  
실패 시 전체 변경 사항이 원래 상태로 되돌려져야 합니다.

🔹 **예시**

* **VPN 변경 작업**
  + 여러 장치에서 설정이 변경되어야 하며, 일부 장치에서만 변경될 경우 서비스 오류 발생 가능
* **QoS (Quality of Service) 정책 추가**
  + 일부 장치에서만 새 정책이 적용되면, 이후 트래픽 처리에서 불일치가 발생할 수 있음

✅ **트랜잭션을 보장하는 방법**

* 모든 대상 장치에서 **구성 잠금(lock)** 유지
* 모든 장치에서 **구성 검증(validation)** 및 **체크포인트 생성(checkpointing)** 수행
* 모든 장치에서 변경 사항을 적용한 후, 최종적으로 **변경 사항을 확정(commit)**
* 실패 시 **모든 장치에서 원래 구성으로 복원(rollback)**
* 성공 또는 실패 여부에 따라 **구성 잠금 해제(unlock)**

**E.2.2. 다중 장치 트랜잭션을 위한 단계**

단일 장치 구성 변경(E.1)에서 수행한 단계를  
모든 장치에 병렬(parallel)로 적용해야 합니다.

**🔹 1. 모든 장치에서 구성 잠금 설정**

각 장치에서 동시에 구성 변경이 이루어지지 않도록,  
모든 대상 장치에서 <lock> 연산을 수행합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**🔹 2. 모든 장치에서 체크포인트 저장**

현재 실행 중인 구성을 백업하여,  
이후 실패 시 원래 상태로 복원할 수 있도록 합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

각 장치에서 개별적으로 체크포인트를 생성해야 합니다.

**🔹 3. 새로운 구성 로드 및 검증**

모든 장치에서 새로운 구성을 로드하고,  
구성이 올바르게 적용될 수 있는지 검증합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

검증이 필요하면 <validate> 연산을 수행할 수 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**🔹 4. 실행 중인 구성 변경**

모든 장치에서 새로운 구성을 실행 환경에 반영합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**🔹 5. 변경 사항 테스트**

변경 사항이 정상적으로 적용되었는지 확인합니다.  
이 단계는 개별 장치뿐만 아니라 **네트워크 전체의 동작**을 점검하는 것이 중요합니다.

테스트 예시:

* 모든 장치에서 ping 테스트 수행
* 네트워크 라우팅 테이블 비교
* 서비스 연속성 확인 (예: VPN 연결 테스트)

**🔹 6. 구성 변경 확정 또는 복원**

✅ **모든 장치에서 성공적으로 변경이 적용된 경우**

* 최종 <commit> 연산을 수행하여 변경 사항을 확정합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

❌ **일부 장치에서 실패한 경우**

* <copy-config> 연산을 사용하여 체크포인트에서 원래 구성으로 복원합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**🔹 7. 구성 잠금 해제**

모든 작업이 완료되면,  
각 장치에서 <unlock> 연산을 수행하여 구성 잠금을 해제합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

후보 구성(candidate)이 잠겨있는 경우,  
아래 명령어를 사용하여 해제해야 합니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**결론**

* 다중 장치 연산은 **개별 장치의 실패를 허용할지 여부**에 따라 접근 방식이 달라집니다.
* 중요한 변경 사항(VPN, QoS 등)은 **전체 적용되거나 전체 롤백**되어야 합니다.
* NETCONF의 <lock>, <copy-config>, <edit-config>, <commit>, <validate>, <unlock> 연산을 조합하여  
  트랜잭션 기반의 다중 장치 구성을 수행할 수 있습니다.

# Appendix F. Changes from RFC 4741

이 섹션에서는 본 문서와 **RFC 4741** 간의 주요 변경 사항을 정리합니다.

**🔹 주요 변경 내용**

1️ **새로운 오류 태그 추가**

* "malformed-message" 오류 태그 추가  
  → 메시지 형식이 잘못된 경우 사용됨

2️ **새로운 "operation" 속성 값 추가**

* "remove" 값 추가  
  → 기존 설정에서 특정 요소를 제거할 수 있도록 확장됨

3️ **이전 오류 태그 제거**

* "partial-operation" 오류 태그 사용 중단

4️ **<commit> 연산 기능 확장**

* <persist> 및 <persist-id> 매개변수 추가  
  → 장기 실행되는 커밋 작업을 지속적으로 관리 가능

5️ **기본 프로토콜 URI 업데이트**

* <hello> 메시지 교환 방식 명확화  
  → 특정 세션에서 사용되는 기본 프로토콜 버전 식별 가능

6️ **YANG 모듈 추가 및 XSD 변경**

* NETCONF 연산을 모델링하기 위해 **YANG 모듈** 추가
* 기존 XSD에서 연산 계층 제거

7️ **후보 데이터 저장소(Candidate Datastore) 잠금(lock) 동작 명확화**

* <lock> 연산에 대한 동작 정의 개선

8️ **"delete" 연산 오류 응답 동작 명확화**

* "delete" 연산과 관련된 서버 오류 응답 요구 사항 정리

9️ **서브트리 필터링을 위한 네임스페이스 와일드카드 지원 추가**

* 네임스페이스 기반 필터링 기능 확장

🔟 **<edit-config> 연산의 <test-option> 매개변수 개선**

* "test-only" 값 추가  
  → 구성 변경을 실제로 적용하지 않고 검증만 수행 가능

1️1️ **새로운 연산 추가: <cancel-commit>**

* <commit> 연산을 취소하는 <cancel-commit> 연산 추가

1️2️ **NETCONF 사용자명(Username) 도입**

* NETCONF 프로토콜에서 사용자를 식별하기 위한 **사용자명(username)** 개념 추가
* 전송 프로토콜이 사용자명을 어떻게 파생(derivation)하는지 명확하게 정의해야 함

**📌 결론**

이 변경 사항들은 **NETCONF 프로토콜의 유연성과 신뢰성을 향상**하기 위한 개선 사항입니다.  
특히 **YANG 기반 모델링, 트랜잭션 관리 개선, 필터링 기능 확장** 등의 변화가 눈에 띕니다.