ContainerLabs

K19G

2026-02-28

Table of contents

ContainerLab for CCNA/CCNP: Complete Network Simulation Guide

Welcome to the comprehensive guide for learning CCNA and CCNP-level networking concepts using Nokia's ContainerLab. This course combines theoretical knowledge with hands-on practical experience in a modern, containerized environment, covering everything from fundamental networking to advanced enterprise technologies.

What You'll Learn

This course provides complete coverage of both CCNA 200-301 and CCNP Enterprise exam objectives through practical implementation using ContainerLab:

CCNA Level (Modules 1-10):

- **Network Fundamentals**: Understanding modern networking concepts and protocols
- **Layer 2 Technologies**: Switching, VLANs, and spanning tree protocols
- **Layer 3 Technologies**: Routing protocols including OSPF and EIGRP
- **WAN Technologies**: Wide area networking and VPN implementations
- **Network Services**: DHCP, DNS, and NAT configuration
- **Security Fundamentals**: Network security principles and implementation
- **Automation**: Modern network automation and programmability concepts

CCNP Level (Modules 11-20):

- **Advanced Routing**: BGP, advanced OSPF/EIGRP, route manipulation
- **Advanced Switching**: MST, advanced VLANs, Layer 3 switching
- **Quality of Service**: Traffic classification, queuing, and policy implementation
- **Multicast**: IGMP, PIM, and multicast routing protocols
- **Service Provider**: MPLS, VPNs, and carrier-grade technologies
- **Data Center**: Modern data center networking and SDN concepts
- **Advanced Security**: Enterprise security, VPNs, and monitoring
- **Network Automation**: Python, Ansible, NETCONF/RESTCONF, CI/CD
- **Cloud Networking**: Hybrid cloud, containers, and Kubernetes networking
- **Emerging Technologies**: 5G, IoT, AI/ML in networking

Why ContainerLab?

ContainerLab offers several advantages for network learning:

- **Cost-Effective**: No expensive hardware required
- **Scalable**: Create complex topologies limited only by system resources
- **Realistic**: Production-like network behavior and features
- **Modern**: Integrates with DevOps and automation workflows
- **Multi-Vendor**: Support for Cisco, Arista, Juniper, Nokia, and more

Course Structure

The course is organized into 20 comprehensive modules spanning both CCNA and CCNP levels:

CCNA Track (Modules 1-10):

1. **Foundation and Setup** - Getting started with ContainerLab
2. **Network Fundamentals** - Core networking concepts and topologies
3. **Layer 2 Technologies** - Switching and VLAN implementation
4. **Layer 3 Technologies** - Routing protocols and configuration
5. **WAN Technologies** - Wide area networking concepts
6. **Network Services** - Essential network services
7. **Network Security** - Security fundamentals and implementation
8. **Monitoring and Troubleshooting** - Network operations and maintenance
9. **Automation and Programmability** - Modern network automation
10. **Advanced Topics** - Cloud integration and future technologies

CCNP Track (Modules 11-20):

11. **Advanced Routing** - BGP, advanced OSPF/EIGRP features
12. **Advanced Switching** - MST, advanced VLANs, Layer 3 switching
13. **Quality of Service** - Comprehensive QoS implementation
14. **Multicast Networking** - Multicast protocols and applications
15. **Advanced Security** - Enterprise security and monitoring
16. **Service Provider Technologies** - MPLS and carrier services
17. **Data Center Networking** - Modern data center concepts
18. **Advanced Automation** - Python, Ansible, CI/CD pipelines
19. **Cloud and Hybrid Networking** - Cloud integration and containers
20. **Emerging Technologies** - Future networking trends

Open Source Track (Module 21):

21. **Open Source Networking Solutions** - FRR, VyOS, OpenWrt, Mininet, and other FOSS networking tools

Prerequisites

- Basic understanding of Linux command line
- Fundamental networking concepts (OSI model, TCP/IP)
- System with Docker support (8GB+ RAM recommended)

Getting Started

1. Review the complete Course Syllabus
2. Set up your environment following Chapter 2: Installation and Environment Setup
3. Begin with Chapter 1: Introduction to ContainerLab

Hands-On Approach

Every chapter includes: - Detailed theoretical explanations - Practical lab exercises - Real-world configuration examples - Troubleshooting scenarios - Review questions and exercises

Support and Community

- Weekly office hours for questions
- Online discussion forums
- Lab assistance sessions
- Peer study groups

Acknowledgments

Special thanks to Nokia for developing ContainerLab and making it available as an open-source tool for the networking community. This course builds upon the excellent foundation they've provided for modern network simulation and learning.

Ready to begin your journey into modern network simulation? Let's start with the Course Syllabus to understand the complete learning path ahead.

Intro

ContainerLab for CCNA/CCNP: Complete Network Simulation Course

Course Overview

This comprehensive course covers network automation and simulation using Nokia's ContainerLab tool, designed for both CCNA and CCNP-level networking concepts. Students will learn to build, configure, and manage virtual network topologies while mastering fundamental through advanced networking principles, including enterprise-level routing, switching, and network services.

Prerequisites

- Basic understanding of Linux command line
- Fundamental networking concepts (OSI model, TCP/IP)
- Basic familiarity with Docker containers (helpful but not required)

Learning Objectives

By the end of this course, students will be able to:

1. Install, configure, and manage ContainerLab environments
2. Create complex network topologies using various network operating systems
3. Implement and troubleshoot CCNA-level networking protocols
4. Automate network configurations using modern tools
5. Perform network monitoring and analysis
6. Apply security best practices in containerized network environments

Course Structure

Module 1: Foundation and Setup

Duration: 2 weeks

Chapter 1: Introduction to ContainerLab

- What is ContainerLab and why use it?
- Comparison with other network simulation tools
- Use cases and industry applications
- Architecture overview

Chapter 2: Installation and Environment Setup

- System requirements and prerequisites
- Installing Docker and ContainerLab
- Setting up the development environment
- Troubleshooting common installation issues

Chapter 3: Basic ContainerLab Operations

- Understanding topology files
- Creating your first lab
- Basic CLI commands and operations
- Lab lifecycle management

Module 2: Network Fundamentals in ContainerLab

Duration: 3 weeks

Chapter 4: Network Topologies and Design

- Topology file structure and syntax
- Creating point-to-point connections
- Building complex multi-node topologies
- Best practices for topology design

Chapter 5: Supported Network Operating Systems

- Overview of supported NOSs
- Cisco IOS-XE and IOS-XR
- Arista EOS
- Juniper vMX and vSRX
- Nokia SR OS
- Open source alternatives (FRR, VyOS)

Chapter 6: Container Networking Fundamentals

- Docker networking basics
- Bridge networks and VLANs
- Network namespaces
- Inter-container communication

Module 3: Layer 2 Technologies

Duration: 2 weeks

Chapter 7: Ethernet and Switching Fundamentals

- Ethernet frame structure
- MAC address learning and forwarding
- Switch configuration in ContainerLab
- Collision and broadcast domains

Chapter 8: VLANs and Trunking

- VLAN concepts and implementation
- Configuring VLANs across different NOSs
- Trunk ports and VLAN tagging
- Inter-VLAN routing setup

Chapter 9: Spanning Tree Protocol (STP)

- STP fundamentals and loop prevention
- Configuring STP in virtual environments
- Rapid STP and Multiple STP
- Troubleshooting STP issues

Module 4: Layer 3 Technologies

Duration: 3 weeks

Chapter 10: IP Addressing and Subnetting

- IPv4 addressing fundamentals
- Subnetting and VLSM
- IPv6 basics and addressing
- IP configuration in ContainerLab

Chapter 11: Static and Dynamic Routing

- Static routing configuration
- Default routes and route summarization
- Dynamic routing protocol overview
- Route selection and administrative distance

Chapter 12: OSPF Configuration and Troubleshooting

- OSPF fundamentals and areas
- Single-area OSPF configuration
- Multi-area OSPF design
- OSPF troubleshooting techniques

Chapter 13: EIGRP Implementation

- EIGRP concepts and metrics
- EIGRP configuration and verification
- Load balancing and route summarization
- EIGRP troubleshooting

Module 5: WAN Technologies

Duration: 2 weeks

Chapter 14: WAN Fundamentals

- WAN technologies overview
- Point-to-point connections
- Frame Relay concepts
- MPLS basics

Chapter 15: VPN Technologies

- VPN fundamentals
- Site-to-site VPN configuration
- GRE tunnels
- IPSec implementation

Module 6: Network Services and Applications

Duration: 2 weeks

Chapter 16: DHCP and DNS Services

- DHCP server configuration
- DHCP relay and helper addresses
- DNS fundamentals
- Implementing DNS in ContainerLab

Chapter 17: Network Address Translation (NAT)

- NAT concepts and types
- Static and dynamic NAT configuration
- PAT (Port Address Translation)
- NAT troubleshooting

Chapter 18: Access Control Lists (ACLs)

- ACL fundamentals and types
- Standard and extended ACLs
- Named ACLs
- ACL placement and best practices

Module 7: Network Security

Duration: 2 weeks

Chapter 19: Network Security Fundamentals

- Security threats and vulnerabilities
- Defense in depth strategy
- Physical and logical security
- Security policies and procedures

Chapter 20: Switch and Router Security

- Device hardening techniques
- Password security and encryption
- SSH configuration
- Port security implementation

Chapter 21: Wireless Security

- Wireless security protocols
- WPA/WPA2/WPA3 configuration
- Enterprise wireless security
- Wireless troubleshooting

Module 8: Network Monitoring and Troubleshooting

Duration: 2 weeks

Chapter 22: Network Monitoring Tools

- SNMP fundamentals
- Syslog configuration
- Network monitoring with ContainerLab
- Performance monitoring tools

Chapter 23: Troubleshooting Methodologies

- Systematic troubleshooting approach
- OSI model troubleshooting
- Common network issues
- Documentation and change management

Chapter 24: Advanced Troubleshooting Tools

- Packet capture and analysis
- Network debugging commands
- Log analysis techniques
- Performance optimization

Module 9: Automation and Programmability

Duration: 2 weeks

Chapter 25: Network Automation Fundamentals

- Introduction to network automation
- APIs and programmability
- Configuration management
- Automation tools overview

Chapter 26: Scripting and Configuration Management

- Basic scripting for network tasks
- Configuration templates
- Version control for network configs
- Automated testing

Module 10: Advanced Topics and Integration

Duration: 1 week

Chapter 27: Cloud Integration

- Cloud networking concepts
- Hybrid cloud connectivity
- Container orchestration
- DevOps practices for networking

Chapter 28: Future Technologies

- Software-Defined Networking (SDN)
- Network Function Virtualization (NFV)
- Intent-based networking
- AI/ML in networking

Assessment Methods

Practical Labs (60%)

- Hands-on lab exercises for each chapter
- Progressive complexity building on previous concepts
- Real-world scenario implementations

Projects (25%)

- Mid-term project: Design and implement a campus network
- Final project: Complete enterprise network with security and monitoring

Quizzes and Exams (15%)

- Weekly quizzes on theoretical concepts
- Midterm and final examinations
- Troubleshooting scenarios

Required Resources

Software

- ContainerLab (latest version)
- Docker Desktop or Docker Engine
- Text editor (VS Code recommended)
- Git for version control

Hardware Requirements

- Minimum 8GB RAM (16GB recommended)
- 50GB available disk space
- Multi-core processor (4+ cores recommended)
- Stable internet connection

Reference Materials

- Official ContainerLab documentation
- Cisco CCNA Official Cert Guide
- Network simulation best practices guides
- Vendor-specific configuration guides

Module 11: Advanced Routing (CCNP Level)

Duration: 3 weeks

Chapter 29: Advanced OSPF Features

- OSPF LSA types and database optimization
- OSPF areas: stub, totally stubby, NSSA
- OSPF virtual links and authentication
- OSPF performance tuning and scalability

Chapter 30: Advanced EIGRP Configuration

- EIGRP for IPv6 and named mode
- EIGRP authentication and security
- EIGRP load balancing and optimization
- EIGRP troubleshooting and convergence

Chapter 31: BGP Fundamentals and Configuration

- BGP concepts and operation
- eBGP and iBGP configuration
- BGP path selection and attributes
- BGP route filtering and manipulation

Chapter 32: Advanced BGP Features

- BGP route reflectors and confederations
- BGP communities and extended communities
- BGP security and best practices
- BGP troubleshooting methodologies

Module 12: Advanced Switching (CCNP Level)

Duration: 3 weeks

Chapter 33: Advanced STP and MST

- Multiple Spanning Tree (MST) configuration
- STP optimization and tuning
- STP security features
- Loop guard and root guard

Chapter 34: Advanced VLAN Features

- Private VLANs and port isolation
- VLAN Trunking Protocol (VTP)
- Dynamic VLAN assignment
- Voice VLANs and QoS integration

Chapter 35: Layer 3 Switching and SVIs

- Inter-VLAN routing optimization
- Switch Virtual Interfaces (SVIs)
- Routed ports and Layer 3 EtherChannels
- HSRP, VRRP, and GLBP configuration

Chapter 36: Advanced EtherChannel

- LACP and PAgP advanced features
- Layer 3 EtherChannel configuration
- EtherChannel load balancing
- EtherChannel troubleshooting

Module 13: Quality of Service (QoS)

Duration: 2 weeks

Chapter 37: QoS Fundamentals

- QoS concepts and requirements
- Traffic classification and marking
- Queuing mechanisms and scheduling
- Traffic shaping and policing

Chapter 38: Advanced QoS Implementation

- Modular QoS CLI (MQC)
- Class-based weighted fair queuing
- Low latency queuing (LLQ)
- QoS for voice and video

Module 14: Multicast Networking

Duration: 2 weeks

Chapter 39: Multicast Fundamentals

- Multicast concepts and addressing
- IGMP configuration and optimization
- Multicast forwarding and RPF
- Multicast troubleshooting

Chapter 40: Advanced Multicast Protocols

- PIM sparse mode and dense mode
- Rendezvous Point (RP) configuration
- Multicast Source Discovery Protocol
- Anycast RP and BSR

Module 15: Advanced Security

Duration: 3 weeks

Chapter 41: Advanced Access Control

- Object-group ACLs and optimization
- Time-based and reflexive ACLs
- Zone-based firewalls
- Application inspection and control

Chapter 42: Advanced VPN Technologies

- DMVPN configuration and optimization
- FlexVPN implementation
- SSL VPN and remote access
- VPN troubleshooting and monitoring

Chapter 43: Network Security Monitoring

- SIEM integration and log analysis
- Network behavior analysis
- Threat detection and response
- Security automation and orchestration

Module 16: Service Provider Technologies

Duration: 2 weeks

Chapter 44: MPLS Fundamentals

- MPLS concepts and label switching
- LDP configuration and operation
- MPLS VPN basics
- MPLS traffic engineering

Chapter 45: Service Provider Services

- Layer 3 VPN implementation
- Layer 2 VPN services
- QoS in service provider networks
- Service provider security

Module 17: Data Center Networking

Duration: 2 weeks

Chapter 46: Data Center Fundamentals

- Data center architecture and design
- Fabric technologies and protocols
- Storage networking basics
- Data center virtualization

Chapter 47: Software-Defined Networking

- SDN concepts and architectures
- OpenFlow and controller technologies
- Network programmability
- Intent-based networking

Module 18: Advanced Automation and Orchestration

Duration: 2 weeks

Chapter 48: Advanced Network Automation

- Ansible for network automation
- Python scripting for networking
- NETCONF and RESTCONF protocols
- Network CI/CD pipelines

Chapter 49: Network Orchestration

- Infrastructure as Code (IaC)
- Container orchestration for networking
- Microservices architecture
- DevNetOps practices

Module 19: Cloud and Hybrid Networking

Duration: 2 weeks

Chapter 50: Cloud Networking Fundamentals

- Public cloud networking concepts
- AWS, Azure, and GCP networking
- Hybrid cloud connectivity
- Cloud security considerations

Chapter 51: Container and Kubernetes Networking

- Container networking models
- Kubernetes networking concepts
- Service mesh technologies
- Cloud-native network security

Module 20: Emerging Technologies and Future Trends

Duration: 1 week

Chapter 52: Emerging Network Technologies

- 5G and edge computing
- IoT networking requirements
- AI/ML in network operations
- Quantum networking concepts

Module 21: Open Source Networking Solutions

Duration: 3 weeks

Chapter 53: FRRouting (FRR) - Open Source Routing Suite

- FRR architecture and daemon structure
- Multi-protocol routing (OSPF, BGP, IS-IS, RIP)
- Advanced routing features and optimization
- Integration with commercial equipment

Chapter 54: VyOS - Open Source Network Operating System

- VyOS configuration and management
- Routing, switching, and security features
- VPN services and high availability
- Firewall and NAT configuration

Chapter 55: OpenWrt - Open Source Wireless and Embedded Networking

- OpenWrt installation and configuration
- Wireless access point management
- Network services and QoS
- Custom firmware development

Chapter 56: Mininet - Network Emulation and SDN Testing

- Mininet topology creation
- OpenFlow controller integration
- SDN application development
- Network testing and validation

Chapter 57: BIRD Internet Routing Daemon

- BIRD configuration and management
- Advanced routing policies
- Route filtering and manipulation
- Performance optimization

Chapter 58: Strongswan VPN Solutions

- IPSec VPN configuration
- Certificate management
- High availability VPN setups
- Mobile and remote access VPNs

Chapter 59: pfSense and OPNsense Firewalls

- Open source firewall deployment
- Advanced security features
- VPN and routing capabilities
- High availability configurations

Chapter 60: Open Source Network Monitoring

- Nagios and Zabbix monitoring
- PRTG and LibreNMS alternatives
- Custom monitoring solutions
- Integration with network automation

Course Timeline

Total Duration: 35 weeks (9 months) **CCNA Track:** Modules 1-10 (20 weeks) **CCNP Track:** Modules 11-20 (12 weeks) **Open Source Track:** Module 21 (3 weeks) **Weekly Commitment:** 12-18 hours **Lab Sessions:** 4 hours per week **Self-Study:** 8-14 hours per week

Certification Preparation

CCNA 200-301 Alignment (Modules 1-10):

- Network Fundamentals (20%)
- Network Access (20%)
- IP Connectivity (25%)
- IP Services (10%)
- Security Fundamentals (15%)
- Automation and Programmability (10%)

CCNP Enterprise Alignment (Modules 11-20):

- **ENCOR 350-401**: Advanced routing, switching, wireless, and security
- **ENARSI 300-410**: Advanced routing and services implementation
- **ENWLSI 300-430**: Wireless implementation (covered in wireless modules)
- **ENSLD 300-420**: Data center networking fundamentals

Support and Resources

- Weekly office hours
- Online discussion forums
- Lab assistance sessions
- Peer study groups
- Industry mentor program

This syllabus is designed to provide comprehensive coverage of CCNA-level networking concepts using ContainerLab as the primary learning platform. The hands-on approach ensures practical skills development alongside theoretical understanding.

Introduction

Chapter 1: Introduction to ContainerLab

Learning Objectives

By the end of this chapter, you will be able to: - Understand what ContainerLab is and its core purpose - Compare ContainerLab with other network simulation tools - Identify key use cases and industry applications - Explain the architecture and components of ContainerLab

What is ContainerLab?

ContainerLab is an open-source network simulation platform developed by Nokia that enables network engineers to create complex, multi-vendor network topologies using containerized network operating systems. It provides a unified way to deploy, manage, and interact with virtual network labs.

Key Features

- **Multi-vendor support**: Works with various network operating systems including Cisco, Arista, Juniper, Nokia, and open-source alternatives
- **Container-based**: Leverages Docker containers for lightweight, fast deployment
- **Declarative configuration**: Uses YAML files to define network topologies
- **Automation-friendly**: Integrates well with CI/CD pipelines and automation tools
- **Realistic simulation**: Provides near-production network behavior

Why Use ContainerLab?

Traditional Challenges in Network Learning

Before ContainerLab, network engineers faced several challenges:

1. **Hardware costs**: Physical equipment is expensive and limited
2. **Scalability issues**: Difficult to create large topologies
3. **Time consumption**: Setting up labs takes significant time
4. **Maintenance overhead**: Physical equipment requires ongoing maintenance
5. **Limited flexibility**: Hard to quickly modify topologies

ContainerLab Solutions

ContainerLab addresses these challenges by providing:

```yaml
# Example: Simple two-router topology
name: basic-lab
topology:
  nodes:
    router1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
    router2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
  links:
    - endpoints: ["router1:eth1", "router2:eth1"]
```

Benefits: - **Cost-effective**: No physical hardware required - **Rapid deployment**: Labs start in minutes, not hours - **Infinite scalability**: Limited only by system resources - **Version control**: Topology files can be stored in Git - **Reproducible**: Same topology works across different environments

Comparison with Other Network Simulation Tools

GNS3

Feature	ContainerLab	GNS3
Deployment	Container-based	VM-based
Resource usage	Lower	Higher
Startup time	Faster	Slower
Multi-vendor	Native support	Requires images
Automation	CLI-first	GUI-first
Learning curve	Moderate	Steep

EVE-NG

Feature	ContainerLab	EVE-NG
Platform	Linux/macOS/Windows	Linux-based
Interface	CLI + Web	Web-based
Licensing	Open source	Freemium
Container support	Native	Limited
Scalability	High	Medium

Cisco Packet Tracer

Feature	ContainerLab	Packet Tracer
Realism	Production-like	Simplified
Vendor support	Multi-vendor	Cisco-focused
Advanced features	Full feature set	Educational subset
Target audience	Professional	Educational
Complexity	Real-world	Simplified

Use Cases and Industry Applications

1. Network Education and Training

CCNA/CCNP Preparation:

```yaml
# CCNA lab topology
name: ccna-lab
topology:
  nodes:
    sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
    sw2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
    r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
  links:
    - endpoints: ["sw1:eth1", "sw2:eth1"]
    - endpoints: ["sw1:eth2", "r1:eth1"]
```

2. Network Design and Testing

Pre-deployment validation: - Test configurations before production deployment - Validate network designs and architectures - Perform capacity planning and performance testing

3. Automation Development

CI/CD Integration:

```bash
# Automated testing pipeline
containerlab deploy -t topology.yml
```

```
ansible-playbook configure-network.yml
pytest network-tests/
containerlab destroy -t topology.yml
```

4. Troubleshooting and Learning

Scenario-based learning: - Create specific problem scenarios - Practice troubleshooting methodologies - Develop debugging skills

5. Vendor Evaluation

Multi-vendor comparison: - Test different vendor solutions - Compare feature implementations - Evaluate performance characteristics

Architecture Overview

Core Components

1. ContainerLab Engine

- **Topology parser**: Reads and validates YAML topology files
- **Container orchestrator**: Manages container lifecycle
- **Network manager**: Creates and manages virtual networks
- **State manager**: Tracks lab state and metadata

2. Container Runtime

- **Docker integration**: Uses Docker as the container runtime
- **Image management**: Handles NOS container images
- **Resource allocation**: Manages CPU, memory, and storage

3. Network Fabric

- **Virtual bridges**: Creates Layer 2 connectivity
- **Network namespaces**: Provides network isolation
- **Virtual interfaces**: Connects containers

System Architecture Diagram

```
                 ContainerLab CLI

                Topology Parser

              Container Orchestrator

                 Docker Runtime

              Host Operating System
```

Workflow Process

1. **Topology Definition**: Create YAML topology file
2. **Validation**: ContainerLab validates the topology
3. **Image Preparation**: Pull required container images
4. **Network Creation**: Create virtual networks and bridges
5. **Container Deployment**: Start containers with proper networking
6. **Configuration**: Apply initial configurations
7. **Monitoring**: Track lab status and health

Key Concepts and Terminology

Topology File

A YAML file that defines the complete network lab structure: - Nodes (network devices) - Links (connections between nodes) - Configuration parameters - Metadata and labels

Kinds

Predefined device types that ContainerLab supports: - `cisco_iosxe`: Cisco IOS-XE devices - `arista_eos`: Arista EOS switches - `nokia_sros`: Nokia Service Router OS - `linux`: Generic Linux containers

Links

Connections between network nodes: - Point-to-point links - Multi-access networks - VLAN configurations - Custom network parameters

Lab Lifecycle

The stages of a ContainerLab deployment: 1. **Deploy**: Create and start the lab 2. **Configure**: Apply device configurations 3. **Operate**: Use the lab for testing/learning 4. **Destroy**: Clean up and remove the lab

Benefits for CCNA Learning

1. Hands-on Practice

- Real device behavior and responses
- Authentic command-line interfaces
- Production-like troubleshooting scenarios

2. Scalable Learning Environment

- Start with simple topologies
- Gradually increase complexity
- Support for large-scale scenarios

3. Multi-vendor Exposure

- Experience different vendor implementations
- Understand protocol variations
- Develop vendor-neutral skills

4. Modern Workflow Integration

- Version control for lab configurations
- Automation and scripting practice
- DevOps methodology exposure

Getting Started Checklist

Before proceeding to the next chapter, ensure you understand:

- ☐ What ContainerLab is and its primary purpose
- ☐ How ContainerLab compares to other simulation tools
- ☐ Key use cases for ContainerLab in networking
- ☐ Basic architecture and components
- ☐ Benefits for CCNA-level learning

Summary

ContainerLab represents a modern approach to network simulation and learning. By leveraging containerization technology, it provides a cost-effective, scalable, and realistic environment for network education and testing. Its multi-vendor support and automation-friendly design make it an ideal platform for both learning fundamental networking concepts and developing modern network engineering skills.

In the next chapter, we'll walk through the installation and setup process to get your ContainerLab environment ready for hands-on learning.

Review Questions

1. What are the main advantages of using containers for network simulation?
2. How does ContainerLab differ from traditional hardware-based labs?
3. What types of network operating systems does ContainerLab support?
4. Describe three key use cases for ContainerLab in enterprise environments.
5. What are the core components of ContainerLab's architecture?

Additional Resources

- ContainerLab Official Documentation
- Nokia ContainerLab GitHub Repository
- Docker Networking Fundamentals
- Network Simulation Best Practices

Chapter 2: Installation and Environment Setup

Learning Objectives

By the end of this chapter, you will be able to: - Identify system requirements for ContainerLab - Install Docker and ContainerLab on different operating systems - Configure the development environment - Troubleshoot common installation issues - Verify the installation and run basic tests

System Requirements

Minimum Requirements

Component	Requirement
Operating System	Linux (Ubuntu 18.04+, CentOS 7+, RHEL 7+), macOS 10.15+, Windows 10/11 with WSL2
CPU	2 cores (4+ recommended)
RAM	4GB (8GB+ recommended)
Storage	20GB free space (50GB+ recommended)
Network	Internet connection for image downloads

Recommended Requirements

Component	Recommendation
CPU	8+ cores for complex topologies
RAM	16GB+ for multiple concurrent labs
Storage	SSD with 100GB+ free space
Network	High-speed internet for faster image pulls

Platform-Specific Considerations

Linux (Recommended Platform)

- Native Docker support
- Best performance
- Full feature compatibility
- Easiest troubleshooting

macOS

- Docker Desktop required
- Good performance with Apple Silicon
- Some limitations with nested virtualization
- Resource sharing considerations

Windows

- WSL2 required for optimal performance
- Docker Desktop with WSL2 backend
- Additional complexity in setup
- Potential performance overhead

Prerequisites Installation

Installing Docker

Linux (Ubuntu/Debian)

```
# Update package index
sudo apt update

# Install required packages
sudo apt install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/k

# Add Docker repository
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyri
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker Engine
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io

# Add user to docker group
sudo usermod -aG docker $USER
```

```
# Start and enable Docker service
sudo systemctl start docker
sudo systemctl enable docker
```

Linux (CentOS/RHEL)

```
# Install required packages
sudo yum install -y yum-utils

# Add Docker repository
sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

# Install Docker Engine
sudo yum install -y docker-ce docker-ce-cli containerd.io

# Start and enable Docker service
sudo systemctl start docker
sudo systemctl enable docker

# Add user to docker group
sudo usermod -aG docker $USER
```

macOS

```
# Install Docker Desktop using Homebrew
brew install --cask docker

# Or download from Docker website
# https://docs.docker.com/desktop/mac/install/

# Start Docker Desktop application
open /Applications/Docker.app
```

Windows (WSL2)

```
# Enable WSL2 feature
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norest
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

# Restart computer and set WSL2 as default
```

```
wsl --set-default-version 2

# Install Ubuntu from Microsoft Store
# Then install Docker Desktop for Windows with WSL2 backend
```

Verify Docker Installation

```
# Check Docker version
docker --version

# Test Docker functionality
docker run hello-world

# Check Docker system info
docker system info
```

ContainerLab Installation

Method 1: Binary Installation (Recommended)

Linux/macOS

```
# Download and install the latest release
bash -c "$(curl -sL https://get.containerlab.dev)"

# Or install specific version
sudo curl -sL https://github.com/srl-labs/containerlab/releases/download/v0.47.0/containerla
```

Manual Installation

```
# Download specific version
wget https://github.com/srl-labs/containerlab/releases/download/v0.47.0/containerlab_0.47.0_

# Extract and install
tar -xzf containerlab_0.47.0_linux_amd64.tar.gz
sudo mv containerlab /usr/local/bin/

# Make executable
sudo chmod +x /usr/local/bin/containerlab
```

Method 2: Package Manager Installation

Using Homebrew (macOS/Linux)

```
# Add ContainerLab tap
brew tap srl-labs/containerlab

# Install ContainerLab
brew install containerlab
```

Using APT (Ubuntu/Debian)

```
# Add repository key
curl -sL https://containerlab.dev/setup | sudo bash

# Install ContainerLab
sudo apt update
sudo apt install containerlab
```

Method 3: Container Installation

```
# Create alias for containerized version
alias containerlab='docker run --rm -it --privileged \
    --network host \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /etc/hosts:/etc/hosts \
    -v $(pwd):$(pwd) \
    -w $(pwd) \
    ghcr.io/srl-labs/containerlab'
```

Environment Configuration

Setting Up Working Directory

```
# Create ContainerLab workspace
mkdir -p ~/containerlab-labs
cd ~/containerlab-labs

# Create directory structure
mkdir -p {topologies,configs,scripts,docs}
```

```
# Set up environment variables
echo 'export CLAB_WORKDIR=~/containerlab-labs' >> ~/.bashrc
echo 'export PATH=$PATH:/usr/local/bin' >> ~/.bashrc
source ~/.bashrc
```

Configure Docker for ContainerLab

Increase Docker Resources

```
# Edit Docker daemon configuration
sudo nano /etc/docker/daemon.json
```

Add the following configuration:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  },
  "default-ulimits": {
    "nofile": {
      "Name": "nofile",
      "Hard": 64000,
      "Soft": 64000
    }
  }
}
```

Restart Docker:

```
sudo systemctl restart docker
```

Network Configuration

Enable IP Forwarding

```
# Temporary enable
sudo sysctl net.ipv4.ip_forward=1
sudo sysctl net.ipv6.conf.all.forwarding=1

# Permanent enable
echo 'net.ipv4.ip_forward=1' | sudo tee -a /etc/sysctl.conf
echo 'net.ipv6.conf.all.forwarding=1' | sudo tee -a /etc/sysctl.conf
```

Configure Bridge Networks

```bash
# Load bridge module
sudo modprobe bridge
echo 'bridge' | sudo tee -a /etc/modules

# Configure bridge settings
echo 'net.bridge.bridge-nf-call-iptables=0' | sudo tee -a /etc/sysctl.conf
echo 'net.bridge.bridge-nf-call-ip6tables=0' | sudo tee -a /etc/sysctl.conf
```

Verification and Testing

Verify ContainerLab Installation

```bash
# Check ContainerLab version
containerlab version

# Display help information
containerlab help

# Check available commands
containerlab --help
```

Expected output:

```
                           _                   _       _
                 _        (_)                 | |     | |
 ____ ___  ____ | |_  ____ _ ____   ____  ____| | ____| |__
|  _ \/ _ \|  _ \|  _)/ _  | |  _ \ / _  )/ ___) |/ _  |  _ \
| |_| | |_| | | | |_( ( | | | | | ( (/ /| |   | ( ( | | |_) )
|  __/ \___/|_| |_|\__)_||_|_|_| |_|\____)_|   |_|\_||_|____/
|_|

    version: 0.47.0
     commit: 8c54dd96
       date: 2023-11-15T10:30:45Z
     source: https://github.com/srl-labs/containerlab
 rel. notes: https://containerlab.dev/rn/0.47/
```

Create First Test Lab

Create a simple test topology:

```
# Create test topology file
cat > test-lab.yml << EOF
name: test-lab
topology:
  nodes:
    alpine1:
      kind: linux
      image: alpine:latest
    alpine2:
      kind: linux
      image: alpine:latest
  links:
    - endpoints: ["alpine1:eth1", "alpine2:eth1"]
EOF
```

Deploy and test the lab:

```
# Deploy the lab
containerlab deploy -t test-lab.yml

# Check lab status
containerlab inspect -t test-lab.yml

# Connect to a container
docker exec -it clab-test-lab-alpine1 sh

# Test connectivity (from within alpine1)
ping alpine2

# Exit container
exit

# Destroy the lab
containerlab destroy -t test-lab.yml
```

Troubleshooting Common Issues

Issue 1: Permission Denied

Problem: `permission denied while trying to connect to the Docker daemon socket`

Solution:

```
# Add user to docker group
sudo usermod -aG docker $USER
```

```
# Log out and log back in, or run:
newgrp docker

# Verify group membership
groups $USER
```

Issue 2: ContainerLab Command Not Found

Problem: `containerlab: command not found`

Solution:

```
# Check if binary exists
ls -la /usr/local/bin/containerlab

# Add to PATH if needed
echo 'export PATH=$PATH:/usr/local/bin' >> ~/.bashrc
source ~/.bashrc

# Or create symlink
sudo ln -s /usr/local/bin/containerlab /usr/bin/containerlab
```

Issue 3: Docker Image Pull Failures

Problem: Cannot pull container images

Solution:

```
# Check Docker daemon status
sudo systemctl status docker

# Test Docker connectivity
docker run hello-world

# Check DNS resolution
nslookup registry-1.docker.io

# Configure Docker proxy if needed
sudo mkdir -p /etc/systemd/system/docker.service.d
sudo nano /etc/systemd/system/docker.service.d/http-proxy.conf
```

Issue 4: Insufficient Resources

Problem: Containers fail to start due to resource constraints

Solution:

```
# Check system resources
free -h
df -h

# Monitor Docker resource usage
docker system df
docker stats

# Clean up unused resources
docker system prune -a
```

Issue 5: Network Connectivity Issues

Problem: Containers cannot communicate

Solution:

```
# Check IP forwarding
sysctl net.ipv4.ip_forward

# Verify bridge configuration
brctl show

# Check iptables rules
sudo iptables -L

# Reset Docker networks if needed
docker network prune
```

Development Environment Setup

Text Editor Configuration

VS Code Extensions

```
# Install useful extensions
code --install-extension ms-vscode.vscode-yaml
code --install-extension redhat.vscode-yaml
code --install-extension ms-python.python
code --install-extension ms-vscode.vscode-json
```

Vim Configuration

```
# Add YAML syntax highlighting
echo 'syntax on' >> ~/.vimrc
echo 'set tabstop=2' >> ~/.vimrc
echo 'set shiftwidth=2' >> ~/.vimrc
echo 'set expandtab' >> ~/.vimrc
```

Shell Aliases and Functions

```
# Add useful aliases
cat >> ~/.bashrc << 'EOF'
# ContainerLab aliases
alias clab='containerlab'
alias clab-deploy='containerlab deploy -t'
alias clab-destroy='containerlab destroy -t'
alias clab-inspect='containerlab inspect -t'

# Docker aliases
alias dps='docker ps'
alias dimg='docker images'
alias dlog='docker logs'

# ContainerLab functions
clab-connect() {
    docker exec -it "clab-$1-$2" bash 2>/dev/null || docker exec -it "clab-$1-$2" sh
}

clab-logs() {
    docker logs "clab-$1-$2"
}
EOF

source ~/.bashrc
```

Performance Optimization

System Tuning

```
# Increase file descriptor limits
echo '* soft nofile 65536' | sudo tee -a /etc/security/limits.conf
echo '* hard nofile 65536' | sudo tee -a /etc/security/limits.conf
```

```
# Optimize kernel parameters
echo 'vm.max_map_count=262144' | sudo tee -a /etc/sysctl.conf
echo 'fs.file-max=2097152' | sudo tee -a /etc/sysctl.conf

# Apply changes
sudo sysctl -p
```

Docker Optimization

```
# Configure Docker storage driver
sudo nano /etc/docker/daemon.json
```

Add storage driver configuration:

```
{
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
```

Installation Verification Checklist

Before proceeding to the next chapter, verify:

- ☐ Docker is installed and running
- ☐ ContainerLab is installed and accessible
- ☐ User has proper permissions
- ☐ Test lab deploys successfully
- ☐ Network connectivity works between containers
- ☐ Development environment is configured
- ☐ System resources are adequate

Summary

Proper installation and environment setup are crucial for a smooth ContainerLab experience. This chapter covered the complete setup process across different operating systems, common troubleshooting scenarios, and performance optimization techniques. With your environment properly configured, you're ready to start creating and managing network topologies.

In the next chapter, we'll explore basic ContainerLab operations and learn how to create your first meaningful network lab.

Review Questions

1. What are the minimum system requirements for running ContainerLab?
2. Why is Docker required for ContainerLab operation?
3. How do you verify that ContainerLab is properly installed?
4. What are common causes of permission denied errors?
5. How can you optimize system performance for ContainerLab?

Hands-on Exercises

Exercise 1: Installation Verification

1. Install Docker and ContainerLab on your system
2. Create and deploy the test lab from this chapter
3. Verify connectivity between containers
4. Document any issues encountered and their solutions

Exercise 2: Environment Customization

1. Set up shell aliases for common ContainerLab commands
2. Configure your preferred text editor for YAML editing
3. Create a workspace directory structure
4. Test the customizations with a simple lab deployment

Exercise 3: Troubleshooting Practice

1. Intentionally break your Docker installation
2. Practice the troubleshooting steps from this chapter
3. Document the resolution process
4. Restore full functionality

Additional Resources

- Docker Installation Guide
- ContainerLab Installation Documentation
- System Requirements and Optimization
- Troubleshooting Guide

Chapter 3: Basic ContainerLab Operations

Learning Objectives

By the end of this chapter, you will be able to: - Understand ContainerLab topology file structure and syntax - Create and deploy basic network topologies - Use essential ContainerLab CLI commands - Manage lab lifecycle effectively - Navigate and interact with deployed containers

Understanding Topology Files

YAML Basics for ContainerLab

ContainerLab uses YAML (YAML Ain't Markup Language) for topology definitions. YAML is human-readable and uses indentation to represent data structure.

Key YAML Concepts

```yaml
# Comments start with hash
name: my-lab                    # String value
version: 1.0                    # Number value
enabled: true                   # Boolean value

# Lists (arrays)
items:
  - item1
  - item2
  - item3

# Dictionaries (objects)
node:
  name: router1
  type: cisco
  image: cisco/iosxe:latest
```

Basic Topology Structure

Every ContainerLab topology file contains these main sections:

```yaml
name: lab-name                    # Lab identifier
prefix: custom                    # Optional prefix for container names
mgmt:                           # Management network configuration
  network: custom-mgmt          # Custom management network
  ipv4-subnet: 172.20.20.0/24   # Management subnet

topology:
  nodes:                        # Network devices definition
    node1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
    node2:
      kind: arista_eos
      image: arista/ceos:latest

  links:                        # Connections between nodes
    - endpoints: ["node1:eth1", "node2:eth1"]
    - endpoints: ["node1:eth2", "node2:eth2"]
```

Node Configuration

Basic Node Properties

```yaml
topology:
  nodes:
    router1:
      kind: cisco_iosxe           # Device type
      image: cisco/iosxe:latest   # Container image
      mgmt-ipv4: 172.20.20.10    # Management IP
      ports:                      # Port mappings
        - "22:22"                # SSH access
        - "80:80"                # HTTP access
      env:                        # Environment variables
        HOSTNAME: router1
      labels:                     # Metadata labels
        role: edge-router
        location: site-a
```

Advanced Node Configuration

```yaml
topology:
  nodes:
    core-switch:
      kind: arista_eos
      image: arista/ceos:4.28.3M
      mgmt-ipv4: 172.20.20.20
      startup-config: configs/core-switch.cfg
      binds:                     # Volume mounts
        - /host/path:/container/path
      sysctls:                   # Kernel parameters
        net.ipv4.ip_forward: 1
      cpu: 2                     # CPU limit
      memory: 4GB               # Memory limit
```

Link Configuration

Point-to-Point Links

```
topology:
  links:
    # Basic link
    - endpoints: ["router1:eth1", "router2:eth1"]

    # Link with custom properties
    - endpoints: ["switch1:eth1", "switch2:eth1"]
      mtu: 9000

    # Link with VLAN configuration
    - endpoints: ["router1:eth2", "switch1:eth2"]
      vars:
        vlan: 100
```

Multi-Access Networks

```
topology:
  nodes:
    router1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
    router2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
    router3:
      kind: cisco_iosxe
      image: cisco/iosxe:latest

  links:
    # All routers connected to same network segment
    - endpoints: ["router1:eth1", "router2:eth1", "router3:eth1"]
```

Essential CLI Commands

Lab Deployment Commands

Deploy a Lab

```
# Deploy lab from topology file
containerlab deploy -t topology.yml

# Deploy with custom name
containerlab deploy -t topology.yml --name custom-lab

# Deploy in background
containerlab deploy -t topology.yml --detach

# Deploy with specific prefix
containerlab deploy -t topology.yml --prefix mylab
```

Destroy a Lab

```
# Destroy lab
containerlab destroy -t topology.yml

# Destroy by name
containerlab destroy --name lab-name

# Force destroy (cleanup even if errors)
containerlab destroy -t topology.yml --cleanup

# Destroy all labs
containerlab destroy --all
```

Lab Inspection Commands

Inspect Lab Status

```
# Inspect specific lab
containerlab inspect -t topology.yml

# Inspect by name
containerlab inspect --name lab-name

# Show all running labs
containerlab inspect --all

# Output in different formats
containerlab inspect -t topology.yml --format table
containerlab inspect -t topology.yml --format json
```

Graph Generation

```
# Generate topology graph
containerlab graph -t topology.yml

# Generate with custom output
containerlab graph -t topology.yml --output topology.png

# Generate interactive graph
containerlab graph -t topology.yml --format html
```

Configuration Management

Save Configurations

```
# Save all node configurations
containerlab save -t topology.yml

# Save specific node configuration
containerlab save -t topology.yml --node router1

# Save to custom directory
containerlab save -t topology.yml --destination ./backups/
```

Load Configurations

```
# Load configurations to all nodes
containerlab config -t topology.yml

# Load configuration to specific node
containerlab config -t topology.yml --node router1 --config router1.cfg
```

Creating Your First Lab

Simple Two-Router Topology

Let's create a basic lab with two Cisco routers:

```
# File: basic-routers.yml
name: basic-routers
prefix: lab

topology:
  nodes:
    r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10

    r2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11

  links:
    - endpoints: ["r1:GigabitEthernet2", "r2:GigabitEthernet2"]
```

Deploy and test:

```
# Deploy the lab
containerlab deploy -t basic-routers.yml

# Check lab status
containerlab inspect -t basic-routers.yml

# Connect to router 1
docker exec -it clab-lab-r1 bash

# From within the router, access CLI
```

```
cli

# Check interface status
show ip interface brief

# Exit router CLI and container
exit
exit
```

Multi-Vendor Topology

Create a lab with different vendor devices:

```
# File: multi-vendor.yml
name: multi-vendor
prefix: mv

topology:
  nodes:
    cisco-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10

    arista-switch:
      kind: arista_eos
      image: arista/ceos:latest
      mgmt-ipv4: 172.20.20.11

    linux-host:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.12

  links:
    - endpoints: ["cisco-router:eth1", "arista-switch:eth1"]
    - endpoints: ["arista-switch:eth2", "linux-host:eth1"]
```

Campus Network Topology

A more complex example representing a small campus:

```
# File: campus-network.yml
name: campus-network
prefix: campus
```

```yaml
mgmt:
  network: campus-mgmt
  ipv4-subnet: 192.168.100.0/24

topology:
  nodes:
    # Core Layer
    core-sw1:
      kind: arista_eos
      image: arista/ceos:latest
      mgmt-ipv4: 192.168.100.10
      labels:
        layer: core

    core-sw2:
      kind: arista_eos
      image: arista/ceos:latest
      mgmt-ipv4: 192.168.100.11
      labels:
        layer: core

    # Distribution Layer
    dist-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 192.168.100.20
      labels:
        layer: distribution

    dist-sw2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 192.168.100.21
      labels:
        layer: distribution

    # Access Layer
    access-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 192.168.100.30
      labels:
        layer: access

    access-sw2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 192.168.100.31
      labels:
        layer: access

    # End Devices
    pc1:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 192.168.100.100

    pc2:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 192.168.100.101

  links:
    # Core interconnection
    - endpoints: ["core-sw1:eth1", "core-sw2:eth1"]
    - endpoints: ["core-sw1:eth2", "core-sw2:eth2"]

    # Core to Distribution
    - endpoints: ["core-sw1:eth3", "dist-sw1:eth1"]
    - endpoints: ["core-sw1:eth4", "dist-sw2:eth1"]
    - endpoints: ["core-sw2:eth3", "dist-sw1:eth2"]
    - endpoints: ["core-sw2:eth4", "dist-sw2:eth2"]

    # Distribution to Access
    - endpoints: ["dist-sw1:eth3", "access-sw1:eth1"]
    - endpoints: ["dist-sw2:eth3", "access-sw2:eth1"]

    # Access to End Devices
    - endpoints: ["access-sw1:eth2", "pc1:eth1"]
    - endpoints: ["access-sw2:eth2", "pc2:eth1"]
```

Lab Lifecycle Management

Deployment Process

1. **Validation Phase**
   - Topology file syntax check
   - Image availability verification
   - Resource requirement assessment
2. **Preparation Phase**

    - Container image pulling
    - Network creation
    - Volume preparation
3. **Deployment Phase**
    - Container creation and startup
    - Network interface attachment
    - Initial configuration application
4. **Verification Phase**
    - Container health checks
    - Network connectivity validation
    - Service availability confirmation

Monitoring Lab Status

```
# Check overall lab health
containerlab inspect -t topology.yml

# Monitor container resources
docker stats $(docker ps --filter "label=containerlab" --format "{{.Names}}")

# Check container logs
docker logs clab-lab-router1

# Monitor network interfaces
docker exec clab-lab-router1 ip addr show
```

Lab Maintenance

Updating Configurations

```
# Apply new configuration to running lab
docker cp new-config.cfg clab-lab-router1:/tmp/
docker exec -it clab-lab-router1 cli
# From router CLI:
copy tftp://management-ip/config running-config
```

Scaling Labs

```
# Add nodes to existing topology
# Edit topology file to add new nodes
containerlab deploy -t updated-topology.yml --reconfigure

# Remove nodes
# Edit topology file to remove nodes
containerlab deploy -t updated-topology.yml --reconfigure
```

Container Interaction

Accessing Containers

Direct Shell Access

```
# Access container shell
docker exec -it clab-lab-router1 bash

# Access with specific user
docker exec -it --user root clab-lab-router1 bash

# Run single command
docker exec clab-lab-router1 show version
```

Network Device CLI Access

```
# Cisco devices
docker exec -it clab-lab-cisco-router cli

# Arista devices
docker exec -it clab-lab-arista-switch Cli

# Nokia devices
docker exec -it clab-lab-nokia-router sr_cli
```

File Transfer

Copy Files to/from Containers

```
# Copy file to container
docker cp local-file.cfg clab-lab-router1:/tmp/

# Copy file from container
docker cp clab-lab-router1:/etc/config.cfg ./backup-config.cfg

# Copy directory
docker cp ./configs/ clab-lab-router1:/tmp/configs/
```

Using Volume Mounts

```
topology:
  nodes:
    router1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      binds:
        - ./configs:/tmp/configs:ro    # Read-only mount
        - ./logs:/var/log:rw          # Read-write mount
```

Network Connectivity Testing

Basic Connectivity Tests

```
# From Linux containers
docker exec clab-lab-pc1 ping 192.168.1.1
docker exec clab-lab-pc1 traceroute 192.168.1.1
docker exec clab-lab-pc1 nslookup google.com

# From network devices (varies by vendor)
docker exec clab-lab-router1 cli -c "ping 192.168.1.1"
docker exec clab-lab-switch1 Cli -c "ping 192.168.1.1"
```

Advanced Network Testing

```
# Install network tools in Linux containers
docker exec clab-lab-pc1 apk add --no-cache iperf3 tcpdump nmap

# Performance testing
docker exec -d clab-lab-pc1 iperf3 -s
docker exec clab-lab-pc2 iperf3 -c pc1-ip

# Packet capture
docker exec -d clab-lab-pc1 tcpdump -i eth1 -w /tmp/capture.pcap
```

Best Practices

Topology Design

1. **Use meaningful names** for nodes and labs
2. **Organize by function** (core, distribution, access)
3. **Include metadata** using labels
4. **Plan IP addressing** systematically
5. **Document topology** with comments

Resource Management

1. **Monitor system resources** during deployment
2. **Use appropriate image versions** for your needs
3. **Clean up unused labs** regularly
4. **Optimize container resource limits**

Configuration Management

1. **Use startup configurations** for consistent deployments
2. **Version control** topology files
3. **Backup configurations** regularly
4. **Test changes** in isolated environments

Troubleshooting Common Issues

Deployment Failures

Image Pull Issues

```
# Check image availability
docker pull cisco/iosxe:latest

# Use alternative registry
docker pull registry.example.com/cisco/iosxe:latest

# Check Docker Hub rate limits
docker system events --filter type=image
```

Resource Constraints

```
# Check system resources
free -h
df -h

# Monitor during deployment
watch -n 1 'docker stats --no-stream'

# Adjust container limits
topology:
  nodes:
    router1:
      cpu: 1
      memory: 2GB
```

Connectivity Issues

Container Network Problems

```
# Check container networks
docker network ls
docker network inspect clab-network

# Verify interface configuration
docker exec clab-lab-router1 ip addr show

# Check routing
```

```
docker exec clab-lab-router1 ip route show
```

Inter-Container Communication

```
# Test basic connectivity
docker exec clab-lab-pc1 ping clab-lab-pc2

# Check bridge configuration
brctl show

# Verify iptables rules
sudo iptables -L -n
```

Summary

This chapter covered the fundamental operations needed to work with ContainerLab effectively. You learned how to structure topology files, use essential CLI commands, and manage lab lifecycles. These skills form the foundation for all subsequent networking labs and experiments.

Key takeaways: - YAML topology files define your entire lab infrastructure - ContainerLab CLI provides comprehensive lab management capabilities - Proper planning and organization improve lab maintainability - Understanding container interaction is crucial for effective troubleshooting

In the next chapter, we'll dive deeper into network topologies and design patterns for more complex scenarios.

Review Questions

1. What are the main sections of a ContainerLab topology file?
2. How do you deploy and destroy a lab using ContainerLab CLI?
3. What's the difference between `kind` and `image` in node configuration?
4. How can you access the CLI of different network operating systems?
5. What are best practices for lab resource management?

Hands-on Exercises

Exercise 1: Basic Lab Creation

1. Create a simple two-router topology
2. Deploy the lab and verify connectivity
3. Access both routers and check interface status
4. Save the router configurations
5. Destroy the lab

Exercise 2: Multi-Vendor Lab

1. Create a topology with Cisco, Arista, and Linux nodes
2. Configure basic IP addressing
3. Test connectivity between all nodes
4. Generate a topology graph
5. Document the lab setup

Exercise 3: Campus Network Simulation

1. Implement the campus network topology from this chapter
2. Deploy and verify all nodes are running
3. Plan and document IP addressing scheme
4. Test management connectivity to all devices
5. Practice lab lifecycle management operations

Additional Resources

- ContainerLab Topology Definition
- Supported Network Operating Systems
- CLI Reference Guide
- YAML Syntax Guide

Chapter 4: Network Topologies and Design

Learning Objectives

By the end of this chapter, you will be able to: - Design and implement various network topologies in ContainerLab - Understand topology file structure and advanced syntax - Create scalable and maintainable network designs - Implement hierarchical network models - Apply network design best practices

Network Topology Fundamentals

Physical vs. Logical Topologies

In ContainerLab, we work primarily with logical topologies that represent how network devices are interconnected. Understanding both physical and logical aspects helps in creating realistic simulations.

Physical Topology Considerations

- **Cable types and limitations**: Simulated through link properties
- **Distance constraints**: Represented by latency and bandwidth settings
- **Hardware limitations**: Modeled through container resource constraints
- **Redundancy requirements**: Implemented through multiple links

Logical Topology Elements

- **Network segments**: Created through ContainerLab links
- **Broadcast domains**: Defined by switch configurations
- **Routing domains**: Established through routing protocol configurations
- **Security zones**: Implemented through firewall and ACL configurations

Common Network Topologies

Star Topology

```yaml
# Star topology with central switch
name: star-topology
topology:
  nodes:
    central-switch:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.10

    pc1:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.11

    pc2:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.12

    pc3:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.13

    pc4:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.14

  links:
    - endpoints: ["central-switch:eth1", "pc1:eth1"]
    - endpoints: ["central-switch:eth2", "pc2:eth1"]
    - endpoints: ["central-switch:eth3", "pc3:eth1"]
    - endpoints: ["central-switch:eth4", "pc4:eth1"]
```

Advantages: - Simple to implement and troubleshoot - Centralized management - Easy to add new devices

Disadvantages: - Single point of failure - Limited scalability - Bandwidth bottleneck at center

Ring Topology

```
# Ring topology for redundancy
name: ring-topology
topology:
  nodes:
    switch1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.10

    switch2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.11

    switch3:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.12

    switch4:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.13

  links:
    # Ring connections
    - endpoints: ["switch1:eth1", "switch2:eth1"]
    - endpoints: ["switch2:eth2", "switch3:eth1"]
    - endpoints: ["switch3:eth2", "switch4:eth1"]
    - endpoints: ["switch4:eth2", "switch1:eth2"]  # Completes the ring
```

Advantages: - Built-in redundancy - No single point of failure - Predictable performance

Disadvantages: - Complex troubleshooting - Potential for loops without proper protocols - Limited bandwidth sharing

Mesh Topology

```
# Full mesh topology for maximum redundancy
name: mesh-topology
topology:
  nodes:
    router1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10

    router2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11

    router3:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.12

    router4:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.13

  links:
    # Full mesh - every router connected to every other router
    - endpoints: ["router1:eth1", "router2:eth1"]
    - endpoints: ["router1:eth2", "router3:eth1"]
    - endpoints: ["router1:eth3", "router4:eth1"]
    - endpoints: ["router2:eth2", "router3:eth2"]
    - endpoints: ["router2:eth3", "router4:eth2"]
    - endpoints: ["router3:eth3", "router4:eth3"]
```

Advantages: - Maximum redundancy - Optimal path selection - High fault tolerance

Disadvantages: - Expensive to implement - Complex configuration - Scalability challenges

Hierarchical Network Design

Three-Tier Architecture

The three-tier hierarchical model is the foundation of most enterprise networks:

Core Layer

- High-speed packet switching
- Redundancy and fault tolerance
- Minimal processing overhead

Distribution Layer

- Policy enforcement
- Routing between VLANs
- Access control and security

Access Layer

- End-device connectivity
- Port security
- VLAN assignment

Implementing Three-Tier Design

```yaml
# Three-tier campus network
name: three-tier-campus
prefix: campus

mgmt:
  network: campus-mgmt
  ipv4-subnet: 192.168.100.0/24

topology:
  nodes:
    # Core Layer - High-performance switches
    core-sw1:
      kind: arista_eos
      image: arista/ceos:latest
      mgmt-ipv4: 192.168.100.10
      labels:
        tier: core
        role: core-switch
      cpu: 2
      memory: 4GB

    core-sw2:
      kind: arista_eos
      image: arista/ceos:latest
      mgmt-ipv4: 192.168.100.11
      labels:
        tier: core
        role: core-switch
      cpu: 2
      memory: 4GB
    # Distribution Layer - Policy and routing
    dist-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 192.168.100.20
      labels:
        tier: distribution
        role: distribution-switch
        building: building-a
      cpu: 1
      memory: 2GB

    dist-sw2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 192.168.100.21
      labels:
        tier: distribution
        role: distribution-switch
        building: building-a
      cpu: 1
      memory: 2GB

    dist-sw3:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 192.168.100.22
      labels:
        tier: distribution
        role: distribution-switch
        building: building-b
      cpu: 1
      memory: 2GB

    dist-sw4:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 192.168.100.23
      labels:
        tier: distribution
        role: distribution-switch
        building: building-b
      cpu: 1
      memory: 2GB

    # Access Layer - End device connectivity
    access-sw1:
```

```
    kind: cisco_iosxe
    image: cisco/catalyst:latest
    mgmt-ipv4: 192.168.100.30
    labels:
      tier: access
      role: access-switch
      building: building-a
      floor: floor-1

  access-sw2:
    kind: cisco_iosxe
    image: cisco/catalyst:latest
    mgmt-ipv4: 192.168.100.31
    labels:
      tier: access
      role: access-switch
      building: building-a
      floor: floor-2

  access-sw3:
    kind: cisco_iosxe
    image: cisco/catalyst:latest
    mgmt-ipv4: 192.168.100.32
    labels:
      tier: access
      role: access-switch
      building: building-b
      floor: floor-1

  access-sw4:
    kind: cisco_iosxe
    image: cisco/catalyst:latest
    mgmt-ipv4: 192.168.100.33
    labels:
      tier: access
      role: access-switch
      building: building-b
      floor: floor-2

  # End Devices
  pc1:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 192.168.100.100
    labels:
      role: end-device
      location: building-a-floor-1
  pc2:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 192.168.100.101
    labels:
      role: end-device
      location: building-a-floor-2

  pc3:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 192.168.100.102
    labels:
      role: end-device
      location: building-b-floor-1

  pc4:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 192.168.100.103
    labels:
      role: end-device
      location: building-b-floor-2

  # Servers
  server1:
    kind: linux
    image: ubuntu:20.04
    mgmt-ipv4: 192.168.100.200
    labels:
      role: server
      service: web-server

  server2:
    kind: linux
    image: ubuntu:20.04
    mgmt-ipv4: 192.168.100.201
    labels:
      role: server
      service: database-server

links:
  # Core Layer Interconnection (redundant)
  - endpoints: ["core-sw1:eth1", "core-sw2:eth1"]
  - endpoints: ["core-sw1:eth2", "core-sw2:eth2"]
```

```
    # Core to Distribution (dual-homed)
    - endpoints: ["core-sw1:eth3", "dist-sw1:eth1"]
    - endpoints: ["core-sw1:eth4", "dist-sw2:eth1"]
    - endpoints: ["core-sw1:eth5", "dist-sw3:eth1"]
    - endpoints: ["core-sw1:eth6", "dist-sw4:eth1"]

    - endpoints: ["core-sw2:eth3", "dist-sw1:eth2"]
    - endpoints: ["core-sw2:eth4", "dist-sw2:eth2"]
    - endpoints: ["core-sw2:eth5", "dist-sw3:eth2"]
    - endpoints: ["core-sw2:eth6", "dist-sw4:eth2"]

    # Distribution Layer Interconnection (within building)
    - endpoints: ["dist-sw1:eth3", "dist-sw2:eth3"]
    - endpoints: ["dist-sw3:eth3", "dist-sw4:eth3"]

    # Distribution to Access
    - endpoints: ["dist-sw1:eth4", "access-sw1:eth1"]
    - endpoints: ["dist-sw1:eth5", "access-sw2:eth1"]
    - endpoints: ["dist-sw2:eth4", "access-sw1:eth2"]
    - endpoints: ["dist-sw2:eth5", "access-sw2:eth2"]

    - endpoints: ["dist-sw3:eth4", "access-sw3:eth1"]
    - endpoints: ["dist-sw3:eth5", "access-sw4:eth1"]
    - endpoints: ["dist-sw4:eth4", "access-sw3:eth2"]
    - endpoints: ["dist-sw4:eth5", "access-sw4:eth2"]

    # Access to End Devices
    - endpoints: ["access-sw1:eth3", "pc1:eth1"]
    - endpoints: ["access-sw2:eth3", "pc2:eth1"]
    - endpoints: ["access-sw3:eth3", "pc3:eth1"]
    - endpoints: ["access-sw4:eth3", "pc4:eth1"]

    # Servers connected to core (high bandwidth)
    - endpoints: ["core-sw1:eth7", "server1:eth1"]
    - endpoints: ["core-sw2:eth7", "server2:eth1"]
```

Two-Tier (Collapsed Core) Design

For smaller networks, the core and distribution layers can be combined:

```
# Two-tier collapsed core design
name: two-tier-network
topology:
  nodes:
    # Collapsed Core/Distribution
    core-dist-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.10
      labels:
        tier: core-distribution

    core-dist-sw2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.11
      labels:
        tier: core-distribution

    # Access Layer
    access-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.20

    access-sw2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.21

  links:
    # Core-Distribution interconnection
    - endpoints: ["core-dist-sw1:eth1", "core-dist-sw2:eth1"]
    - endpoints: ["core-dist-sw1:eth2", "core-dist-sw2:eth2"]

    # Dual-homed access switches
    - endpoints: ["core-dist-sw1:eth3", "access-sw1:eth1"]
    - endpoints: ["core-dist-sw2:eth3", "access-sw1:eth2"]
    - endpoints: ["core-dist-sw1:eth4", "access-sw2:eth1"]
    - endpoints: ["core-dist-sw2:eth4", "access-sw2:eth2"]
```

Advanced Topology Features

Custom Link Properties

ContainerLab allows you to define custom properties for links to simulate real-world conditions:

```
topology:
  links:
    # High-speed core link
    - endpoints: ["core-sw1:eth1", "core-sw2:eth1"]
```

```
      vars:
        bandwidth: 10Gbps
        latency: 1ms
        mtu: 9000

    # WAN link with constraints
    - endpoints: ["router1:eth1", "router2:eth1"]
      vars:
        bandwidth: 100Mbps
        latency: 50ms
        packet_loss: 0.1%
        jitter: 5ms

    # Backup link
    - endpoints: ["router1:eth2", "router2:eth2"]
      vars:
        bandwidth: 10Mbps
        latency: 100ms
        cost: 200
        backup: true
```

Network Segmentation

VLAN-based Segmentation

```
topology:
  nodes:
    switch1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      startup-config: |
        vlan 10
         name USERS
        vlan 20
         name SERVERS
        vlan 30
         name MANAGEMENT
        !
        interface GigabitEthernet1/0/1
         switchport mode access
         switchport access vlan 10
        !
        interface GigabitEthernet1/0/2
         switchport mode access
         switchport access vlan 20
```

```
          !
          interface GigabitEthernet1/0/3
           switchport mode trunk
           switchport trunk allowed vlan 10,20,30

  links:
    - endpoints: ["switch1:eth1", "user-pc:eth1"]
      vars:
        vlan: 10
    - endpoints: ["switch1:eth2", "server:eth1"]
      vars:
        vlan: 20
    - endpoints: ["switch1:eth3", "router:eth1"]
      vars:
        trunk: [10, 20, 30]
```

Subnet-based Segmentation

```
topology:
  nodes:
    router1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        interface GigabitEthernet0/0/1
         ip address 192.168.10.1 255.255.255.0
         description USER_NETWORK
        !
        interface GigabitEthernet0/0/2
         ip address 192.168.20.1 255.255.255.0
         description SERVER_NETWORK
        !
        interface GigabitEthernet0/0/3
         ip address 192.168.30.1 255.255.255.0
         description MANAGEMENT_NETWORK
```

Multi-Site Topologies

Hub-and-Spoke WAN Design

```
# Hub-and-spoke WAN topology
name: hub-spoke-wan
topology:
  nodes:
```

```yaml
# Hub site
hub-router:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.10
  labels:
    site: headquarters
    role: hub-router

hub-switch:
  kind: cisco_iosxe
  image: cisco/catalyst:latest
  mgmt-ipv4: 172.20.20.11
  labels:
    site: headquarters
    role: lan-switch

# Spoke sites
spoke1-router:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.20
  labels:
    site: branch-office-1
    role: spoke-router

spoke1-switch:
  kind: cisco_iosxe
  image: cisco/catalyst:latest
  mgmt-ipv4: 172.20.20.21
  labels:
    site: branch-office-1
    role: lan-switch

spoke2-router:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.30
  labels:
    site: branch-office-2
    role: spoke-router

spoke2-switch:
  kind: cisco_iosxe
  image: cisco/catalyst:latest
  mgmt-ipv4: 172.20.20.31
  labels:
        site: branch-office-2
        role: lan-switch

    # Internet/WAN simulation
    wan-cloud:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.100
      labels:
        role: wan-simulation

  links:
    # Hub LAN
    - endpoints: ["hub-router:eth1", "hub-switch:eth1"]

    # Spoke LANs
    - endpoints: ["spoke1-router:eth1", "spoke1-switch:eth1"]
    - endpoints: ["spoke2-router:eth1", "spoke2-switch:eth1"]

    # WAN connections (hub-and-spoke)
    - endpoints: ["hub-router:eth2", "wan-cloud:eth1"]
      vars:
        bandwidth: 1Gbps
        latency: 10ms

    - endpoints: ["spoke1-router:eth2", "wan-cloud:eth2"]
      vars:
        bandwidth: 100Mbps
        latency: 30ms

    - endpoints: ["spoke2-router:eth2", "wan-cloud:eth3"]
      vars:
        bandwidth: 100Mbps
        latency: 40ms
```

Full Mesh WAN Design

```yaml
# Full mesh WAN for critical sites
name: mesh-wan
topology:
  nodes:
    site1-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      labels:
        site: site-1

    site2-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      labels:
        site: site-2

    site3-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      labels:
        site: site-3

    site4-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      labels:
        site: site-4

  links:
    # Full mesh WAN connections
    - endpoints: ["site1-router:eth1", "site2-router:eth1"]
      vars: {bandwidth: "1Gbps", latency: "20ms"}
    - endpoints: ["site1-router:eth2", "site3-router:eth1"]
      vars: {bandwidth: "1Gbps", latency: "25ms"}
    - endpoints: ["site1-router:eth3", "site4-router:eth1"]
      vars: {bandwidth: "1Gbps", latency: "30ms"}
    - endpoints: ["site2-router:eth2", "site3-router:eth2"]
      vars: {bandwidth: "1Gbps", latency: "15ms"}
    - endpoints: ["site2-router:eth3", "site4-router:eth2"]
      vars: {bandwidth: "1Gbps", latency: "35ms"}
    - endpoints: ["site3-router:eth3", "site4-router:eth3"]
      vars: {bandwidth: "1Gbps", latency: "20ms"}
```

Design Best Practices

Scalability Considerations

1. **Modular Design**
   - Use consistent naming conventions
   - Group related components
   - Plan for future expansion
2. **Resource Planning**

- Estimate container resource requirements
- Plan for peak usage scenarios
- Consider host system limitations

3. **Network Addressing**

   - Use hierarchical IP addressing
   - Reserve address space for growth
   - Document addressing schemes

Redundancy and High Availability

1. **Link Redundancy**

   - Implement dual-homed connections
   - Use different physical paths
   - Configure appropriate spanning tree

2. **Device Redundancy**

   - Deploy redundant core devices
   - Use HSRP/VRRP for gateway redundancy
   - Implement load balancing

3. **Service Redundancy**

   - Distribute critical services
   - Implement clustering where appropriate
   - Plan for disaster recovery

Performance Optimization

1. **Bandwidth Planning**

   - Size links appropriately
   - Consider oversubscription ratios
   - Plan for traffic growth

2. **Latency Minimization**

   - Optimize routing paths
   - Minimize hop counts
   - Use appropriate QoS policies

3. **Resource Allocation**

   - Right-size container resources
   - Monitor resource utilization
   - Implement resource limits

Topology Validation and Testing

Pre-deployment Validation

```
# Validate topology syntax
containerlab validate -t topology.yml

# Check resource requirements
containerlab inspect -t topology.yml --dry-run

# Verify image availability
containerlab images -t topology.yml
```

Post-deployment Testing

```
# Deploy and test connectivity
containerlab deploy -t topology.yml

# Generate topology graph
containerlab graph -t topology.yml

# Test basic connectivity
for node in $(containerlab inspect -t topology.yml --format json | jq -r '.containers[].name
    echo "Testing $node..."
    docker exec $node ping -c 1 8.8.8.8
done

# Performance testing
docker exec clab-lab-pc1 iperf3 -c clab-lab-pc2
```

Automated Testing

```
#!/bin/bash
# automated-topology-test.sh

TOPOLOGY_FILE="$1"
TEST_RESULTS="test-results-$(date +%Y%m%d-%H%M%S).log"

echo "Starting topology test for $TOPOLOGY_FILE" | tee $TEST_RESULTS

# Deploy topology
echo "Deploying topology..." | tee -a $TEST_RESULTS
if containerlab deploy -t $TOPOLOGY_FILE; then
```

```
    echo "Deployment successful" | tee -a $TEST_RESULTS
else
    echo "Deployment failed" | tee -a $TEST_RESULTS
    exit 1
fi

# Wait for containers to stabilize
sleep 30

# Test connectivity
echo "Testing connectivity..." | tee -a $TEST_RESULTS
CONTAINERS=$(containerlab inspect -t $TOPOLOGY_FILE --format json | jq -r '.containers[].nam

for container in $CONTAINERS; do
    echo "Testing $container..." | tee -a $TEST_RESULTS
    if docker exec $container ping -c 3 -W 5 8.8.8.8 > /dev/null 2>&1; then
        echo "$container: Connectivity OK" | tee -a $TEST_RESULTS
    else
        echo "$container: Connectivity FAILED" | tee -a $TEST_RESULTS
    fi
done

# Cleanup
echo "Cleaning up..." | tee -a $TEST_RESULTS
containerlab destroy -t $TOPOLOGY_FILE

echo "Test completed. Results in $TEST_RESULTS"
```

Summary

Network topology design is fundamental to creating effective learning environments and realistic simulations. This chapter covered various topology types, hierarchical design principles, and advanced features available in ContainerLab. Understanding these concepts enables you to create scalable, maintainable, and realistic network simulations.

Key takeaways: - Choose appropriate topology types based on requirements - Implement hierarchical designs for scalability - Use advanced features for realistic simulations - Follow best practices for maintainable designs - Validate and test topologies thoroughly

In the next chapter, we'll explore the various network operating systems supported by ContainerLab and their specific configurations.

Review Questions

1. What are the advantages and disadvantages of different topology types?

2. How does the three-tier hierarchical model improve network design?
3. What are the key considerations for multi-site topology design?
4. How can you implement redundancy in ContainerLab topologies?
5. What are best practices for topology validation and testing?

Hands-on Exercises

Exercise 1: Topology Comparison

1. Implement star, ring, and mesh topologies with 4 nodes each
2. Compare deployment time and resource usage
3. Test connectivity and performance characteristics
4. Document advantages and disadvantages of each

Exercise 2: Three-Tier Design

1. Implement the three-tier campus network from this chapter
2. Configure appropriate VLANs and IP addressing
3. Test connectivity between all tiers
4. Implement and test redundancy features

Exercise 3: Multi-Site WAN

1. Create a hub-and-spoke WAN topology with 3 spoke sites
2. Configure routing between sites
3. Simulate WAN link failures and test failover
4. Compare with a full mesh design

Exercise 4: Custom Topology Design

1. Design a topology for a specific scenario (e.g., small business, data center)
2. Implement the design in ContainerLab
3. Create automated testing scripts
4. Document the design decisions and trade-offs

Additional Resources

- Network Design Principles
- Hierarchical Network Design
- ContainerLab Topology Examples
- Network Topology Best Practices

Chapter 7: Ethernet and Switching Fundamentals

Learning Objectives

By the end of this chapter, you will be able to: - Understand Ethernet frame structure and operation - Configure basic switching functionality in ContainerLab - Implement MAC address learning and forwarding - Understand collision and broadcast domains - Troubleshoot common switching issues

Ethernet Fundamentals

Ethernet Frame Structure

Ethernet is the most common Layer 2 protocol used in modern networks. Understanding the frame structure is crucial for network troubleshooting and optimization.

Ethernet II Frame Format

```
|  Preamble   | SFD | Destination MAC | Source MAC | EtherType | Payload | FCS |
|  7 bytes    | 1B  |    6 bytes      |  6 bytes   |  2 bytes  | 46-1500B| 4B  |
```

Frame Components: - **Preamble**: 7 bytes of alternating 1s and 0s for synchronization - **Start Frame Delimiter (SFD)**: 1 byte marking frame start - **Destination MAC**: 6-byte hardware address of receiving device - **Source MAC**: 6-byte hardware address of sending device - **EtherType**: 2-byte field indicating upper layer protocol - **Payload**: 46-1500 bytes of actual data - **Frame Check Sequence (FCS)**: 4-byte error detection field

MAC Address Structure

MAC (Media Access Control) addresses are 48-bit identifiers assigned to network interfaces.

MAC Address Format

```
Format: XX:XX:XX:XX:XX:XX (hexadecimal)
Example: 00:1B:44:11:3A:B7

Structure:
- First 24 bits: Organizationally Unique Identifier (OUI)
- Last 24 bits: Device-specific identifier

Special Addresses:
- Broadcast: FF:FF:FF:FF:FF:FF
- Multicast: First bit of first octet = 1
- Unicast: First bit of first octet = 0
```

Switch Operation Fundamentals

MAC Address Learning

Switches learn MAC addresses by examining the source MAC address of incoming frames and associating them with the ingress port.

Learning Process

1. **Frame Reception**: Switch receives frame on a port
2. **Source Learning**: Records source MAC and ingress port
3. **Destination Lookup**: Checks MAC table for destination
4. **Forwarding Decision**: Forwards, floods, or filters frame

Creating a Basic Switch Lab

```yaml
# Basic switching lab
name: basic-switching
prefix: sw

topology:
  nodes:
    switch1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Switch1
        !
        interface GigabitEthernet1/0/1
```

```
          description PC1
          switchport mode access
          switchport access vlan 1
          no shutdown
         !
         interface GigabitEthernet1/0/2
          description PC2
          switchport mode access
          switchport access vlan 1
          no shutdown
         !
         interface GigabitEthernet1/0/3
          description PC3
          switchport mode access
          switchport access vlan 1
          no shutdown
         !

    pc1:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.11
      exec:
        - ip addr add 192.168.1.10/24 dev eth1
        - ip link set eth1 up

    pc2:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.12
      exec:
        - ip addr add 192.168.1.11/24 dev eth1
        - ip link set eth1 up

    pc3:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.13
      exec:
        - ip addr add 192.168.1.12/24 dev eth1
        - ip link set eth1 up

  links:
    - endpoints: ["switch1:eth1", "pc1:eth1"]
    - endpoints: ["switch1:eth2", "pc2:eth1"]
    - endpoints: ["switch1:eth3", "pc3:eth1"]
```

Deploy and test the lab:

```
# Deploy the lab
containerlab deploy -t basic-switching.yml

# Connect to the switch
docker exec -it clab-sw-switch1 cli

# Check MAC address table (initially empty)
show mac address-table

# Generate traffic from PC1 to PC2
docker exec clab-sw-pc1 ping -c 3 192.168.1.11

# Check MAC address table again (should show learned addresses)
docker exec -it clab-sw-switch1 cli -c "show mac address-table"
```

MAC Address Table Management

Viewing MAC Address Table

```
# Cisco IOS commands
show mac address-table
show mac address-table dynamic
show mac address-table interface GigabitEthernet1/0/1
show mac address-table vlan 1

# Arista EOS commands
show mac address-table
show mac address-table dynamic
show mac address-table interface Ethernet1
```

MAC Address Table Aging

```
# Configure MAC address aging
startup-config: |
  mac address-table aging-time 300
  !
  interface range GigabitEthernet1/0/1-24
   switchport mode access
   switchport access vlan 1
  !
```

Frame Forwarding Process

Unicast Frame Forwarding

1. **Known Unicast**: Destination MAC in table → Forward to specific port
2. **Unknown Unicast**: Destination MAC not in table → Flood to all ports except ingress

Broadcast Frame Handling

Broadcast frames (destination FF:FF:FF:FF:FF:FF) are flooded to all ports in the same VLAN except the ingress port.

Multicast Frame Handling

Multicast frames are typically flooded like broadcasts unless IGMP snooping is configured.

Collision and Broadcast Domains

Collision Domains

A collision domain is a network segment where data collisions can occur. In modern switched networks, each switch port represents a separate collision domain.

Hub vs. Switch Collision Domains

```yaml
# Simulating hub behavior (single collision domain)
name: collision-domain-demo
topology:
  nodes:
    # Simulate hub with bridge
    hub-sim:
      kind: linux
      image: alpine:latest
      exec:
        - apk add --no-cache bridge-utils
        - brctl addbr br0
        - brctl stp br0 off
        - ip link set br0 up

    pc1:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.1.10/24 dev eth1
    pc2:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.1.11/24 dev eth1

    pc3:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.1.12/24 dev eth1

  links:
    - endpoints: ["hub-sim:eth1", "pc1:eth1"]
    - endpoints: ["hub-sim:eth2", "pc2:eth1"]
    - endpoints: ["hub-sim:eth3", "pc3:eth1"]
```

Broadcast Domains

A broadcast domain is a network segment where broadcast frames are propagated. VLANs are used to separate broadcast domains.

Single Broadcast Domain

```
# Single broadcast domain example
name: single-broadcast-domain
topology:
  nodes:
    switch1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      startup-config: |
        hostname Switch1
        !
        vlan 10
         name USERS
        !
        interface range GigabitEthernet1/0/1-4
         switchport mode access
         switchport access vlan 10
        !

    pc1:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.10/24 dev eth1

    pc2:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.11/24 dev eth1

    pc3:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.12/24 dev eth1

    pc4:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.13/24 dev eth1

  links:
    - endpoints: ["switch1:eth1", "pc1:eth1"]
    - endpoints: ["switch1:eth2", "pc2:eth1"]
    - endpoints: ["switch1:eth3", "pc3:eth1"]
    - endpoints: ["switch1:eth4", "pc4:eth1"]
```

Test broadcast behavior:

```
# Generate broadcast traffic
docker exec clab-single-broadcast-domain-pc1 ping -b 192.168.10.255

# Monitor traffic on other PCs
docker exec clab-single-broadcast-domain-pc2 tcpdump -i eth1 icmp
```

Advanced Switching Features

Port Security

Port security limits the number of MAC addresses that can be learned on a switch port.

```
# Port security configuration
startup-config: |
  interface GigabitEthernet1/0/1
```

```
     switchport mode access
     switchport access vlan 10
     switchport port-security
     switchport port-security maximum 1
     switchport port-security violation shutdown
     switchport port-security mac-address sticky
    !
```

Port Security Violation Actions

- **Shutdown**: Disables the port (default)
- **Restrict**: Drops violating frames, sends SNMP trap
- **Protect**: Drops violating frames silently

Storm Control

Storm control prevents broadcast, multicast, or unicast storms from overwhelming the network.

```
# Storm control configuration
startup-config: |
  interface GigabitEthernet1/0/1
   storm-control broadcast level 50.00
   storm-control multicast level 50.00
   storm-control action shutdown
  !
```

Link Aggregation (EtherChannel)

Link aggregation combines multiple physical links into a single logical link for increased bandwidth and redundancy.

```
# EtherChannel configuration
name: etherchannel-lab
topology:
  nodes:
    switch1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      startup-config: |
        interface range GigabitEthernet1/0/1-2
         channel-group 1 mode active
         no shutdown
        !
        interface Port-channel1
```

```
          switchport mode trunk
          switchport trunk allowed vlan all
         !

    switch2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      startup-config: |
        interface range GigabitEthernet1/0/1-2
         channel-group 1 mode active
         no shutdown
        !
        interface Port-channel1
         switchport mode trunk
         switchport trunk allowed vlan all
        !

  links:
    - endpoints: ["switch1:eth1", "switch2:eth1"]
    - endpoints: ["switch1:eth2", "switch2:eth2"]
```

EtherChannel Protocols

- **LACP (Link Aggregation Control Protocol)**: IEEE 802.3ad standard
- **PAgP (Port Aggregation Protocol)**: Cisco proprietary

```
# Verify EtherChannel status
show etherchannel summary
show etherchannel port-channel
show lacp neighbor
```

Switch Configuration Examples

Basic Switch Configuration

```
# Complete basic switch setup
startup-config: |
  hostname Access-Switch-01
  !
  enable secret cisco123
  !
  username admin privilege 15 secret admin123
  !
  ip domain-name lab.local
  crypto key generate rsa modulus 2048
  !
  line vty 0 15
   login local
   transport input ssh
  !
  interface vlan 1
   ip address 192.168.1.10 255.255.255.0
   no shutdown
  !
  ip default-gateway 192.168.1.1
  !
  interface range GigabitEthernet1/0/1-24
   switchport mode access
   switchport access vlan 1
   spanning-tree portfast
   no shutdown
  !
  interface range GigabitEthernet1/0/25-26
   switchport mode trunk
   switchport trunk allowed vlan all
   no shutdown
  !
```

Multi-Switch Topology

```
# Multi-switch campus network
name: campus-switching
topology:
  nodes:
    core-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Core-Switch-1
        !
        vlan 10
         name USERS
        vlan 20
         name SERVERS
        vlan 30
         name MANAGEMENT
        !
        interface GigabitEthernet1/0/1
         switchport mode trunk
         switchport trunk allowed vlan 10,20,30
        !
        interface GigabitEthernet1/0/2
         switchport mode trunk
         switchport trunk allowed vlan 10,20,30
        !

    access-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Access-Switch-1
        !
        vlan 10
         name USERS
        !
        interface GigabitEthernet1/0/1
         switchport mode trunk
         switchport trunk allowed vlan 10
        !
        interface range GigabitEthernet1/0/2-5
         switchport mode access
         switchport access vlan 10
         spanning-tree portfast
        !

    access-sw2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.12
      startup-config: |
        hostname Access-Switch-2
        !
        vlan 20
         name SERVERS
        !
        interface GigabitEthernet1/0/1
         switchport mode trunk
         switchport trunk allowed vlan 20
        !
        interface range GigabitEthernet1/0/2-5
         switchport mode access
         switchport access vlan 20
        !
```

```yaml
    # End devices
    pc1:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.10/24 dev eth1

    pc2:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.11/24 dev eth1

    server1:
      kind: linux
      image: ubuntu:20.04
      exec:
        - ip addr add 192.168.20.10/24 dev eth1

    server2:
      kind: linux
      image: ubuntu:20.04
      exec:
        - ip addr add 192.168.20.11/24 dev eth1

  links:
    # Core to access switches
    - endpoints: ["core-sw1:eth1", "access-sw1:eth1"]
    - endpoints: ["core-sw1:eth2", "access-sw2:eth1"]

    # End devices to access switches
    - endpoints: ["access-sw1:eth2", "pc1:eth1"]
    - endpoints: ["access-sw1:eth3", "pc2:eth1"]
    - endpoints: ["access-sw2:eth2", "server1:eth1"]
    - endpoints: ["access-sw2:eth3", "server2:eth1"]
```

Troubleshooting Switching Issues

Common Switching Problems

Duplicate MAC Addresses

```
# Symptoms
show mac address-table | include <mac-address>

# Causes
# - Virtualization without proper MAC management
# - Cloned network cards
# - Software bugs

# Resolution
clear mac address-table dynamic address <mac-address>
```

MAC Address Table Overflow

```
# Check MAC table utilization
show mac address-table count

# Configure MAC table size (if supported)
mac address-table limit vlan 10 maximum 1000

# Implement port security
interface GigabitEthernet1/0/1
 switchport port-security
 switchport port-security maximum 10
```

Port Security Violations

```
# Check port security status
show port-security
show port-security interface GigabitEthernet1/0/1

# Clear security violations
clear port-security sticky interface GigabitEthernet1/0/1
```

Diagnostic Commands

Essential Show Commands

```
# Interface status
show interfaces status
show interfaces GigabitEthernet1/0/1

# MAC address table
show mac address-table
show mac address-table dynamic
show mac address-table interface GigabitEthernet1/0/1

# Port security
show port-security
show port-security interface GigabitEthernet1/0/1

# EtherChannel
show etherchannel summary
show etherchannel port-channel

# Storm control
show storm-control
```

Traffic Analysis

```
# Monitor interface counters
show interfaces GigabitEthernet1/0/1 counters

# Clear counters for baseline
clear counters GigabitEthernet1/0/1

# Monitor in real-time
show interfaces GigabitEthernet1/0/1 | include (input|output) rate
```

Lab Testing Scenarios

Scenario 1: MAC Learning Verification

```
#!/bin/bash
# Test MAC address learning

echo "Testing MAC address learning..."
```

```
# Clear MAC table
docker exec -it clab-sw-switch1 cli -c "clear mac address-table dynamic"

# Check empty table
echo "Initial MAC table:"
docker exec -it clab-sw-switch1 cli -c "show mac address-table"

# Generate traffic
docker exec clab-sw-pc1 ping -c 1 192.168.1.11

# Check learned addresses
echo "MAC table after traffic:"
docker exec -it clab-sw-switch1 cli -c "show mac address-table"
```

Scenario 2: Broadcast Domain Testing

```
#!/bin/bash
# Test broadcast domain behavior

echo "Testing broadcast domain..."

# Start packet capture on PC2
docker exec -d clab-sw-pc2 tcpdump -i eth1 -w /tmp/broadcast-test.pcap

# Generate broadcast from PC1
docker exec clab-sw-pc1 ping -b -c 3 192.168.1.255

# Stop capture and analyze
docker exec clab-sw-pc2 pkill tcpdump
docker exec clab-sw-pc2 tcpdump -r /tmp/broadcast-test.pcap
```

Scenario 3: Port Security Testing

```
#!/bin/bash
# Test port security functionality

echo "Testing port security..."

# Configure port security
docker exec -it clab-sw-switch1 cli -c "
configure terminal
interface GigabitEthernet1/0/1
switchport port-security
switchport port-security maximum 1
switchport port-security violation shutdown
end"

# Check port security status
docker exec -it clab-sw-switch1 cli -c "show port-security interface GigabitEthernet1/0/1"

# Simulate violation (would require additional setup)
echo "Port security configured. Test violation scenarios manually."
```

Performance Optimization

Switch Performance Factors

1. **MAC Table Size**: Larger tables support more devices
2. **Switching Capacity**: Backplane bandwidth and PPS rates
3. **Buffer Size**: Affects burst handling capability
4. **CPU Utilization**: Impacts control plane operations

Optimization Techniques

Interface Optimization

```
startup-config: |
  interface range GigabitEthernet1/0/1-24
   speed 1000
   duplex full
   no negotiation auto
   spanning-tree portfast
  !
```

Buffer Tuning

```
startup-config: |
  # Platform-specific buffer tuning
  platform buffer-allocation ratio 70

  # QoS buffer allocation
  mls qos queue-set output 1 buffers 10 15 70 5
```

Summary

Ethernet and switching form the foundation of modern networks. Understanding frame structure, MAC address learning, and switching operations is crucial for network engineers. ContainerLab provides an excellent platform for experimenting with these concepts in a controlled environment.

Key concepts covered: - Ethernet frame structure and MAC addressing - Switch learning and forwarding processes - Collision and broadcast domain concepts - Advanced switching features like port security and EtherChannel - Troubleshooting methodologies and tools

In the next chapter, we'll explore VLANs and trunking, which build upon these switching fundamentals to provide network segmentation and scalability.

Review Questions

1. What are the components of an Ethernet frame and their purposes?
2. How does a switch learn MAC addresses and make forwarding decisions?
3. What's the difference between collision domains and broadcast domains?
4. How does port security enhance network security?
5. What are the benefits and considerations of link aggregation?

Hands-on Exercises

Exercise 1: Basic Switching Lab

1. Deploy the basic switching lab from this chapter
2. Generate traffic between PCs and observe MAC learning
3. Use show commands to verify switch operation
4. Test broadcast behavior

Exercise 2: Port Security Implementation

1. Configure port security on switch ports
2. Test different violation actions
3. Implement sticky MAC addresses
4. Document security benefits and limitations

Exercise 3: Multi-Switch Network

1. Deploy the campus switching topology
2. Configure VLANs and trunking (preview of next chapter)
3. Test connectivity between different network segments
4. Implement and test EtherChannel

Exercise 4: Troubleshooting Scenarios

1. Create various switching problems (duplicate MACs, security violations)
2. Practice diagnostic commands and procedures
3. Develop troubleshooting methodologies
4. Document solutions and prevention strategies

Additional Resources

- Ethernet Standards (IEEE 802.3)
- Cisco Switching Configuration Guide
- Network Switching Fundamentals
- ContainerLab Switching Examples

Chapter 8: VLANs and Trunking

Learning Objectives

By the end of this chapter, you will be able to: - Understand VLAN concepts and benefits - Configure VLANs on different network operating systems - Implement VLAN trunking protocols - Configure inter-VLAN routing - Troubleshoot VLAN-related issues

VLAN Fundamentals

What are VLANs?

Virtual Local Area Networks (VLANs) are logical network segments that allow you to group devices together regardless of their physical location. VLANs operate at Layer 2 and create separate broadcast domains within a single physical switch infrastructure.

Benefits of VLANs

1. **Broadcast Domain Segmentation**: Reduces broadcast traffic
2. **Security**: Isolates sensitive traffic
3. **Flexibility**: Easy device grouping and moves
4. **Performance**: Reduces network congestion
5. **Cost Efficiency**: Eliminates need for separate physical switches

VLAN Types

Data VLANs

- Carry user-generated traffic
- Most common VLAN type
- Configured on access ports

Voice VLANs

- Dedicated to VoIP traffic
- Quality of Service (QoS) enabled
- Often configured alongside data VLANs

Management VLANs

- Used for switch management traffic
- Provides secure administrative access
- Typically VLAN 1 by default (should be changed)

Native VLANs

- Untagged traffic on trunk ports
- Default is VLAN 1
- Security best practice: change from default

Basic VLAN Configuration

Single Switch VLAN Lab

```
# Basic VLAN configuration lab
name: basic-vlan-lab
prefix: vlan

topology:
  nodes:
    switch1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname VLAN-Switch
        !
        ! Create VLANs
        vlan 10
         name SALES
        !
        vlan 20
         name ENGINEERING
        !
        vlan 30
         name MANAGEMENT
        !
        vlan 99
         name NATIVE
        !
        ! Configure access ports
        interface range GigabitEthernet1/0/1-2
         switchport mode access
```

```
            switchport access vlan 10
            spanning-tree portfast
            no shutdown
           !
           interface range GigabitEthernet1/0/3-4
            switchport mode access
            switchport access vlan 20
            spanning-tree portfast
            no shutdown
           !
           interface GigabitEthernet1/0/5
            switchport mode access
            switchport access vlan 30
            spanning-tree portfast
            no shutdown
           !
           ! Management VLAN interface
           interface vlan 30
            ip address 192.168.30.10 255.255.255.0
            no shutdown
           !
           ip default-gateway 192.168.30.1
           !

       # Sales department PCs
       sales-pc1:
         kind: linux
         image: alpine:latest
         mgmt-ipv4: 172.20.20.11
         exec:
           - ip addr add 192.168.10.10/24 dev eth1
           - ip route add default via 192.168.10.1

       sales-pc2:
         kind: linux
         image: alpine:latest
         mgmt-ipv4: 172.20.20.12
         exec:
           - ip addr add 192.168.10.11/24 dev eth1
           - ip route add default via 192.168.10.1

       # Engineering department PCs
       eng-pc1:
         kind: linux
         image: alpine:latest
         mgmt-ipv4: 172.20.20.13
         exec:
```

```
        - ip addr add 192.168.20.10/24 dev eth1
        - ip route add default via 192.168.20.1

    eng-pc2:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.14
      exec:
        - ip addr add 192.168.20.11/24 dev eth1
        - ip route add default via 192.168.20.1

    # Management PC
    mgmt-pc:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.15
      exec:
        - ip addr add 192.168.30.20/24 dev eth1
        - ip route add default via 192.168.30.1

  links:
    # Sales VLAN connections
    - endpoints: ["switch1:eth1", "sales-pc1:eth1"]
    - endpoints: ["switch1:eth2", "sales-pc2:eth1"]

    # Engineering VLAN connections
    - endpoints: ["switch1:eth3", "eng-pc1:eth1"]
    - endpoints: ["switch1:eth4", "eng-pc2:eth1"]

    # Management VLAN connection
    - endpoints: ["switch1:eth5", "mgmt-pc:eth1"]
```

Testing VLAN Isolation

```
# Deploy the lab
containerlab deploy -t basic-vlan-lab.yml

# Test connectivity within same VLAN (should work)
docker exec clab-vlan-sales-pc1 ping -c 3 192.168.10.11

# Test connectivity between different VLANs (should fail)
docker exec clab-vlan-sales-pc1 ping -c 3 192.168.20.10

# Check VLAN configuration on switch
docker exec -it clab-vlan-switch1 cli -c "show vlan brief"
```

VLAN Configuration Commands

Creating VLANs

```
# Cisco IOS/IOS-XE
configure terminal
vlan 10
 name SALES
 exit
vlan 20
 name ENGINEERING
 exit

# Alternative method
vlan database
vlan 10 name SALES
vlan 20 name ENGINEERING
exit
```

Assigning Ports to VLANs

```
# Access port configuration
interface GigabitEthernet1/0/1
 switchport mode access
 switchport access vlan 10
 no shutdown

# Range configuration
interface range GigabitEthernet1/0/1-5
 switchport mode access
 switchport access vlan 10
```

VLAN Trunking

Trunk Port Fundamentals

Trunk ports carry traffic for multiple VLANs between switches. They use VLAN tagging to identify which VLAN each frame belongs to.

VLAN Tagging Protocols

IEEE 802.1Q (Dot1Q) - Industry standard - 4-byte tag inserted into Ethernet frame - Supports up to 4,094 VLANs - Native VLAN concept for untagged traffic

ISL (Inter-Switch Link) - Cisco proprietary (legacy) - Encapsulates entire frame - 30-byte overhead - Being phased out

802.1Q Frame Format

```
Original Frame:
| Dest MAC | Src MAC | EtherType | Data | FCS |

Tagged Frame:
| Dest MAC | Src MAC | 802.1Q Tag | EtherType | Data | FCS |

802.1Q Tag (4 bytes):
| TPID (2B) | TCI (2B) |
            | PCP(3b) | DEI(1b) | VID(12b) |

TPID: Tag Protocol Identifier (0x8100)
PCP: Priority Code Point (QoS)
DEI: Drop Eligible Indicator
VID: VLAN Identifier (1-4094)
```

Multi-Switch VLAN Lab

```
# Multi-switch VLAN with trunking
name: multi-switch-vlan
prefix: trunk

topology:
  nodes:
    # Core switch
    core-sw:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Core-Switch
        !
        ! Create VLANs
        vlan 10
         name SALES
        !
        vlan 20
         name ENGINEERING
        !
        vlan 30
         name MANAGEMENT
        !
        vlan 99
         name NATIVE
        !
        ! Configure trunk ports
        interface range GigabitEthernet1/0/1-2
         switchport mode trunk
         switchport trunk encapsulation dot1q
         switchport trunk native vlan 99
         switchport trunk allowed vlan 10,20,30,99
         no shutdown
        !
        ! Management interface
        interface vlan 30
         ip address 192.168.30.1 255.255.255.0
         no shutdown
        !

    # Access switch 1 (Sales)
    access-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Access-Switch-1
        !
        ! Create VLANs
        vlan 10
         name SALES
        !
        vlan 99
         name NATIVE
        !
        ! Trunk to core
        interface GigabitEthernet1/0/1
         switchport mode trunk
         switchport trunk encapsulation dot1q
         switchport trunk native vlan 99
         switchport trunk allowed vlan 10,99
         no shutdown
        !
        ! Access ports for sales
        interface range GigabitEthernet1/0/2-5
         switchport mode access
         switchport access vlan 10
         spanning-tree portfast
         no shutdown
        !
```

```
# Access switch 2 (Engineering)
access-sw2:
  kind: cisco_iosxe
  image: cisco/catalyst:latest
  mgmt-ipv4: 172.20.20.12
  startup-config: |
    hostname Access-Switch-2
    !
    ! Create VLANs
    vlan 20
     name ENGINEERING
    !
    vlan 99
     name NATIVE
    !
    ! Trunk to core
    interface GigabitEthernet1/0/1
     switchport mode trunk
     switchport trunk encapsulation dot1q
     switchport trunk native vlan 99
     switchport trunk allowed vlan 20,99
     no shutdown
    !
    ! Access ports for engineering
    interface range GigabitEthernet1/0/2-5
     switchport mode access
     switchport access vlan 20
     spanning-tree portfast
     no shutdown
    !

# Router for inter-VLAN routing
router1:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.20
  startup-config: |
    hostname Inter-VLAN-Router
    !
    ! Configure trunk interface with subinterfaces
    interface GigabitEthernet0/0/0
     no ip address
     no shutdown
    !
    interface GigabitEthernet0/0/0.10
     description Sales-VLAN
```

```
          encapsulation dot1Q 10
          ip address 192.168.10.1 255.255.255.0
         !
         interface GigabitEthernet0/0/0.20
          description Engineering-VLAN
          encapsulation dot1Q 20
          ip address 192.168.20.1 255.255.255.0
         !
         interface GigabitEthernet0/0/0.30
          description Management-VLAN
          encapsulation dot1Q 30
          ip address 192.168.30.1 255.255.255.0
         !

    # End devices
    sales-pc1:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.10/24 dev eth1
        - ip route add default via 192.168.10.1

    sales-pc2:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.11/24 dev eth1
        - ip route add default via 192.168.10.1

    eng-pc1:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.20.10/24 dev eth1
        - ip route add default via 192.168.20.1

    eng-pc2:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.20.11/24 dev eth1
        - ip route add default via 192.168.20.1

  links:
    # Trunk connections
    - endpoints: ["core-sw:eth1", "access-sw1:eth1"]
    - endpoints: ["core-sw:eth2", "access-sw2:eth1"]
    - endpoints: ["core-sw:eth3", "router1:eth1"]

    # Access connections
    - endpoints: ["access-sw1:eth2", "sales-pc1:eth1"]
    - endpoints: ["access-sw1:eth3", "sales-pc2:eth1"]
    - endpoints: ["access-sw2:eth2", "eng-pc1:eth1"]
    - endpoints: ["access-sw2:eth3", "eng-pc2:eth1"]
```

Trunk Configuration Commands

Basic Trunk Configuration

```
# Configure trunk port
interface GigabitEthernet1/0/1
 switchport mode trunk
 switchport trunk encapsulation dot1q
 switchport trunk native vlan 99
 switchport trunk allowed vlan 10,20,30
 no shutdown
```

Trunk Verification

```
# Show trunk status
show interfaces trunk
show interfaces GigabitEthernet1/0/1 trunk

# Show VLAN information
show vlan brief
show vlan id 10

# Show interface switchport status
show interfaces GigabitEthernet1/0/1 switchport
```

Dynamic Trunking Protocol (DTP)

DTP is a Cisco proprietary protocol that automatically negotiates trunking between switches.

DTP Modes

Mode	Description	Behavior
trunk	Permanent trunk	Always trunk, sends DTP
access	Permanent access	Never trunk, ignores DTP

Mode	Description	Behavior
dynamic auto	Passive negotiation	Trunk if neighbor is trunk/desirable
dynamic desirable	Active negotiation	Actively tries to form trunk
nonegotiate	No DTP	Trunk without DTP negotiation

```
# Configure DTP modes
interface GigabitEthernet1/0/1
 switchport mode dynamic desirable

interface GigabitEthernet1/0/2
 switchport mode dynamic auto

# Disable DTP (security best practice)
interface GigabitEthernet1/0/1
 switchport mode trunk
 switchport nonegotiate
```

Inter-VLAN Routing

Router-on-a-Stick

Traditional method using a single router interface with subinterfaces for each VLAN.

Subinterface Configuration

```
# Configure physical interface
interface GigabitEthernet0/0/0
 no ip address
 no shutdown

# Configure subinterfaces
interface GigabitEthernet0/0/0.10
 description Sales-VLAN
 encapsulation dot1Q 10
 ip address 192.168.10.1 255.255.255.0

interface GigabitEthernet0/0/0.20
 description Engineering-VLAN
 encapsulation dot1Q 20
 ip address 192.168.20.1 255.255.255.0
```

Switch Virtual Interfaces (SVIs)

Layer 3 switches can route between VLANs using SVIs.

```
# Layer 3 switch inter-VLAN routing
startup-config: |
  ! Enable IP routing
  ip routing
  !
  ! Create VLANs
  vlan 10
   name SALES
  !
  vlan 20
   name ENGINEERING
  !
  ! Configure SVIs
  interface vlan 10
   ip address 192.168.10.1 255.255.255.0
   no shutdown
  !
  interface vlan 20
   ip address 192.168.20.1 255.255.255.0
   no shutdown
  !
  ! Configure access ports
  interface range GigabitEthernet1/0/1-10
   switchport mode access
   switchport access vlan 10
  !
  interface range GigabitEthernet1/0/11-20
   switchport mode access
   switchport access vlan 20
  !
```

VLAN Security

VLAN Hopping Attacks

Switch Spoofing Attack

Attacker configures device to trunk and gains access to all VLANs.

Prevention:

```
# Explicitly configure access ports
interface GigabitEthernet1/0/1
 switchport mode access
 switchport nonegotiate

# Disable unused ports
interface range GigabitEthernet1/0/20-24
 shutdown
 switchport mode access
 switchport access vlan 999  # Unused VLAN
```

Double Tagging Attack

Attacker sends frames with two 802.1Q tags to access different VLANs.

Prevention:

```
# Change native VLAN from default
interface GigabitEthernet1/0/1
 switchport trunk native vlan 999

# Use dedicated native VLAN
vlan 999
 name NATIVE_VLAN
```

VLAN Security Best Practices

1. **Change default native VLAN** from VLAN 1
2. **Disable unused ports** and assign to unused VLAN
3. **Use explicit trunk configuration** (avoid DTP)
4. **Implement port security** on access ports
5. **Separate management traffic** into dedicated VLAN
6. **Regular VLAN audits** and cleanup

```
# Security hardening example
! Change native VLAN
interface range GigabitEthernet1/0/1-2
 switchport trunk native vlan 999
 switchport nonegotiate

! Secure unused ports
interface range GigabitEthernet1/0/20-24
 shutdown
 switchport mode access
 switchport access vlan 999
```

```
! Port security on access ports
interface range GigabitEthernet1/0/3-10
 switchport port-security
 switchport port-security maximum 2
 switchport port-security violation shutdown
```

Voice VLANs

Voice VLANs provide dedicated bandwidth and QoS for VoIP traffic.

Voice VLAN Configuration

```
# Configure voice VLAN
interface GigabitEthernet1/0/1
 switchport mode access
 switchport access vlan 10        # Data VLAN
 switchport voice vlan 20         # Voice VLAN
 spanning-tree portfast

# QoS for voice traffic
mls qos trust cos
```

Voice VLAN Lab Example

```
# Voice VLAN demonstration
startup-config: |
  ! Create VLANs
  vlan 10
   name DATA
  !
  vlan 20
   name VOICE
  !
  ! Configure voice-enabled port
  interface GigabitEthernet1/0/1
   switchport mode access
   switchport access vlan 10
   switchport voice vlan 20
   spanning-tree portfast
   mls qos trust cos
  !
```

Troubleshooting VLANs

Common VLAN Issues

VLAN Mismatch

```
# Symptoms
- Devices in same VLAN cannot communicate
- Trunk not passing traffic for specific VLAN

# Diagnosis
show vlan brief
show interfaces trunk

# Resolution
- Verify VLAN exists on both switches
- Check trunk allowed VLAN list
- Verify VLAN is active
```

Native VLAN Mismatch

```
# Symptoms
- CDP/LLDP native VLAN mismatch messages
- Connectivity issues for untagged traffic

# Diagnosis
show interfaces GigabitEthernet1/0/1 trunk
show cdp neighbors detail

# Resolution
- Configure same native VLAN on both ends
- Or use different native VLANs intentionally
```

Trunk Negotiation Issues

```
# Symptoms
- Trunk not forming
- DTP negotiation failures

# Diagnosis
show interfaces GigabitEthernet1/0/1 switchport
show dtp interface GigabitEthernet1/0/1

# Resolution
```

```
- Configure explicit trunk mode
- Disable DTP if not needed
- Check encapsulation compatibility
```

Diagnostic Commands

VLAN Verification

```
# VLAN information
show vlan brief
show vlan id 10
show vlan name SALES

# Interface VLAN assignment
show interfaces GigabitEthernet1/0/1 switchport
show interfaces status

# Trunk verification
show interfaces trunk
show interfaces GigabitEthernet1/0/1 trunk
```

Traffic Analysis

```
# MAC address table per VLAN
show mac address-table vlan 10
show mac address-table interface GigabitEthernet1/0/1

# Spanning tree per VLAN
show spanning-tree vlan 10
show spanning-tree interface GigabitEthernet1/0/1

# Interface counters
show interfaces GigabitEthernet1/0/1 counters
```

Advanced VLAN Features

Private VLANs

Private VLANs provide Layer 2 isolation within a VLAN.

Private VLAN Types

- **Primary VLAN**: Main VLAN containing all ports
- **Isolated VLAN**: Ports can only communicate with promiscuous ports
- **Community VLAN**: Ports can communicate with each other and promiscuous ports
- **Promiscuous VLAN**: Can communicate with all VLAN types

```
# Private VLAN configuration
vlan 100
 private-vlan primary
!
vlan 101
 private-vlan isolated
!
vlan 102
 private-vlan community
!
vlan 100
 private-vlan association 101,102
!
interface GigabitEthernet1/0/1
 switchport mode private-vlan promiscuous
 switchport private-vlan mapping 100 101,102
!
interface GigabitEthernet1/0/2
 switchport mode private-vlan host
 switchport private-vlan host-association 100 101
```

VLAN Access Control Lists

VACLs filter traffic within a VLAN.

```
# Configure VACL
ip access-list extended BLOCK_HTTP
 deny tcp any any eq 80
 permit ip any any
!
vlan access-map VLAN_FILTER 10
 match ip address BLOCK_HTTP
 action drop
!
vlan access-map VLAN_FILTER 20
 action forward
!
vlan filter VLAN_FILTER vlan-list 10
```

Performance Optimization

VLAN Performance Considerations

1. **VLAN Spanning**: Minimize VLANs spanning multiple switches
2. **Trunk Utilization**: Monitor trunk link utilization
3. **STP Optimization**: Optimize spanning tree per VLAN
4. **Load Balancing**: Distribute VLANs across trunk links

Monitoring and Optimization

```
# Monitor trunk utilization
show interfaces GigabitEthernet1/0/1 | include rate

# Check spanning tree load balancing
show spanning-tree vlan 10 | include Root

# VLAN statistics
show vlan counters
```

Summary

VLANs are fundamental to modern network design, providing segmentation, security, and flexibility. Understanding VLAN configuration, trunking protocols, and inter-VLAN routing is essential for network engineers. Proper VLAN design and security implementation help create scalable and secure network infrastructures.

Key concepts covered: - VLAN fundamentals and benefits - VLAN configuration and management - Trunking protocols and configuration - Inter-VLAN routing methods - VLAN security considerations - Troubleshooting methodologies

In the next chapter, we'll explore Spanning Tree Protocol, which prevents loops in switched networks with redundant paths.

Review Questions

1. What are the benefits of implementing VLANs in a network?
2. How does 802.1Q tagging work in trunk links?
3. What are the security risks associated with VLANs and how can they be mitigated?
4. What's the difference between router-on-a-stick and SVI inter-VLAN routing?
5. How do you troubleshoot VLAN connectivity issues?

Hands-on Exercises

Exercise 1: Basic VLAN Implementation

1. Deploy the basic VLAN lab topology
2. Configure VLANs and assign ports
3. Test VLAN isolation between different VLANs
4. Verify VLAN configuration with show commands

Exercise 2: Multi-Switch VLAN with Trunking

1. Implement the multi-switch VLAN topology
2. Configure trunk links between switches
3. Test VLAN connectivity across switches
4. Implement inter-VLAN routing

Exercise 3: VLAN Security Hardening

1. Implement VLAN security best practices
2. Test and prevent VLAN hopping attacks
3. Configure voice VLANs with QoS
4. Document security improvements

Exercise 4: VLAN Troubleshooting

1. Create various VLAN problems (misconfigurations, trunk issues)
2. Practice diagnostic commands and procedures
3. Develop systematic troubleshooting approaches
4. Document solutions and prevention strategies

Additional Resources

- IEEE 802.1Q Standard
- Cisco VLAN Configuration Guide
- VLAN Security Best Practices
- Inter-VLAN Routing Guide

Chapter 11: Static and Dynamic Routing

Learning Objectives

By the end of this chapter, you will be able to: - Configure and troubleshoot static routing - Understand routing table concepts and administrative distance - Implement default routes and route summarization - Compare static and dynamic routing approaches - Design routing solutions for different network scenarios

Routing Fundamentals

What is Routing?

Routing is the process of selecting paths in a network along which to send network traffic. Routers use routing tables to determine the best path to forward packets toward their destination.

Key Routing Concepts

- **Routing Table**: Database containing network destinations and next-hop information
- **Administrative Distance**: Trustworthiness rating of routing information sources
- **Metric**: Value used to determine the best path when multiple routes exist
- **Next Hop**: The next router in the path toward the destination
- **Route Summarization**: Combining multiple routes into a single advertisement

Routing Table Structure

```
# Example routing table output
Router# show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override

Gateway of last resort is 192.168.1.1 to network 0.0.0.0
```

```
S*    0.0.0.0/0 [1/0] via 192.168.1.1
      10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C        10.1.1.0/24 is directly connected, GigabitEthernet0/0/0
L        10.1.1.1/32 is directly connected, GigabitEthernet0/0/0
S        10.2.2.0/24 [1/0] via 10.1.1.2
S        10.3.3.0/24 [1/0] via 10.1.1.3
```

Static Routing

Static Route Configuration

Static routes are manually configured routes that don't change unless manually modified.

Basic Static Route Lab

```
# Static routing lab topology
name: static-routing-lab
prefix: static

topology:
  nodes:
    r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Router1
        !
        interface GigabitEthernet0/0/0
         ip address 10.1.1.1 255.255.255.0
         no shutdown
        !
        interface GigabitEthernet0/0/1
         ip address 192.168.1.1 255.255.255.0
         no shutdown
        !

    r2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Router2
        !
        interface GigabitEthernet0/0/0
         ip address 10.1.1.2 255.255.255.0
         no shutdown
        !
        interface GigabitEthernet0/0/1
         ip address 192.168.2.1 255.255.255.0
         no shutdown
        !

    r3:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.12
      startup-config: |
        hostname Router3
        !
        interface GigabitEthernet0/0/0
         ip address 10.1.1.3 255.255.255.0
         no shutdown
        !
        interface GigabitEthernet0/0/1
         ip address 192.168.3.1 255.255.255.0
         no shutdown
        !

    pc1:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.21
      exec:
        - ip addr add 192.168.1.10/24 dev eth1
        - ip route add default via 192.168.1.1

    pc2:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.22
      exec:
        - ip addr add 192.168.2.10/24 dev eth1
        - ip route add default via 192.168.2.1

    pc3:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.23
      exec:
        - ip addr add 192.168.3.10/24 dev eth1
```

```
        - ip route add default via 192.168.3.1

  links:
    # Router interconnections
    - endpoints: ["r1:eth1", "r2:eth1"]
    - endpoints: ["r1:eth1", "r3:eth1"]  # Multi-access network

    # PC connections
    - endpoints: ["r1:eth2", "pc1:eth1"]
    - endpoints: ["r2:eth2", "pc2:eth1"]
    - endpoints: ["r3:eth2", "pc3:eth1"]
```

Configuring Static Routes

```
# Deploy the lab
containerlab deploy -t static-routing-lab.yml

# Connect to Router1 and configure static routes
docker exec -it clab-static-r1 cli

# Configure static routes on R1
configure terminal
ip route 192.168.2.0 255.255.255.0 10.1.1.2
ip route 192.168.3.0 255.255.255.0 10.1.1.3
exit

# Verify routing table
show ip route

# Test connectivity
ping 192.168.2.10
ping 192.168.3.10
```

Static Route Types

Standard Static Routes

```
# Basic static route syntax
ip route <destination_network> <subnet_mask> <next_hop_ip>

# Examples
ip route 192.168.2.0 255.255.255.0 10.1.1.2
ip route 10.0.0.0 255.0.0.0 172.16.1.1
```

Default Routes

```
# Default route (gateway of last resort)
ip route 0.0.0.0 0.0.0.0 <next_hop_ip>

# Example
ip route 0.0.0.0 0.0.0.0 192.168.1.1
```

Floating Static Routes

Backup routes with higher administrative distance:

```
# Primary route (AD = 1)
ip route 192.168.2.0 255.255.255.0 10.1.1.2

# Backup route (AD = 5)
ip route 192.168.2.0 255.255.255.0 10.1.1.3 5
```

Interface-Based Static Routes

```
# Route pointing to an interface (for point-to-point links)
ip route 192.168.2.0 255.255.255.0 Serial0/0/0

# Fully specified route (interface + next-hop)
ip route 192.168.2.0 255.255.255.0 Serial0/0/0 10.1.1.2
```

Advanced Static Routing Lab

```
# Advanced static routing with redundancy
name: advanced-static-routing
prefix: adv-static

topology:
  nodes:
    # Core routers
    core-r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Core-Router-1
        !
        interface GigabitEthernet0/0/0
```

```
         description To-Core-R2-Primary
         ip address 10.1.1.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description To-Core-R2-Backup
         ip address 10.1.2.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description To-Branch-R1
         ip address 10.2.1.1 255.255.255.252
         no shutdown
        !
        interface Loopback0
         ip address 1.1.1.1 255.255.255.255
        !
        ! Static routes with redundancy
        ip route 2.2.2.2 255.255.255.255 10.1.1.2
        ip route 2.2.2.2 255.255.255.255 10.1.2.2 5
        ip route 192.168.10.0 255.255.255.0 10.2.1.2
        ip route 0.0.0.0 0.0.0.0 10.1.1.2
        ip route 0.0.0.0 0.0.0.0 10.1.2.2 5
        !

    core-r2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Core-Router-2
        !
        interface GigabitEthernet0/0/0
         description To-Core-R1-Primary
         ip address 10.1.1.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description To-Core-R1-Backup
         ip address 10.1.2.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description To-Internet
         ip address 203.0.113.1 255.255.255.252
         no shutdown
        !
        interface Loopback0
         ip address 2.2.2.2 255.255.255.255
        !
        ! Static routes
        ip route 1.1.1.1 255.255.255.255 10.1.1.1
        ip route 1.1.1.1 255.255.255.255 10.1.2.1 5
        ip route 192.168.10.0 255.255.255.0 10.1.1.1
        ip route 192.168.10.0 255.255.255.0 10.1.2.1 5
        ip route 0.0.0.0 0.0.0.0 203.0.113.2
        !

    branch-r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.12
      startup-config: |
        hostname Branch-Router-1
        !
        interface GigabitEthernet0/0/0
         description To-Core-R1
         ip address 10.2.1.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description LAN-Interface
         ip address 192.168.10.1 255.255.255.0
         no shutdown
        !
        ! Static routes
        ip route 0.0.0.0 0.0.0.0 10.2.1.1
        !

    internet-sim:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.100
      exec:
        - ip addr add 203.0.113.2/30 dev eth1
        - ip route add 0.0.0.0/0 via 203.0.113.1

    branch-pc:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.20
      exec:
        - ip addr add 192.168.10.10/24 dev eth1
        - ip route add default via 192.168.10.1
```

```
links:
  # Core router connections (redundant)
  - endpoints: ["core-r1:eth1", "core-r2:eth1"]
  - endpoints: ["core-r1:eth2", "core-r2:eth2"]

  # Branch connection
  - endpoints: ["core-r1:eth3", "branch-r1:eth1"]

  # Internet connection
  - endpoints: ["core-r2:eth3", "internet-sim:eth1"]

  # Branch LAN
  - endpoints: ["branch-r1:eth2", "branch-pc:eth1"]
```

Administrative Distance

Administrative Distance (AD) determines the trustworthiness of routing information sources. Lower AD values are preferred.

Default Administrative Distances

Route Source	Administrative Distance
Connected Interface	0
Static Route	1
EIGRP	90
OSPF	110
RIP	120
External EIGRP	170
Unknown	255 (unreachable)

Configuring Administrative Distance

```
# Static route with custom AD
ip route 192.168.1.0 255.255.255.0 10.1.1.2 150

# Modify AD for routing protocols
router ospf 1
 distance 95

router eigrp 100
 distance eigrp 85 155
```

Route Summarization

Route summarization reduces routing table size by combining multiple routes into a single advertisement.

Manual Route Summarization

```
# Instead of advertising individual subnets:
# 192.168.1.0/24
# 192.168.2.0/24
# 192.168.3.0/24
# 192.168.4.0/24

# Advertise summary route:
ip route 192.168.0.0 255.255.252.0 10.1.1.2
```

Summarization Lab Example

```
# Route summarization lab
name: route-summarization
topology:
  nodes:
    hub-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        hostname Hub-Router
        !
        interface GigabitEthernet0/0/0
         ip address 10.1.1.1 255.255.255.0
         no shutdown
        !
        ! Summary route instead of individual routes
        ip route 192.168.0.0 255.255.252.0 10.1.1.2
        !

    spoke-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        hostname Spoke-Router
        !
        interface GigabitEthernet0/0/0
         ip address 10.1.1.2 255.255.255.0
```

```
          no shutdown
         !
         interface GigabitEthernet0/0/1
          ip address 192.168.1.1 255.255.255.0
          no shutdown
         !
         interface GigabitEthernet0/0/2
          ip address 192.168.2.1 255.255.255.0
          no shutdown
         !
         interface GigabitEthernet0/0/3
          ip address 192.168.3.1 255.255.255.0
          no shutdown
         !
         ! Individual routes back to hub
         ip route 0.0.0.0 0.0.0.0 10.1.1.1
         !

    links:
      - endpoints: ["hub-router:eth1", "spoke-router:eth1"]
```

Dynamic Routing Overview

Dynamic routing protocols automatically discover and maintain routing information.

Types of Dynamic Routing Protocols

Distance Vector Protocols

- **RIP (Routing Information Protocol)**
- **EIGRP (Enhanced Interior Gateway Routing Protocol)**

Characteristics: - Share routing tables with neighbors - Use hop count or composite metrics - Prone to routing loops - Slower convergence

Link State Protocols

- **OSPF (Open Shortest Path First)**
- **IS-IS (Intermediate System to Intermediate System)**

Characteristics: - Share link state information - Build topology database - Use SPF algorithm - Faster convergence - More CPU and memory intensive

Comparison: Static vs Dynamic Routing

Aspect	Static Routing	Dynamic Routing
Configuration	Manual	Automatic
Scalability	Poor	Good
Convergence	None	Fast
Resource Usage	Low	Higher
Security	High	Medium
Maintenance	High	Low
Flexibility	Low	High

When to Use Static Routing

1. **Small networks** with few routers
2. **Stub networks** with single exit point
3. **Security-critical** environments
4. **Bandwidth-constrained** links
5. **Default routes** to ISPs

When to Use Dynamic Routing

1. **Large networks** with many routers
2. **Redundant paths** requiring automatic failover
3. **Frequently changing** topologies
4. **Load balancing** requirements
5. **Complex network** designs

Routing Troubleshooting

Common Routing Issues

Missing Routes

```
# Symptoms
ping fails to remote networks
traceroute shows incomplete path

# Diagnosis
show ip route
show ip route <destination>

# Resolution
Add missing static routes
```

```
Verify next-hop reachability
```

Routing Loops

```
# Symptoms
High CPU utilization
Packets with decreasing TTL
Intermittent connectivity

# Diagnosis
traceroute shows repeating hops
show ip route shows conflicting routes

# Resolution
Verify route summarization
Check administrative distances
Implement route filtering
```

Suboptimal Routing

```
# Symptoms
Poor performance
Unexpected path selection

# Diagnosis
show ip route
traceroute analysis
show ip route <destination> longer-prefixes

# Resolution
Adjust administrative distances
Modify route metrics
Implement policy routing
```

Diagnostic Commands

Essential Routing Commands

```
# Routing table
show ip route
show ip route summary
show ip route <network>
```

```
# Interface status
show ip interface brief
show interfaces

# Connectivity testing
ping <destination>
traceroute <destination>
telnet <destination> <port>

# Route verification
show ip route <destination> longer-prefixes
show ip cef <destination>
```

Advanced Troubleshooting

```
# Debug routing (use carefully)
debug ip routing
debug ip packet

# Route monitoring
show ip route summary
show ip route profile

# Interface statistics
show interfaces <interface> | include (input|output) rate
show interfaces <interface> counters
```

Practical Routing Scenarios

Scenario 1: Hub-and-Spoke with Backup

```
# Hub-and-spoke with backup connectivity
name: hub-spoke-backup
topology:
  nodes:
    hub:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        hostname Hub
        !
        interface GigabitEthernet0/0/0
```

```
     description Primary-to-Spoke1
     ip address 10.1.1.1 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description Primary-to-Spoke2
     ip address 10.1.2.1 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/2
     description Backup-Link
     ip address 10.1.3.1 255.255.255.252
     no shutdown
    !
    ! Primary routes
    ip route 192.168.1.0 255.255.255.0 10.1.1.2
    ip route 192.168.2.0 255.255.255.0 10.1.2.2
    ! Backup routes (higher AD)
    ip route 192.168.1.0 255.255.255.0 10.1.3.2 5
    ip route 192.168.2.0 255.255.255.0 10.1.3.2 5
    !

spoke1:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  startup-config: |
    hostname Spoke1
    !
    interface GigabitEthernet0/0/0
     description Primary-to-Hub
     ip address 10.1.1.2 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description LAN
     ip address 192.168.1.1 255.255.255.0
     no shutdown
    !
    interface GigabitEthernet0/0/2
     description Backup-to-Spoke2
     ip address 10.1.3.2 255.255.255.252
     no shutdown
    !
    ! Default route via hub
    ip route 0.0.0.0 0.0.0.0 10.1.1.1
    ! Backup default route
    ip route 0.0.0.0 0.0.0.0 10.1.3.1 5
```

```
        !

    spoke2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        hostname Spoke2
        !
        interface GigabitEthernet0/0/0
         description Primary-to-Hub
         ip address 10.1.2.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description LAN
         ip address 192.168.2.1 255.255.255.0
         no shutdown
        !
        ! Default route via hub
        ip route 0.0.0.0 0.0.0.0 10.1.2.1
        !

  links:
    - endpoints: ["hub:eth1", "spoke1:eth1"]
    - endpoints: ["hub:eth2", "spoke2:eth1"]
    - endpoints: ["hub:eth3", "spoke1:eth3"]  # Backup link
```

Scenario 2: Load Balancing

```
# Equal-cost load balancing
startup-config: |
  ! Multiple equal-cost paths
  ip route 192.168.10.0 255.255.255.0 10.1.1.2
  ip route 192.168.10.0 255.255.255.0 10.1.2.2

  ! Enable load balancing
  maximum-paths 4
```

Best Practices

Static Routing Best Practices

1. **Documentation**: Maintain accurate network diagrams and route tables
2. **Consistency**: Use consistent naming and addressing schemes

3. **Redundancy**: Implement backup routes with appropriate AD
4. **Summarization**: Use route summarization to reduce table size
5. **Security**: Implement route filtering and access controls

Route Management

1. **Change Control**: Document all routing changes
2. **Testing**: Test routes in lab before production
3. **Monitoring**: Monitor routing table size and convergence
4. **Backup**: Maintain configuration backups
5. **Automation**: Use scripts for repetitive tasks

Performance Considerations

1. **Route Table Size**: Monitor and optimize table size
2. **Convergence Time**: Minimize convergence delays
3. **CPU Usage**: Monitor routing process CPU utilization
4. **Memory Usage**: Track routing table memory consumption
5. **Bandwidth**: Consider routing protocol overhead

Summary

Static and dynamic routing serve different purposes in network design. Static routing provides simplicity and security for small networks and specific use cases, while dynamic routing offers scalability and automatic adaptation for larger, more complex networks. Understanding both approaches and their appropriate applications is essential for effective network design.

Key concepts covered: - Static route configuration and types - Administrative distance and route selection - Route summarization techniques - Comparison of static vs dynamic routing - Troubleshooting methodologies - Practical implementation scenarios

In the next chapter, we'll dive deep into OSPF, one of the most important dynamic routing protocols for enterprise networks.

Review Questions

1. What are the advantages and disadvantages of static routing?
2. How does administrative distance affect route selection?
3. When would you use floating static routes?
4. What are the benefits of route summarization?
5. How do you troubleshoot routing connectivity issues?

Hands-on Exercises

Exercise 1: Basic Static Routing

1. Deploy the static routing lab topology
2. Configure static routes on all routers
3. Test connectivity between all networks
4. Verify routing tables and trace packet paths

Exercise 2: Redundant Static Routes

1. Implement the advanced static routing lab
2. Configure primary and backup routes
3. Test failover by disabling primary links
4. Monitor route table changes during failover

Exercise 3: Route Summarization

1. Create a network with multiple subnets
2. Implement route summarization
3. Compare routing table sizes before and after
4. Test connectivity to verify summarization works

Exercise 4: Troubleshooting Scenarios

1. Create various routing problems (missing routes, loops)
2. Practice diagnostic commands and procedures
3. Develop systematic troubleshooting approaches
4. Document solutions and prevention strategies

Additional Resources

- Cisco Routing Configuration Guide
- Static Routing Best Practices
- Administrative Distance Reference
- Route Summarization Guide

Chapter 12: OSPF Configuration and Troubleshooting

Learning Objectives

By the end of this chapter, you will be able to: - Understand OSPF fundamentals and operation - Configure single-area and multi-area OSPF - Implement OSPF authentication and security - Troubleshoot OSPF routing issues - Optimize OSPF performance and convergence

OSPF Fundamentals

What is OSPF?

Open Shortest Path First (OSPF) is a link-state routing protocol that uses the Shortest Path First (SPF) algorithm to calculate the best paths through a network. OSPF is an Interior Gateway Protocol (IGP) designed for use within an autonomous system.

Key OSPF Characteristics

- **Link-State Protocol**: Maintains complete network topology
- **Classless**: Supports VLSM and CIDR
- **Fast Convergence**: Rapid response to network changes
- **Scalable**: Hierarchical design with areas
- **Standards-Based**: Open standard (RFC 2328)
- **Load Balancing**: Equal-cost multipath support

OSPF Operation Overview

OSPF Process

1. **Neighbor Discovery**: Find adjacent routers
2. **Database Synchronization**: Exchange link-state information
3. **SPF Calculation**: Calculate shortest paths
4. **Routing Table Update**: Install best routes

OSPF Packet Types

Type	Name	Purpose
1	Hello	Neighbor discovery and maintenance
2	Database Description (DBD)	Database synchronization
3	Link State Request (LSR)	Request specific LSAs
4	Link State Update (LSU)	Send LSAs
5	Link State Acknowledgment (LSAck)	Acknowledge LSAs

OSPF Areas

OSPF uses areas to create a hierarchical network design that improves scalability and reduces routing overhead.

Area Types

- **Backbone Area (Area 0)**: Central area, all other areas must connect
- **Standard Area**: Normal area with full LSA database
- **Stub Area**: Blocks external LSAs, uses default route
- **Totally Stubby Area**: Blocks external and summary LSAs
- **Not-So-Stubby Area (NSSA)**: Allows limited external routes

Single-Area OSPF Configuration

Basic OSPF Lab

```
# Single-area OSPF topology
name: ospf-single-area
prefix: ospf

topology:
  nodes:
    r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname R1
        !
        interface Loopback0
         ip address 1.1.1.1 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description To-R2
         ip address 10.1.12.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description To-R3
         ip address 10.1.13.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description LAN
         ip address 192.168.1.1 255.255.255.0
         no shutdown
        !
        router ospf 1
         router-id 1.1.1.1
         network 1.1.1.1 0.0.0.0 area 0
         network 10.1.12.0 0.0.0.3 area 0
         network 10.1.13.0 0.0.0.3 area 0
         network 192.168.1.0 0.0.0.255 area 0
        !

    r2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname R2
        !
        interface Loopback0
         ip address 2.2.2.2 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description To-R1
         ip address 10.1.12.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description To-R3
         ip address 10.1.23.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description LAN
         ip address 192.168.2.1 255.255.255.0
         no shutdown
        !
```

```
    router ospf 1
     router-id 2.2.2.2
     network 2.2.2.2 0.0.0.0 area 0
     network 10.1.12.0 0.0.0.3 area 0
     network 10.1.23.0 0.0.0.3 area 0
     network 192.168.2.0 0.0.0.255 area 0
    !

r3:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.12
  startup-config: |
    hostname R3
    !
    interface Loopback0
     ip address 3.3.3.3 255.255.255.255
    !
    interface GigabitEthernet0/0/0
     description To-R1
     ip address 10.1.13.2 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description To-R2
     ip address 10.1.23.1 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/2
     description LAN
     ip address 192.168.3.1 255.255.255.0
     no shutdown
    !
    router ospf 1
     router-id 3.3.3.3
     network 3.3.3.3 0.0.0.0 area 0
     network 10.1.13.0 0.0.0.3 area 0
     network 10.1.23.0 0.0.0.3 area 0
     network 192.168.3.0 0.0.0.255 area 0
    !

# End devices for testing
pc1:
  kind: linux
  image: alpine:latest
  exec:
    - ip addr add 192.168.1.10/24 dev eth1
```

```
        - ip route add default via 192.168.1.1

    pc2:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.2.10/24 dev eth1
        - ip route add default via 192.168.2.1

    pc3:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.3.10/24 dev eth1
        - ip route add default via 192.168.3.1

  links:
    # Router interconnections
    - endpoints: ["r1:eth1", "r2:eth1"]
    - endpoints: ["r1:eth2", "r3:eth1"]
    - endpoints: ["r2:eth2", "r3:eth2"]

    # LAN connections
    - endpoints: ["r1:eth3", "pc1:eth1"]
    - endpoints: ["r2:eth3", "pc2:eth1"]
    - endpoints: ["r3:eth3", "pc3:eth1"]
```

Basic OSPF Configuration Commands

Enabling OSPF

```
# Enable OSPF process
router ospf <process-id>
 router-id <router-id>
 network <network> <wildcard-mask> area <area-id>

# Example
router ospf 1
 router-id 1.1.1.1
 network 10.1.1.0 0.0.0.255 area 0
 network 192.168.1.0 0.0.0.255 area 0
```

Interface-Specific Configuration

```
# Configure OSPF on specific interface
interface GigabitEthernet0/0/0
 ip ospf 1 area 0
 ip ospf cost 100
 ip ospf hello-interval 10
 ip ospf dead-interval 40
```

OSPF Verification Commands

```
# Deploy and test the lab
containerlab deploy -t ospf-single-area.yml

# Connect to R1 and verify OSPF
docker exec -it clab-ospf-r1 cli

# Check OSPF neighbors
show ip ospf neighbor

# View OSPF database
show ip ospf database

# Check OSPF interfaces
show ip ospf interface

# Verify routing table
show ip route ospf

# Test connectivity
ping 2.2.2.2
ping 3.3.3.3
traceroute 192.168.3.10
```

OSPF Neighbor Relationships

Neighbor States

OSPF routers go through several states when forming neighbor relationships:

1. **Down**: No Hello packets received
2. **Init**: Hello packet received
3. **2-Way**: Bidirectional communication established
4. **ExStart**: Master/slave relationship established

5. **Exchange**: Database description packets exchanged
6. **Loading**: Link state requests sent
7. **Full**: Databases synchronized

Hello Protocol

OSPF uses Hello packets for neighbor discovery and maintenance.

Hello Packet Contents

- Router ID
- Area ID
- Network mask
- Hello interval
- Dead interval
- Designated Router (DR)
- Backup Designated Router (BDR)
- Neighbor list

Hello Timers

```
# Default timers
# Broadcast/Point-to-Point: Hello 10s, Dead 40s
# NBMA: Hello 30s, Dead 120s

# Modify timers
interface GigabitEthernet0/0/0
 ip ospf hello-interval 5
 ip ospf dead-interval 20
```

Designated Router (DR) Election

On multi-access networks, OSPF elects a DR and BDR to reduce LSA flooding.

DR Election Process

1. **Priority**: Highest OSPF priority wins (0-255)
2. **Router ID**: Highest Router ID if priority ties
3. **Preemption**: No preemption (first elected stays)

```
# Configure OSPF priority
interface GigabitEthernet0/0/0
 ip ospf priority 100
```

```
# Disable DR election (point-to-point)
interface GigabitEthernet0/0/0
 ip ospf network point-to-point
```

Multi-Area OSPF

Multi-Area OSPF Benefits

- **Reduced SPF calculations**: Changes in one area don't affect others
- **Smaller routing tables**: Route summarization at area borders
- **Faster convergence**: Localized flooding
- **Better scalability**: Hierarchical design

Multi-Area OSPF Lab

```
# Multi-area OSPF topology
name: ospf-multi-area
prefix: ospf-ma

topology:
  nodes:
    # Area 0 (Backbone) routers
    r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname ABR-R1
        !
        interface Loopback0
         ip address 1.1.1.1 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description Backbone-to-R2
         ip address 10.0.12.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description Area1-to-R3
         ip address 10.1.13.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description Area1-LAN
         ip address 192.168.1.1 255.255.255.0
         no shutdown
        !
        router ospf 1
         router-id 1.1.1.1
         network 1.1.1.1 0.0.0.0 area 0
         network 10.0.12.0 0.0.0.3 area 0
         network 10.1.13.0 0.0.0.3 area 1
         network 192.168.1.0 0.0.0.255 area 1
         area 1 range 192.168.0.0 255.255.252.0
        !

    r2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname ABR-R2
        !
        interface Loopback0
         ip address 2.2.2.2 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description Backbone-to-R1
         ip address 10.0.12.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description Area2-to-R4
         ip address 10.2.24.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description Area2-LAN
         ip address 192.168.4.1 255.255.255.0
         no shutdown
        !
        router ospf 1
         router-id 2.2.2.2
         network 2.2.2.2 0.0.0.0 area 0
         network 10.0.12.0 0.0.0.3 area 0
         network 10.2.24.0 0.0.0.3 area 2
         network 192.168.4.0 0.0.0.255 area 2
         area 2 range 192.168.4.0 255.255.252.0
        !
```

```
# Area 1 router
r3:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.12
  startup-config: |
    hostname Area1-R3
    !
    interface Loopback0
     ip address 3.3.3.3 255.255.255.255
    !
    interface GigabitEthernet0/0/0
     description To-ABR-R1
     ip address 10.1.13.2 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description Area1-LAN
     ip address 192.168.2.1 255.255.255.0
     no shutdown
    !
    router ospf 1
     router-id 3.3.3.3
     network 3.3.3.3 0.0.0.0 area 1
     network 10.1.13.0 0.0.0.3 area 1
     network 192.168.2.0 0.0.0.255 area 1
    !

# Area 2 router
r4:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.13
  startup-config: |
    hostname Area2-R4
    !
    interface Loopback0
     ip address 4.4.4.4 255.255.255.255
    !
    interface GigabitEthernet0/0/0
     description To-ABR-R2
     ip address 10.2.24.1 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description Area2-LAN
     ip address 192.168.5.1 255.255.255.0
```

```
       no shutdown
      !
      router ospf 1
       router-id 4.4.4.4
       network 4.4.4.4 0.0.0.0 area 2
       network 10.2.24.0 0.0.0.3 area 2
       network 192.168.5.0 0.0.0.255 area 2
      !

  # Test devices
  pc1:
    kind: linux
    image: alpine:latest
    exec:
      - ip addr add 192.168.1.10/24 dev eth1
      - ip route add default via 192.168.1.1

  pc2:
    kind: linux
    image: alpine:latest
    exec:
      - ip addr add 192.168.2.10/24 dev eth1
      - ip route add default via 192.168.2.1

  pc4:
    kind: linux
    image: alpine:latest
    exec:
      - ip addr add 192.168.4.10/24 dev eth1
      - ip route add default via 192.168.4.1

  pc5:
    kind: linux
    image: alpine:latest
    exec:
      - ip addr add 192.168.5.10/24 dev eth1
      - ip route add default via 192.168.5.1

links:
  # Backbone area connections
  - endpoints: ["r1:eth1", "r2:eth1"]

  # Area border connections
  - endpoints: ["r1:eth2", "r3:eth1"]
  - endpoints: ["r2:eth2", "r4:eth1"]

  # LAN connections
```

```
    - endpoints: ["r1:eth3", "pc1:eth1"]
    - endpoints: ["r3:eth2", "pc2:eth1"]
    - endpoints: ["r2:eth3", "pc4:eth1"]
    - endpoints: ["r4:eth2", "pc5:eth1"]
```

Area Border Router (ABR) Configuration

ABRs connect different OSPF areas and perform route summarization.

```
# Configure area summarization
router ospf 1
 area 1 range 192.168.0.0 255.255.252.0
 area 2 range 192.168.4.0 255.255.252.0

# Verify ABR status
show ip ospf
show ip ospf border-routers
```

OSPF Authentication

Authentication Types

Plain Text Authentication

```
# Area-wide authentication
router ospf 1
 area 0 authentication

# Interface authentication
interface GigabitEthernet0/0/0
 ip ospf authentication-key cisco123
```

MD5 Authentication (Recommended)

```
# Area-wide MD5 authentication
router ospf 1
 area 0 authentication message-digest

# Interface MD5 key
interface GigabitEthernet0/0/0
 ip ospf message-digest-key 1 md5 SecureKey123
```

Authentication Lab Example

```
# OSPF with MD5 authentication
startup-config: |
  router ospf 1
   router-id 1.1.1.1
   area 0 authentication message-digest
   network 10.1.1.0 0.0.0.255 area 0
  !
  interface GigabitEthernet0/0/0
   ip address 10.1.1.1 255.255.255.0
   ip ospf message-digest-key 1 md5 MySecretKey
   no shutdown
  !
```

OSPF Route Types and LSAs

OSPF Route Types

Code	Type	Description
O	Intra-area	Routes within the same area
O IA	Inter-area	Routes from other areas
O E1	External Type 1	External routes with internal cost
O E2	External Type 2	External routes with external cost only
O N1	NSSA External Type 1	NSSA external with internal cost
O N2	NSSA External Type 2	NSSA external with external cost

Link State Advertisements (LSAs)

LSA Types

Type	Name	Generated By	Purpose
1	Router LSA	All routers	Describe router's links
2	Network LSA	DR	Describe multi-access network
3	Summary LSA	ABR	Advertise networks between areas
4	ASBR Summary LSA	ABR	Advertise ASBR location
5	External LSA	ASBR	Advertise external routes
7	NSSA External LSA	ASBR in NSSA	External routes in NSSA

```
# View LSA database
show ip ospf database
show ip ospf database router
show ip ospf database network
show ip ospf database summary
show ip ospf database external
```

OSPF Metrics and Path Selection

OSPF Cost Calculation

OSPF uses cost as its metric, calculated as: **Cost = Reference Bandwidth / Interface Bandwidth**

Default reference bandwidth: 100 Mbps

```
# Modify reference bandwidth
router ospf 1
 auto-cost reference-bandwidth 10000  # 10 Gbps

# Set interface cost manually
interface GigabitEthernet0/0/0
 ip ospf cost 50

# View interface costs
show ip ospf interface
```

Load Balancing

OSPF supports equal-cost load balancing across multiple paths.

```
# Configure maximum paths (default is 4)
router ospf 1
 maximum-paths 6

# Verify load balancing
show ip route 192.168.1.0
show ip cef 192.168.1.0
```

OSPF Troubleshooting

Common OSPF Issues

Neighbor Adjacency Problems

```
# Symptoms
- Neighbors not forming
- Stuck in ExStart/Exchange state
- Frequent neighbor flapping

# Diagnosis
show ip ospf neighbor
show ip ospf interface
debug ip ospf hello
debug ip ospf adj

# Common causes and solutions
# 1. Hello/Dead timer mismatch
interface GigabitEthernet0/0/0
 ip ospf hello-interval 10
 ip ospf dead-interval 40

# 2. Area mismatch
router ospf 1
 network 10.1.1.0 0.0.0.255 area 0

# 3. Authentication mismatch
interface GigabitEthernet0/0/0
 ip ospf message-digest-key 1 md5 CorrectKey

# 4. MTU mismatch
interface GigabitEthernet0/0/0
 ip mtu 1500
 ip ospf mtu-ignore
```

Routing Table Issues

```
# Missing routes
show ip ospf database
show ip route ospf
show ip ospf border-routers

# Suboptimal routing
show ip ospf interface | include Cost
show ip route 192.168.1.0 longer-prefixes
```

LSA Database Problems

```
# Database synchronization issues
show ip ospf database
show ip ospf statistics
clear ip ospf process

# LSA aging and refresh
show ip ospf database | include Age
show ip ospf database self-originate
```

Diagnostic Commands

Essential OSPF Show Commands

```
# Neighbor information
show ip ospf neighbor
show ip ospf neighbor detail

# Interface information
show ip ospf interface
show ip ospf interface brief

# Database information
show ip ospf database
show ip ospf database router
show ip ospf database summary

# Process information
show ip ospf
show ip protocols
show ip route ospf
```

Advanced Troubleshooting

```
# Debug commands (use carefully)
debug ip ospf hello
debug ip ospf adj
debug ip ospf spf
debug ip ospf lsa-generation
```

```
# Statistics and monitoring
show ip ospf statistics
show ip ospf flood-list
show ip ospf request-list
show ip ospf retransmission-list
```

OSPF Optimization

Convergence Optimization

SPF Throttling

```
# Configure SPF timers
router ospf 1
 timers throttle spf 5 50 5000
 # Initial delay: 5ms
 # Minimum hold time: 50ms
 # Maximum hold time: 5000ms
```

LSA Throttling

```
# Configure LSA generation throttling
router ospf 1
 timers throttle lsa 5 50 5000
```

Hello Interval Tuning

```
# Faster convergence with shorter timers
interface GigabitEthernet0/0/0
 ip ospf hello-interval 1
 ip ospf dead-interval 3
```

Memory and CPU Optimization

Area Design

```
# Implement proper area hierarchy
# Keep areas small (< 50 routers)
# Use area summarization
router ospf 1
```

```
 area 1 range 192.168.0.0 255.255.252.0
```

LSA Filtering

```
# Filter LSAs at area borders
router ospf 1
 area 1 filter-list prefix AREA1-FILTER in
 area 1 filter-list prefix AREA1-FILTER out
```

Advanced OSPF Features

Stub Areas

Stub Area Configuration

```
# Configure stub area
router ospf 1
 area 1 stub
 network 10.1.1.0 0.0.0.255 area 1

# Totally stubby area (Cisco proprietary)
router ospf 1
 area 1 stub no-summary
```

NSSA Configuration

```
# Configure NSSA
router ospf 1
 area 1 nssa
 redistribute static subnets

# NSSA totally stubby
router ospf 1
 area 1 nssa no-summary
```

Virtual Links

Virtual links connect areas to the backbone through a transit area.

```
# Configure virtual link
router ospf 1
 area 1 virtual-link 2.2.2.2
 # 2.2.2.2 is the router ID of the other end
```

Route Filtering and Manipulation

```
# Filter routes with distribute lists
router ospf 1
 distribute-list 10 out
 distribute-list prefix OSPF-FILTER in

# Modify route attributes
route-map OSPF-METRIC permit 10
 set metric 100
 set metric-type type-1

router ospf 1
 redistribute static route-map OSPF-METRIC
```

OSPF Best Practices

Design Guidelines

1. **Hierarchical Design**: Use proper area structure
2. **Area Size**: Keep areas manageable (< 50 routers)
3. **Backbone Connectivity**: All areas must connect to Area 0
4. **Route Summarization**: Implement at area borders
5. **Authentication**: Use MD5 authentication

Configuration Best Practices

1. **Router ID**: Use loopback interfaces for stability
2. **Reference Bandwidth**: Adjust for high-speed links
3. **Timers**: Tune for convergence requirements
4. **Passive Interfaces**: Secure unnecessary OSPF interfaces
5. **Area Types**: Use stub areas where appropriate

```
# Best practice configuration template
router ospf 1
 router-id 1.1.1.1
 auto-cost reference-bandwidth 10000
 passive-interface default
```

```
no passive-interface GigabitEthernet0/0/0
area 0 authentication message-digest
area 1 stub
area 1 range 192.168.0.0 255.255.252.0
timers throttle spf 5 50 5000
```

Summary

OSPF is a robust and scalable routing protocol essential for enterprise networks. Understanding its operation, configuration, and troubleshooting is crucial for network engineers. Proper OSPF design with appropriate area structure, authentication, and optimization ensures reliable and efficient routing.

Key concepts covered: - OSPF fundamentals and operation - Single-area and multi-area configuration - Neighbor relationships and DR election - Authentication and security - Route types and LSAs - Troubleshooting methodologies - Performance optimization techniques

In the next chapter, we'll explore EIGRP, Cisco's proprietary routing protocol with unique features and capabilities.

Review Questions

1. What are the advantages of OSPF over distance vector protocols?
2. How does the DR/BDR election process work?
3. What are the benefits of implementing multi-area OSPF?
4. How do you troubleshoot OSPF neighbor adjacency issues?
5. What are the different OSPF area types and their use cases?

Hands-on Exercises

Exercise 1: Single-Area OSPF

1. Deploy the single-area OSPF lab
2. Configure OSPF on all routers
3. Verify neighbor relationships and routing tables
4. Test connectivity and path selection

Exercise 2: Multi-Area OSPF

1. Implement the multi-area OSPF topology
2. Configure ABRs with route summarization
3. Verify inter-area routing
4. Test area isolation and summarization

Exercise 3: OSPF Authentication

1. Configure MD5 authentication on OSPF areas
2. Test authentication failures and recovery
3. Implement different authentication keys
4. Verify security improvements

Exercise 4: OSPF Troubleshooting

1. Create various OSPF problems (neighbor issues, LSA problems)
2. Practice diagnostic commands and procedures
3. Develop systematic troubleshooting approaches
4. Document solutions and prevention strategies

Additional Resources

- OSPF RFC 2328
- Cisco OSPF Configuration Guide
- OSPF Design Guide
- OSPF Troubleshooting Guide

Chapter 19: Network Security Fundamentals

Learning Objectives

By the end of this chapter, you will be able to: - Understand fundamental network security concepts and threats - Implement defense-in-depth security strategies - Configure basic security features in ContainerLab environments - Apply security best practices to network infrastructure - Recognize and mitigate common network attacks

Network Security Overview

What is Network Security?

Network security encompasses the policies, procedures, and technologies designed to protect network infrastructure, data, and resources from unauthorized access, misuse, modification, or destruction. It involves multiple layers of defense to create a comprehensive security posture.

Core Security Principles

CIA Triad: - **Confidentiality**: Ensuring information is accessible only to authorized users - **Integrity**: Maintaining data accuracy and preventing unauthorized modification - **Availability**: Ensuring systems and data are accessible when needed

Additional Principles: - **Authentication**: Verifying user and device identities - **Authorization**: Controlling access to resources - **Accounting**: Tracking and logging security events - **Non-repudiation**: Preventing denial of actions

Common Network Threats

External Threats

1. **Malware**: Viruses, worms, trojans, ransomware
2. **DDoS Attacks**: Distributed denial of service
3. **Man-in-the-Middle**: Intercepting communications
4. **Social Engineering**: Manipulating users for information
5. **Advanced Persistent Threats (APTs)**: Long-term targeted attacks

Internal Threats

1. **Insider Threats**: Malicious or negligent employees
2. **Privilege Escalation**: Unauthorized access elevation
3. **Data Exfiltration**: Unauthorized data removal
4. **Misconfigurations**: Accidental security weaknesses
5. **Shadow IT**: Unauthorized technology usage

Defense-in-Depth Strategy

Defense-in-depth implements multiple security layers to protect against various threats.

Security Layers

1. **Physical Security**: Securing physical access to infrastructure
2. **Perimeter Security**: Firewalls, IPS, and network segmentation
3. **Network Security**: VLANs, ACLs, and network monitoring
4. **Host Security**: Endpoint protection and hardening
5. **Application Security**: Secure coding and application firewalls
6. **Data Security**: Encryption and data loss prevention

Network Security Lab Environment

Basic Security Lab Setup

```
# Network security demonstration lab
name: network-security-lab
prefix: sec

topology:
  nodes:
    # Internet-facing router (DMZ)
    edge-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Edge-Router
        !
        ! External interface (simulated internet)
        interface GigabitEthernet0/0/0
         description Internet-Facing
         ip address 203.0.113.1 255.255.255.252
         no shutdown
        !
        ! DMZ interface
        interface GigabitEthernet0/0/1
         description DMZ-Network
         ip address 192.168.100.1 255.255.255.0
         no shutdown
        !
        ! Internal interface
        interface GigabitEthernet0/0/2
         description Internal-Network
         ip address 10.1.1.1 255.255.255.252
         no shutdown
        !
        ! Basic security configuration
        no ip source-route
        no ip gratuitous-arps
        ip cef
        !
        ! Access control lists (will be configured later)
        !

    # Internal firewall/router
    internal-fw:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Internal-Firewall
        !
        interface GigabitEthernet0/0/0
         description To-Edge-Router
         ip address 10.1.1.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description Internal-LAN
         ip address 192.168.10.1 255.255.255.0
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description Server-VLAN
         ip address 192.168.20.1 255.255.255.0
         no shutdown
        !
        ! Security hardening
        no ip source-route
        no ip gratuitous-arps
```

```
        service password-encryption
        !

    # DMZ web server
    dmz-server:
      kind: linux
      image: nginx:alpine
      mgmt-ipv4: 172.20.20.20
      exec:
        - ip addr add 192.168.100.10/24 dev eth1
        - ip route add default via 192.168.100.1
        - nginx -g "daemon off;" &

    # Internal workstation
    internal-pc:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.21
      exec:
        - ip addr add 192.168.10.10/24 dev eth1
        - ip route add default via 192.168.10.1
        - apk add --no-cache curl nmap

    # Internal server
    internal-server:
      kind: linux
      image: ubuntu:20.04
      mgmt-ipv4: 172.20.20.22
      exec:
        - ip addr add 192.168.20.10/24 dev eth1
        - ip route add default via 192.168.20.1
        - apt update && apt install -y openssh-server
        - service ssh start

    # Simulated internet/attacker
    internet-sim:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.100
      exec:
        - ip addr add 203.0.113.2/30 dev eth1
        - ip route add default via 203.0.113.1
        - apk add --no-cache nmap hping3 curl

  links:
    # Network connections
    - endpoints: ["edge-router:eth1", "internet-sim:eth1"]
    - endpoints: ["edge-router:eth2", "dmz-server:eth1"]
    - endpoints: ["edge-router:eth3", "internal-fw:eth1"]
    - endpoints: ["internal-fw:eth2", "internal-pc:eth1"]
    - endpoints: ["internal-fw:eth3", "internal-server:eth1"]
```

Access Control Lists (ACLs)

ACL Fundamentals

Access Control Lists are packet filters that permit or deny traffic based on various criteria such as source/destination IP addresses, ports, and protocols.

ACL Types

Standard ACLs (1-99, 1300-1999): - Filter based on source IP address only - Applied close to destination

Extended ACLs (100-199, 2000-2699): - Filter based on source/destination IP, ports, protocols - Applied close to source

Named ACLs: - Use descriptive names instead of numbers - Allow modification of individual entries

Basic ACL Configuration

Standard ACL Example

```
# Deploy the security lab
containerlab deploy -t network-security-lab.yml

# Configure standard ACL on internal firewall
docker exec -it clab-sec-internal-fw cli

configure terminal
! Standard ACL to block specific source
access-list 10 deny 192.168.10.50 0.0.0.0
access-list 10 permit 192.168.10.0 0.0.0.255
access-list 10 deny any

! Apply to interface
interface GigabitEthernet0/0/1
 ip access-group 10 in

! Verify ACL
```

```
show access-lists
show ip interface GigabitEthernet0/0/1
```

Extended ACL Example

```
# Extended ACL for more granular control
configure terminal
! Block HTTP traffic from internal network to internet
access-list 101 deny tcp 192.168.10.0 0.0.0.255 any eq 80
access-list 101 deny tcp 192.168.10.0 0.0.0.255 any eq 443
access-list 101 permit ip 192.168.10.0 0.0.0.255 any

! Apply to interface
interface GigabitEthernet0/0/0
 ip access-group 101 out
```

Named ACL Example

```
# Named ACL for better management
configure terminal
ip access-list extended INTERNET-FILTER
 deny tcp 192.168.10.0 0.0.0.255 any eq 80 log
 deny tcp 192.168.10.0 0.0.0.255 any eq 443 log
 permit tcp 192.168.10.0 0.0.0.255 any eq 22
 permit tcp 192.168.10.0 0.0.0.255 any eq 53
 permit udp 192.168.10.0 0.0.0.255 any eq 53
 deny ip any any log

! Apply to interface
interface GigabitEthernet0/0/0
 ip access-group INTERNET-FILTER out
```

Advanced ACL Features

Time-Based ACLs

```
# Configure time range
time-range BUSINESS-HOURS
 periodic weekdays 8:00 to 17:00

# Apply to ACL
ip access-list extended TIME-BASED-FILTER
```

```
 permit tcp 192.168.10.0 0.0.0.255 any eq 80 time-range BUSINESS-HOURS
 deny tcp 192.168.10.0 0.0.0.255 any eq 80
 permit ip any any
```

Reflexive ACLs

```
# Configure reflexive ACL
ip access-list extended OUTBOUND
 permit tcp 192.168.10.0 0.0.0.255 any reflect TCP-TRAFFIC
 permit udp 192.168.10.0 0.0.0.255 any reflect UDP-TRAFFIC

ip access-list extended INBOUND
 evaluate TCP-TRAFFIC
 evaluate UDP-TRAFFIC
 deny ip any any log

! Apply to interfaces
interface GigabitEthernet0/0/0
 ip access-group OUTBOUND out
 ip access-group INBOUND in
```

Network Address Translation (NAT) Security

NAT as Security Feature

NAT provides security benefits by hiding internal network structure and preventing direct external access to internal hosts.

Basic NAT Configuration

```
# Configure NAT on edge router
configure terminal
! Define inside and outside interfaces
interface GigabitEthernet0/0/0
 ip nat outside

interface GigabitEthernet0/0/2
 ip nat inside

! Configure NAT pool and access list
access-list 1 permit 192.168.10.0 0.0.0.255
ip nat pool INTERNET-POOL 203.0.113.10 203.0.113.20 netmask 255.255.255.240
ip nat inside source list 1 pool INTERNET-POOL overload

! Static NAT for DMZ server
ip nat inside source static 192.168.100.10 203.0.113.5
```

NAT Security Considerations

```
# Monitor NAT translations
show ip nat translations
show ip nat statistics

# Clear NAT translations for security
clear ip nat translation *
clear ip nat translation inside 192.168.10.10
```

Device Hardening

Router and Switch Security

Basic Device Hardening

```
# Security hardening configuration
startup-config: |
  ! Disable unnecessary services
  no ip source-route
  no ip gratuitous-arps
  no ip redirects
  no ip proxy-arp
  no ip unreachables
  no ip mask-reply
  no service pad
  no service finger
  no service udp-small-servers
  no service tcp-small-servers

  ! Enable security features
  service password-encryption
  service timestamps debug datetime msec
  service timestamps log datetime msec

  ! Secure console and VTY lines
  line console 0
   exec-timeout 5 0
   logging synchronous
   login local

  line vty 0 15
   exec-timeout 5 0
   logging synchronous
   login local
   transport input ssh

  ! Configure strong passwords
  enable secret $1$mERr$hx5rVt7rPNoS4wqbXKX7m0
  username admin privilege 15 secret $1$mERr$hx5rVt7rPNoS4wqbXKX7m0

  ! SSH configuration
  ip domain-name secure.lab
  crypto key generate rsa modulus 2048
  ip ssh version 2
  ip ssh time-out 60
  ip ssh authentication-retries 3
```

Advanced Security Features

```
# Configure login security
security authentication failure rate 3 log
security passwords min-length 8

# Enable TCP intercept for SYN flood protection
ip tcp intercept list 100
ip tcp intercept mode intercept
ip tcp intercept max-incomplete low 450
ip tcp intercept max-incomplete high 550

access-list 100 permit tcp any 192.168.100.0 0.0.0.255
```

SNMP Security

```
# Secure SNMP configuration
! Remove default community strings
no snmp-server community public
no snmp-server community private

! Configure SNMPv3 with authentication and encryption
snmp-server group SECURE-GROUP v3 priv
snmp-server user secure-user SECURE-GROUP v3 auth sha AuthPass123 priv aes 128 PrivPass123
```

```
snmp-server view SECURE-VIEW system included
snmp-server view SECURE-VIEW interfaces included
snmp-server group SECURE-GROUP v3 priv read SECURE-VIEW

! Restrict SNMP access
access-list 50 permit 192.168.10.100
snmp-server community SecureComm123 RO 50
```

Network Monitoring and Logging

Logging Configuration

```
# Configure comprehensive logging
logging buffered 16384 informational
logging console critical
logging monitor informational
logging trap informational
logging facility local0
logging source-interface Loopback0
logging 192.168.10.100

! Log security events
login on-failure log
login on-success log
security authentication failure rate 3 log

! Archive logs
archive
 log config
  logging enable
  logging size 100
  notify syslog contenttype plaintext
```

Network Time Protocol (NTP) Security

```
# Secure NTP configuration
ntp authenticate
ntp authentication-key 1 md5 NTPSecretKey123
ntp trusted-key 1
ntp server 192.168.10.100 key 1
ntp access-group serve-only 10

access-list 10 permit 192.168.10.0 0.0.0.255
```

Intrusion Detection and Prevention

Basic IDS/IPS Concepts

Detection Methods

1. **Signature-based**: Matches known attack patterns
2. **Anomaly-based**: Detects deviations from normal behavior
3. **Hybrid**: Combines both approaches

Deployment Models

1. **Network-based (NIDS/NIPS)**: Monitors network traffic
2. **Host-based (HIDS/HIPS)**: Monitors individual hosts
3. **Hybrid**: Combines network and host-based monitoring

Cisco IOS IPS Configuration

```
# Configure IOS IPS
ip ips config location flash:ips/
ip ips name IPS-POLICY

! Create IPS rule
ip ips signature-category
 category all
  retired true
 category ios_ips basic
  retired false

! Apply to interface
interface GigabitEthernet0/0/0
 ip ips IPS-POLICY in
```

Wireless Security

Wireless Security Protocols

WEP (Wired Equivalent Privacy)

- **Status**: Deprecated, insecure
- **Key Length**: 40-bit or 104-bit
- **Vulnerabilities**: Weak encryption, easily cracked

WPA (Wi-Fi Protected Access)

- **Improvement**: Over WEP
- **Encryption**: TKIP
- **Authentication**: PSK or 802.1X

WPA2 (Wi-Fi Protected Access 2)

- **Encryption**: AES-CCMP
- **Authentication**: PSK or 802.1X
- **Status**: Current standard

WPA3 (Wi-Fi Protected Access 3)

- **Improvements**: Enhanced security
- **Features**: SAE, enhanced open, 192-bit security
- **Status**: Latest standard

Wireless Security Best Practices

```
# Wireless security configuration example
! Change default SSID and disable broadcast
dot11 ssid SECURE-CORPORATE
 authentication open
 authentication key-management wpa version 2
 wpa-psk ascii SecureWirelessKey123!
 no guest-mode

! Enable strong encryption
encryption mode ciphers aes-ccmp

! MAC address filtering
dot11 association mac-list 610

! Disable unnecessary features
no dot11 extension aironet
no dot11 beacon dtim-period
```

VPN Security

VPN Types

Site-to-Site VPN

- Connects entire networks
- Typically uses IPSec
- Permanent connections

Remote Access VPN

- Connects individual users
- Uses SSL/TLS or IPSec
- On-demand connections

Basic IPSec Configuration

```
# IPSec site-to-site VPN configuration
crypto isakmp policy 10
 encryption aes 256
 hash sha256
 authentication pre-share
 group 14
 lifetime 28800

crypto isakmp key SecretPresharedKey123 address 203.0.113.100

crypto ipsec transform-set STRONG-SET esp-aes 256 esp-sha256-hmac
 mode tunnel

crypto map VPN-MAP 10 ipsec-isakmp
 set peer 203.0.113.100
 set transform-set STRONG-SET
 match address 110

access-list 110 permit ip 192.168.10.0 0.0.0.255 192.168.20.0 0.0.0.255

interface GigabitEthernet0/0/0
 crypto map VPN-MAP
```

Security Testing and Validation

Penetration Testing Lab

```
# Test network security from internet simulator
docker exec -it clab-sec-internet-sim sh

# Network reconnaissance
nmap -sS 203.0.113.1
nmap -sU 203.0.113.1

# Port scanning
nmap -p 1-1000 203.0.113.1

# Service enumeration
nmap -sV 203.0.113.1

# Test for common vulnerabilities
nmap --script vuln 203.0.113.1
```

Security Monitoring

```
# Monitor security events on routers
show logging | include SECURITY
show ip access-lists
show ip nat translations
show crypto isakmp sa
show crypto ipsec sa

# Real-time monitoring
debug ip packet 101 detail
debug crypto isakmp
debug crypto ipsec
```

Incident Response

Security Incident Handling

Incident Response Process

1. **Preparation**: Establish procedures and tools
2. **Identification**: Detect and analyze incidents
3. **Containment**: Limit damage and prevent spread

4. **Eradication**: Remove threats from environment
5. **Recovery**: Restore normal operations
6. **Lessons Learned**: Improve security posture

Network Isolation Procedures

```
# Emergency network isolation
! Shutdown compromised interfaces
interface GigabitEthernet0/0/1
 shutdown

! Block malicious traffic
access-list 199 deny ip host 192.168.10.50 any
access-list 199 permit ip any any

interface GigabitEthernet0/0/0
 ip access-group 199 in

! Clear NAT translations
clear ip nat translation *

! Reset connections
clear tcp local 192.168.10.50 foreign any
```

Security Best Practices

Network Design Security

1. **Network Segmentation**: Use VLANs and subnets
2. **Least Privilege**: Minimal necessary access
3. **Defense in Depth**: Multiple security layers
4. **Regular Updates**: Keep systems patched
5. **Monitoring**: Continuous security monitoring

Configuration Management

1. **Change Control**: Document all changes
2. **Configuration Backup**: Regular backups
3. **Version Control**: Track configuration versions
4. **Compliance**: Follow security standards
5. **Auditing**: Regular security audits

Security Policies

```
# Security policy template
! Password policy
security passwords min-length 8
enable secret $1$mERr$hx5rVt7rPNoS4wqbXKX7m0

! Access control policy
access-list 1 permit 192.168.10.0 0.0.0.255
access-list 1 deny any log

! Logging policy
logging buffered 16384 informational
logging trap warnings
logging facility local0

! Time synchronization
ntp server 192.168.10.100
service timestamps log datetime msec
```

Summary

Network security is a critical aspect of modern networking that requires a comprehensive, multi-layered approach. Understanding fundamental security concepts, implementing proper access controls, hardening network devices, and maintaining continuous monitoring are essential for protecting network infrastructure and data.

Key concepts covered: - Network security fundamentals and threat landscape - Access control lists and traffic filtering - Device hardening and security configuration - Network monitoring and logging - Intrusion detection and prevention - Wireless and VPN security - Incident response procedures

In the next chapter, we'll explore specific switch and router security implementations and advanced security features.

Review Questions

1. What are the core principles of network security (CIA triad)?
2. How do standard and extended ACLs differ in functionality?
3. What are the key components of a defense-in-depth strategy?
4. How does NAT provide security benefits beyond address translation?
5. What are the essential steps in network device hardening?

Hands-on Exercises

Exercise 1: Basic Network Security Implementation

1. Deploy the network security lab topology
2. Configure ACLs to control traffic flow
3. Implement basic device hardening
4. Test security controls with traffic generation

Exercise 2: Advanced Access Control

1. Configure named and time-based ACLs
2. Implement reflexive ACLs for stateful filtering
3. Set up NAT with security considerations
4. Monitor and log security events

Exercise 3: Security Testing and Validation

1. Perform network reconnaissance from external host
2. Test ACL effectiveness with various traffic types
3. Simulate security incidents and response procedures
4. Document security gaps and improvements

Exercise 4: Comprehensive Security Hardening

1. Implement complete device hardening checklist
2. Configure secure remote access (SSH, SNMP)
3. Set up centralized logging and monitoring
4. Create security policies and procedures

Additional Resources

- NIST Cybersecurity Framework
- Cisco Security Configuration Guide
- Network Security Best Practices
- OWASP Network Security Testing Guide

Advanced

Chapter 29: Advanced OSPF Features

Learning Objectives

By the end of this chapter, you will be able to: - Configure and optimize advanced OSPF area types - Implement OSPF virtual links and advanced authentication - Understand and manipulate OSPF LSA types - Optimize OSPF performance for large-scale networks - Troubleshoot complex OSPF scenarios

Advanced OSPF Area Types

Stub Areas Deep Dive

Stub areas reduce the size of the OSPF database by blocking external LSAs and using a default route for external destinations.

Standard Stub Area Configuration

```
# Advanced OSPF stub area topology
name: ospf-advanced-areas
prefix: ospf-adv

topology:
  nodes:
    # Area 0 (Backbone)
    backbone-r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Backbone-R1
        !
        interface Loopback0
         ip address 1.1.1.1 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description To-Backbone-R2
         ip address 10.0.12.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description To-Stub-ABR
         ip address 10.0.13.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description To-NSSA-ABR
         ip address 10.0.14.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/3
         description To-External-ASBR
         ip address 10.0.15.1 255.255.255.252
         no shutdown
        !
        router ospf 1
         router-id 1.1.1.1
         network 1.1.1.1 0.0.0.0 area 0
         network 10.0.12.0 0.0.0.3 area 0
         network 10.0.13.0 0.0.0.3 area 0
         network 10.0.14.0 0.0.0.3 area 0
         network 10.0.15.0 0.0.0.3 area 0
         default-information originate
        !

    backbone-r2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Backbone-R2
        !
        interface Loopback0
         ip address 2.2.2.2 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description To-Backbone-R1
         ip address 10.0.12.2 255.255.255.252
         no shutdown
        !
        router ospf 1
         router-id 2.2.2.2
         network 2.2.2.2 0.0.0.0 area 0
         network 10.0.12.0 0.0.0.3 area 0
        !
```

```
# Stub Area ABR
stub-abr:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.12
  startup-config: |
    hostname Stub-ABR
    !
    interface Loopback0
     ip address 3.3.3.3 255.255.255.255
    !
    interface GigabitEthernet0/0/0
     description To-Backbone
     ip address 10.0.13.2 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description To-Stub-Internal
     ip address 10.1.34.3 255.255.255.252
     no shutdown
    !
    router ospf 1
     router-id 3.3.3.3
     network 3.3.3.3 0.0.0.0 area 0
     network 10.0.13.0 0.0.0.3 area 0
     network 10.1.34.0 0.0.0.3 area 1
     area 1 stub
     area 1 default-cost 10
    !

# Stub Area Internal Router
stub-internal:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.13
  startup-config: |
    hostname Stub-Internal
    !
    interface Loopback0
     ip address 4.4.4.4 255.255.255.255
    !
    interface GigabitEthernet0/0/0
     description To-Stub-ABR
     ip address 10.1.34.4 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description Stub-LAN
     ip address 192.168.1.1 255.255.255.0
     no shutdown
    !
    router ospf 1
     router-id 4.4.4.4
     network 4.4.4.4 0.0.0.0 area 1
     network 10.1.34.0 0.0.0.3 area 1
     network 192.168.1.0 0.0.0.255 area 1
     area 1 stub
    !

# NSSA ABR
nssa-abr:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.14
  startup-config: |
    hostname NSSA-ABR
    !
    interface Loopback0
     ip address 5.5.5.5 255.255.255.255
    !
    interface GigabitEthernet0/0/0
     description To-Backbone
     ip address 10.0.14.2 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description To-NSSA-ASBR
     ip address 10.2.56.5 255.255.255.252
     no shutdown
    !
    router ospf 1
     router-id 5.5.5.5
     network 5.5.5.5 0.0.0.0 area 0
     network 10.0.14.0 0.0.0.3 area 0
     network 10.2.56.0 0.0.0.3 area 2
     area 2 nssa default-information-originate
    !

# NSSA ASBR
nssa-asbr:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
```

```
    mgmt-ipv4: 172.20.20.15
    startup-config: |
      hostname NSSA-ASBR
      !
      interface Loopback0
       ip address 6.6.6.6 255.255.255.255
      !
      interface GigabitEthernet0/0/0
       description To-NSSA-ABR
       ip address 10.2.56.6 255.255.255.252
       no shutdown
      !
      interface GigabitEthernet0/0/1
       description External-Network
       ip address 172.16.1.1 255.255.255.0
       no shutdown
      !
      router ospf 1
       router-id 6.6.6.6
       network 6.6.6.6 0.0.0.0 area 2
       network 10.2.56.0 0.0.0.3 area 2
       area 2 nssa
       redistribute connected subnets
      !

  # External ASBR
  external-asbr:
    kind: cisco_iosxe
    image: cisco/iosxe:latest
    mgmt-ipv4: 172.20.20.16
    startup-config: |
      hostname External-ASBR
      !
      interface Loopback0
       ip address 7.7.7.7 255.255.255.255
      !
      interface GigabitEthernet0/0/0
       description To-Backbone
       ip address 10.0.15.2 255.255.255.252
       no shutdown
      !
      interface GigabitEthernet0/0/1
       description External-Network
       ip address 203.0.113.1 255.255.255.0
       no shutdown
      !
      router ospf 1
```

```
          router-id 7.7.7.7
          network 7.7.7.7 0.0.0.0 area 0
          network 10.0.15.0 0.0.0.3 area 0
          redistribute connected subnets
         !

    # Test devices
    stub-pc:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.1.10/24 dev eth1
        - ip route add default via 192.168.1.1

    external-server:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 172.16.1.10/24 dev eth1
        - ip route add default via 172.16.1.1

  links:
    # Backbone connections
    - endpoints: ["backbone-r1:eth1", "backbone-r2:eth1"]
    - endpoints: ["backbone-r1:eth2", "stub-abr:eth1"]
    - endpoints: ["backbone-r1:eth3", "nssa-abr:eth1"]
    - endpoints: ["backbone-r1:eth4", "external-asbr:eth1"]

    # Area connections
    - endpoints: ["stub-abr:eth2", "stub-internal:eth1"]
    - endpoints: ["nssa-abr:eth2", "nssa-asbr:eth1"]

    # End device connections
    - endpoints: ["stub-internal:eth2", "stub-pc:eth1"]
    - endpoints: ["nssa-asbr:eth2", "external-server:eth1"]
```

Totally Stubby Areas

Totally stubby areas (Cisco proprietary) block both external and inter-area LSAs.

```
# Configure totally stubby area
router ospf 1
 area 1 stub no-summary

# Verify totally stubby area
show ip ospf database
show ip route ospf
```

Not-So-Stubby Areas (NSSA)

NSSA allows limited external route advertisement within the area using Type-7 LSAs.

```
# NSSA configuration options
router ospf 1
 area 2 nssa
 area 2 nssa default-information-originate
 area 2 nssa no-redistribution
 area 2 nssa no-summary

# NSSA translator election
router ospf 1
 area 2 nssa translate type7 suppress-fa
```

Advanced LSA Types and Database Optimization

Understanding LSA Types in Detail

Type-1 Router LSA

```
# View detailed Router LSA
show ip ospf database router 1.1.1.1

# Router LSA contains:
# - Router ID and area
# - Link types and costs
# - Router capabilities
```

Type-2 Network LSA

```
# View Network LSA (generated by DR)
show ip ospf database network

# Network LSA contains:
# - DR router ID
# - Network mask
# - Attached routers
```

Type-3 Summary LSA

```
# View Summary LSAs
show ip ospf database summary

# Control summary LSA generation
router ospf 1
 area 1 range 192.168.0.0 255.255.252.0
 no area 1 range 192.168.4.0 255.255.255.0
```

Type-4 ASBR Summary LSA

```
# View ASBR Summary LSAs
show ip ospf database asbr-summary

# These LSAs advertise the location of ASBRs
```

Type-5 External LSA

```
# View External LSAs
show ip ospf database external

# Control external route advertisement
router ospf 1
 redistribute connected subnets route-map EXTERNAL-FILTER
 summary-address 172.16.0.0 255.255.0.0
```

Type-7 NSSA External LSA

```
# View NSSA External LSAs
show ip ospf database nssa-external

# Type-7 to Type-5 translation
router ospf 1
 area 2 nssa translate type7 always
```

Database Optimization Techniques

LSA Filtering

```
# Filter LSAs at area borders
router ospf 1
 area 1 filter-list prefix AREA1-IN in
 area 1 filter-list prefix AREA1-OUT out

ip prefix-list AREA1-IN seq 10 deny 192.168.100.0/24
ip prefix-list AREA1-IN seq 20 permit 0.0.0.0/0 le 32
```

Summary Address Configuration

```
# Configure summary addresses
router ospf 1
 area 1 range 192.168.0.0 255.255.252.0
 summary-address 172.16.0.0 255.255.0.0 not-advertise
 summary-address 10.0.0.0 255.0.0.0 tag 100
```

Virtual Links

Virtual links connect areas to the backbone through a transit area when direct backbone connectivity is not possible.

Virtual Link Configuration

```
# Virtual link topology
name: ospf-virtual-link
topology:
  nodes:
    backbone-r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        hostname Backbone-R1
        !
        interface Loopback0
         ip address 1.1.1.1 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         ip address 10.0.12.1 255.255.255.252
         no shutdown
        !
        router ospf 1
         router-id 1.1.1.1
         network 1.1.1.1 0.0.0.0 area 0
         network 10.0.12.0 0.0.0.3 area 1
         area 1 virtual-link 3.3.3.3
        !

    transit-abr:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        hostname Transit-ABR
        !
        interface Loopback0
         ip address 2.2.2.2 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         ip address 10.0.12.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         ip address 10.1.23.2 255.255.255.252
         no shutdown
        !
        router ospf 1
         router-id 2.2.2.2
         network 2.2.2.2 0.0.0.0 area 1
         network 10.0.12.0 0.0.0.3 area 1
         network 10.1.23.0 0.0.0.3 area 1
        !

    remote-abr:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        hostname Remote-ABR
        !
        interface Loopback0
         ip address 3.3.3.3 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         ip address 10.1.23.3 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
```

```
          ip address 10.2.34.3 255.255.255.252
          no shutdown
         !
         router ospf 1
          router-id 3.3.3.3
          network 3.3.3.3 0.0.0.0 area 1
          network 10.1.23.0 0.0.0.3 area 1
          network 10.2.34.0 0.0.0.3 area 2
          area 1 virtual-link 1.1.1.1
         !

    links:
      - endpoints: ["backbone-r1:eth1", "transit-abr:eth1"]
      - endpoints: ["transit-abr:eth2", "remote-abr:eth1"]
```

Virtual Link Authentication

```
# Configure virtual link with authentication
router ospf 1
 area 1 virtual-link 3.3.3.3 authentication message-digest
 area 1 virtual-link 3.3.3.3 message-digest-key 1 md5 VirtualLinkKey123

# Verify virtual link
show ip ospf virtual-links
```

Advanced OSPF Authentication

Area-Wide Authentication

```
# Configure area-wide authentication
router ospf 1
 area 0 authentication message-digest
 area 1 authentication

# Interface authentication keys
interface GigabitEthernet0/0/0
 ip ospf message-digest-key 1 md5 AreaZeroKey123
 ip ospf message-digest-key 2 md5 NewAreaZeroKey456
```

Cryptographic Authentication

```
# Configure cryptographic authentication
router ospf 1
 area 0 authentication message-digest

interface GigabitEthernet0/0/0
 ip ospf message-digest-key 1 md5 7 encrypted-key-string
 ip ospf message-digest-key 2 md5 0 plain-text-key

# Key rollover process
interface GigabitEthernet0/0/0
 ip ospf message-digest-key 2 md5 NewSecureKey789
 no ip ospf message-digest-key 1
```

OSPF Performance Optimization

SPF Throttling

```
# Configure SPF throttling
router ospf 1
 timers throttle spf 5 50 5000
 # Initial delay: 5ms
 # Minimum hold time: 50ms
 # Maximum hold time: 5000ms

# LSA throttling
router ospf 1
 timers throttle lsa 5 50 5000

# Pacing timers
router ospf 1
 timers pacing flood 33
 timers pacing lsa-group 240
 timers pacing retransmission 66
```

Memory and CPU Optimization

```
# Limit LSA generation
router ospf 1
 max-lsa 10000 75 warning-only

# Database overflow protection
```

```
router ospf 1
 overflow database external 1000 5 10

# Incremental SPF
router ospf 1
 ispf
```

Interface Optimization

```
# Optimize hello and dead intervals
interface GigabitEthernet0/0/0
 ip ospf hello-interval 1
 ip ospf dead-interval 3

# Fast hello packets
interface GigabitEthernet0/0/0
 ip ospf dead-interval minimal hello-multiplier 4

# BFD integration
router ospf 1
 bfd all-interfaces

interface GigabitEthernet0/0/0
 bfd interval 50 min_rx 50 multiplier 3
```

Advanced OSPF Troubleshooting

Database Synchronization Issues

```
# Check database synchronization
show ip ospf database database-summary
show ip ospf statistics
show ip ospf flood-list
show ip ospf request-list
show ip ospf retransmission-list

# Force database synchronization
clear ip ospf process
clear ip ospf redistribution
```

LSA Corruption and Aging

```
# Check LSA aging and corruption
show ip ospf database | include Age
show ip ospf database adv-router 1.1.1.1
show ip ospf database self-originate

# Premature aging
router ospf 1
 no area 1 range 192.168.1.0 255.255.255.0
```

Performance Monitoring

```
# Monitor OSPF performance
show ip ospf statistics detail
show ip ospf timers
show ip ospf interface GigabitEthernet0/0/0

# Debug OSPF (use carefully)
debug ip ospf spf statistic
debug ip ospf monitor
debug ip ospf database-timer
```

OSPF Design Best Practices

Scalability Guidelines

1. **Area Size**: Keep areas under 50 routers
2. **LSA Limits**: Monitor LSA count per area
3. **Hierarchy**: Maintain proper area hierarchy
4. **Summarization**: Implement aggressive summarization
5. **Stub Areas**: Use stub areas where appropriate

Performance Optimization

```
# Optimal OSPF configuration template
router ospf 1
 router-id 1.1.1.1
 auto-cost reference-bandwidth 100000
 timers throttle spf 5 50 5000
 timers throttle lsa 5 50 5000
 max-lsa 12000
 area 0 authentication message-digest
 area 1 stub
 area 1 range 192.168.0.0 255.255.252.0
 area 2 nssa default-information-originate
 passive-interface default
 no passive-interface GigabitEthernet0/0/0
 bfd all-interfaces
```

Security Considerations

```
# OSPF security hardening
router ospf 1
 area 0 authentication message-digest
 passive-interface default
 no passive-interface GigabitEthernet0/0/0
 distance ospf intra-area 90 inter-area 100 external 110

interface GigabitEthernet0/0/0
 ip ospf message-digest-key 1 md5 SecureOSPFKey123
 ip ospf network point-to-point
```

Summary

Advanced OSPF features provide the scalability, security, and performance needed for large enterprise networks. Understanding LSA types, area optimization, virtual links, and performance tuning is essential for CCNP-level network design and troubleshooting.

Key advanced concepts covered: - Advanced area types (stub, totally stubby, NSSA) - LSA types and database optimization - Virtual links for complex topologies - Advanced authentication mechanisms - Performance optimization techniques - Scalability best practices

In the next chapter, we'll explore advanced EIGRP features including named mode, IPv6 support, and advanced optimization techniques.

Review Questions

1. What are the differences between stub, totally stubby, and NSSA areas?
2. How do virtual links work and when are they necessary?
3. What are the different LSA types and their purposes?
4. How can you optimize OSPF performance in large networks?
5. What are the security considerations for OSPF deployment?

Hands-on Exercises

Exercise 1: Advanced Area Configuration

1. Deploy the advanced OSPF areas topology
2. Configure stub, totally stubby, and NSSA areas
3. Verify LSA filtering and default route injection
4. Test connectivity and analyze routing tables

Exercise 2: Virtual Link Implementation

1. Create a topology requiring virtual links
2. Configure and verify virtual link operation
3. Implement authentication on virtual links
4. Troubleshoot virtual link issues

Exercise 3: OSPF Performance Optimization

1. Configure SPF and LSA throttling
2. Implement BFD for fast convergence
3. Optimize hello and dead intervals
4. Monitor performance improvements

Exercise 4: Advanced Troubleshooting

1. Create complex OSPF problems (database issues, authentication failures)
2. Use advanced diagnostic commands
3. Develop systematic troubleshooting procedures
4. Document solutions and prevention strategies

Additional Resources

- OSPF Design Guide
- OSPF LSA Types Explained
- OSPF Virtual Links Configuration
- OSPF Performance Tuning

Chapter 31: BGP Fundamentals and Configuration

Learning Objectives

By the end of this chapter, you will be able to: - Understand BGP concepts and operation principles - Configure eBGP and iBGP peering relationships - Implement BGP path selection and attribute manipulation - Configure BGP route filtering and policy implementation - Troubleshoot BGP connectivity and routing issues

BGP Fundamentals

What is BGP?

Border Gateway Protocol (BGP) is the routing protocol that makes the Internet work. It's a path-vector protocol designed to exchange routing information between autonomous systems (AS). BGP is used both between different organizations (eBGP) and within large organizations (iBGP).

Key BGP Characteristics

- **Path Vector Protocol**: Maintains path information to prevent loops
- **Policy-Based**: Extensive policy control and manipulation
- **Scalable**: Handles hundreds of thousands of routes
- **Reliable**: Uses TCP for reliable transport
- **Flexible**: Rich set of attributes for path selection

BGP vs IGP Comparison

Aspect	BGP	IGP (OSPF/EIGRP)
Purpose	Inter-AS routing	Intra-AS routing
Metric	Path attributes	Cost/Metric
Convergence	Slow, stable	Fast convergence
Scalability	Very high	Limited
Policy Control	Extensive	Limited
Loop Prevention	AS-Path	SPF/DUAL

BGP Message Types

Type	Name	Purpose
1	OPEN	Establish BGP session
2	UPDATE	Exchange routing information
3	NOTIFICATION	Error reporting
4	KEEPALIVE	Maintain session

BGP Session Establishment

BGP Neighbor States

BGP neighbors go through several states during session establishment:

1. **Idle**: Initial state, no BGP process
2. **Connect**: TCP connection attempt
3. **Active**: TCP connection failed, retrying
4. **OpenSent**: TCP established, OPEN message sent
5. **OpenConfirm**: OPEN message received and processed
6. **Established**: BGP session fully established

Basic BGP Lab Setup

```yaml
# BGP fundamentals lab
name: bgp-fundamentals
prefix: bgp

topology:
  nodes:
    # ISP A (AS 100)
    isp-a-r1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname ISP-A-R1
        !
        interface Loopback0
         ip address 1.1.1.1 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description To-ISP-B
         ip address 10.1.12.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description To-Customer-A
         ip address 10.1.13.1 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description ISP-A-Internal
         ip address 192.168.100.1 255.255.255.252
         no shutdown
        !
        ! BGP Configuration
        router bgp 100
         bgp router-id 1.1.1.1
         bgp log-neighbor-changes
         ! eBGP to ISP-B
         neighbor 10.1.12.2 remote-as 200
         neighbor 10.1.12.2 description ISP-B-R1
         ! eBGP to Customer A
         neighbor 10.1.13.2 remote-as 300
         neighbor 10.1.13.2 description Customer-A
         ! iBGP to internal router
         neighbor 192.168.100.2 remote-as 100
         neighbor 192.168.100.2 description ISP-A-R2
         neighbor 192.168.100.2 update-source GigabitEthernet0/0/2
         !
         ! Network advertisements
         network 1.1.1.1 mask 255.255.255.255
         network 192.168.100.0 mask 255.255.255.252
        !
        ! Static route for demonstration
        ip route 203.0.113.0 255.255.255.0 Null0
        !

    # ISP A Internal Router
    isp-a-r2:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname ISP-A-R2
        !
        interface Loopback0
         ip address 2.2.2.2 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description ISP-A-Internal
```

```
        ip address 192.168.100.2 255.255.255.252
        no shutdown
       !
       interface GigabitEthernet0/0/1
        description To-Customer-B
        ip address 10.1.24.2 255.255.255.252
        no shutdown
       !
       ! BGP Configuration
       router bgp 100
        bgp router-id 2.2.2.2
        bgp log-neighbor-changes
        ! iBGP to ISP-A-R1
        neighbor 192.168.100.1 remote-as 100
        neighbor 192.168.100.1 description ISP-A-R1
        neighbor 192.168.100.1 update-source GigabitEthernet0/0/0
        ! eBGP to Customer B
        neighbor 10.1.24.4 remote-as 400
        neighbor 10.1.24.4 description Customer-B
        !
        network 2.2.2.2 mask 255.255.255.255
       !

   # ISP B (AS 200)
   isp-b-r1:
     kind: cisco_iosxe
     image: cisco/iosxe:latest
     mgmt-ipv4: 172.20.20.12
     startup-config: |
       hostname ISP-B-R1
       !
       interface Loopback0
        ip address 3.3.3.3 255.255.255.255
       !
       interface GigabitEthernet0/0/0
        description To-ISP-A
        ip address 10.1.12.2 255.255.255.252
        no shutdown
       !
       interface GigabitEthernet0/0/1
        description To-Internet
        ip address 203.0.113.1 255.255.255.252
        no shutdown
       !
       ! BGP Configuration
       router bgp 200
        bgp router-id 3.3.3.3
```

```
      bgp log-neighbor-changes
      ! eBGP to ISP-A
      neighbor 10.1.12.1 remote-as 100
      neighbor 10.1.12.1 description ISP-A-R1
      !
      network 3.3.3.3 mask 255.255.255.255
      network 203.0.113.0 mask 255.255.255.252
     !

# Customer A (AS 300)
customer-a:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.13
  startup-config: |
    hostname Customer-A
    !
    interface Loopback0
     ip address 4.4.4.4 255.255.255.255
    !
    interface GigabitEthernet0/0/0
     description To-ISP-A
     ip address 10.1.13.2 255.255.255.252
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description Customer-A-LAN
     ip address 192.168.10.1 255.255.255.0
     no shutdown
    !
    ! BGP Configuration
    router bgp 300
     bgp router-id 4.4.4.4
     bgp log-neighbor-changes
     ! eBGP to ISP-A
     neighbor 10.1.13.1 remote-as 100
     neighbor 10.1.13.1 description ISP-A-R1
     !
     network 4.4.4.4 mask 255.255.255.255
     network 192.168.10.0 mask 255.255.255.0
    !

# Customer B (AS 400)
customer-b:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.14
```

```
    startup-config: |
      hostname Customer-B
      !
      interface Loopback0
       ip address 5.5.5.5 255.255.255.255
      !
      interface GigabitEthernet0/0/0
       description To-ISP-A
       ip address 10.1.24.4 255.255.255.252
       no shutdown
      !
      interface GigabitEthernet0/0/1
       description Customer-B-LAN
       ip address 192.168.20.1 255.255.255.0
       no shutdown
      !
      ! BGP Configuration
      router bgp 400
       bgp router-id 5.5.5.5
       bgp log-neighbor-changes
       ! eBGP to ISP-A
       neighbor 10.1.24.2 remote-as 100
       neighbor 10.1.24.2 description ISP-A-R2
       !
       network 5.5.5.5 mask 255.255.255.255
       network 192.168.20.0 mask 255.255.255.0
      !

  # Internet simulation
  internet-sim:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 172.20.20.100
    exec:
      - ip addr add 203.0.113.2/30 dev eth1
      - ip route add default via 203.0.113.1

  # Customer LANs
  customer-a-pc:
    kind: linux
    image: alpine:latest
    exec:
      - ip addr add 192.168.10.10/24 dev eth1
      - ip route add default via 192.168.10.1

  customer-b-pc:
    kind: linux
```

```
      image: alpine:latest
      exec:
        - ip addr add 192.168.20.10/24 dev eth1
        - ip route add default via 192.168.20.1

  links:
    # ISP interconnections
    - endpoints: ["isp-a-r1:eth1", "isp-b-r1:eth1"]
    - endpoints: ["isp-a-r1:eth3", "isp-a-r2:eth1"]

    # Customer connections
    - endpoints: ["isp-a-r1:eth2", "customer-a:eth1"]
    - endpoints: ["isp-a-r2:eth2", "customer-b:eth1"]

    # Internet connection
    - endpoints: ["isp-b-r1:eth2", "internet-sim:eth1"]

    # Customer LANs
    - endpoints: ["customer-a:eth2", "customer-a-pc:eth1"]
    - endpoints: ["customer-b:eth2", "customer-b-pc:eth1"]
```

Basic BGP Configuration

eBGP Configuration

```
# Deploy the BGP lab
containerlab deploy -t bgp-fundamentals.yml

# Connect to ISP-A-R1 and verify BGP
docker exec -it clab-bgp-isp-a-r1 cli

# Basic eBGP configuration
configure terminal
router bgp 100
 bgp router-id 1.1.1.1
 neighbor 10.1.12.2 remote-as 200
 neighbor 10.1.12.2 description ISP-B-R1
 network 1.1.1.1 mask 255.255.255.255
exit

# Verify BGP neighbors
show ip bgp summary
show ip bgp neighbors
```

iBGP Configuration

```
# iBGP configuration
router bgp 100
 neighbor 192.168.100.2 remote-as 100
 neighbor 192.168.100.2 update-source Loopback0
 neighbor 192.168.100.2 next-hop-self

# Verify iBGP session
show ip bgp summary
show ip bgp neighbors 192.168.100.2
```

BGP Attributes and Path Selection

BGP Path Attributes

BGP uses various attributes to determine the best path to a destination:

Well-Known Mandatory Attributes

1. **ORIGIN**: How the route was originated (IGP, EGP, Incomplete)
2. **AS_PATH**: List of AS numbers the route has traversed
3. **NEXT_HOP**: Next-hop IP address for the route

Well-Known Discretionary Attributes

1. **LOCAL_PREF**: Local preference within an AS
2. **ATOMIC_AGGREGATE**: Route summarization indicator

Optional Transitive Attributes

1. **AGGREGATOR**: Router that performed aggregation
2. **COMMUNITY**: Route tagging for policy application

Optional Non-Transitive Attributes

1. **MED (Multi-Exit Discriminator)**: Metric to influence inbound traffic
2. **ORIGINATOR_ID**: Route reflector loop prevention
3. **CLUSTER_LIST**: Route reflector cluster information

BGP Path Selection Algorithm

BGP uses the following criteria in order:

1. **Highest Weight** (Cisco proprietary)
2. **Highest Local Preference**
3. **Locally originated routes**
4. **Shortest AS Path**
5. **Lowest Origin code** (IGP < EGP < Incomplete)
6. **Lowest MED**
7. **eBGP over iBGP**
8. **Lowest IGP metric to next-hop**
9. **Oldest route**
10. **Lowest Router ID**
11. **Shortest cluster list**
12. **Lowest neighbor address**

Attribute Manipulation Lab

```
# BGP attribute manipulation
startup-config: |
  ! Configure route-maps for attribute manipulation
  route-map SET-LOCAL-PREF permit 10
   match ip address prefix-list CUSTOMER-ROUTES
   set local-preference 200
  !
  route-map SET-LOCAL-PREF permit 20
   set local-preference 100
  !
  route-map SET-MED permit 10
   match ip address prefix-list PREFERRED-ROUTES
   set metric 50
  !
  route-map SET-MED permit 20
   set metric 100
  !
  ! Apply route-maps to neighbors
  router bgp 100
   neighbor 10.1.12.2 route-map SET-MED out
   neighbor 192.168.100.2 route-map SET-LOCAL-PREF in
  !
  ! Define prefix lists
  ip prefix-list CUSTOMER-ROUTES seq 10 permit 192.168.10.0/24
  ip prefix-list PREFERRED-ROUTES seq 10 permit 1.1.1.1/32
```

BGP Route Filtering

Prefix Lists

```
# Configure prefix lists for filtering
ip prefix-list ALLOW-CUSTOMER seq 10 permit 192.168.10.0/24
ip prefix-list ALLOW-CUSTOMER seq 20 permit 192.168.20.0/24
ip prefix-list DENY-PRIVATE seq 10 deny 10.0.0.0/8 le 32
ip prefix-list DENY-PRIVATE seq 20 deny 172.16.0.0/12 le 32
ip prefix-list DENY-PRIVATE seq 30 deny 192.168.0.0/16 le 32
ip prefix-list DENY-PRIVATE seq 40 permit 0.0.0.0/0 le 32

# Apply to BGP neighbors
router bgp 100
 neighbor 10.1.12.2 prefix-list ALLOW-CUSTOMER out
 neighbor 10.1.12.2 prefix-list DENY-PRIVATE in
```

AS-Path Filtering

```
# Configure AS-path access lists
ip as-path access-list 1 permit ^$
ip as-path access-list 1 deny .*
ip as-path access-list 2 permit ^100_
ip as-path access-list 3 deny _300_
ip as-path access-list 3 permit .*

# Apply AS-path filtering
router bgp 100
 neighbor 10.1.12.2 filter-list 1 out
 neighbor 10.1.13.2 filter-list 2 in
```

Community-Based Filtering

```
# Configure community lists
ip community-list standard CUSTOMER permit 100:100
ip community-list standard NO-EXPORT permit no-export
ip community-list expanded BACKUP-PATH permit _200:.*

# Route-map with community matching
route-map COMMUNITY-FILTER permit 10
 match community CUSTOMER
 set local-preference 200
!
```

```
route-map COMMUNITY-FILTER permit 20
 match community NO-EXPORT
 set community no-advertise
!
route-map COMMUNITY-FILTER permit 30

# Apply community-based filtering
router bgp 100
 neighbor 192.168.100.2 route-map COMMUNITY-FILTER in
```

Advanced BGP Configuration

BGP Timers

```
# Configure BGP timers
router bgp 100
 timers bgp 30 90
 # Keepalive: 30 seconds, Hold time: 90 seconds

# Per-neighbor timers
router bgp 100
 neighbor 10.1.12.2 timers 10 30
```

BGP Authentication

```
# Configure BGP authentication
router bgp 100
 neighbor 10.1.12.2 password BGPSecretKey123

# Verify authentication
show ip bgp neighbors 10.1.12.2 | include Authentication
```

BGP Soft Reconfiguration

```
# Enable soft reconfiguration
router bgp 100
 neighbor 10.1.12.2 soft-reconfiguration inbound

# Perform soft reset
clear ip bgp 10.1.12.2 soft in
clear ip bgp 10.1.12.2 soft out
```

BGP Troubleshooting

Common BGP Issues

Neighbor Adjacency Problems

```
# Check BGP neighbor status
show ip bgp summary
show ip bgp neighbors

# Common issues and solutions:
# 1. TCP connectivity
telnet 10.1.12.2 179

# 2. AS number mismatch
router bgp 100
 neighbor 10.1.12.2 remote-as 200

# 3. Authentication failure
router bgp 100
 neighbor 10.1.12.2 password CorrectPassword

# 4. Update source mismatch
router bgp 100
 neighbor 192.168.100.2 update-source Loopback0
```

Route Advertisement Issues

```
# Check route advertisement
show ip bgp
show ip bgp neighbors 10.1.12.2 advertised-routes
show ip bgp neighbors 10.1.12.2 received-routes

# Debug BGP updates (use carefully)
debug ip bgp updates
debug ip bgp keepalives
```

Path Selection Problems

```
# Analyze path selection
show ip bgp 192.168.10.0/24
show ip bgp regexp ^300$
show ip route bgp
```

```
# Check BGP attributes
show ip bgp 192.168.10.0/24 bestpath
show ip bgp attribute-map
```

BGP Monitoring Commands

```
# Essential BGP show commands
show ip bgp summary
show ip bgp neighbors
show ip bgp
show ip bgp regexp
show ip bgp community
show ip bgp dampening dampened-paths

# BGP statistics
show ip bgp statistics
show ip bgp peer-group
show ip bgp update-group
```

BGP Security Considerations

BGP Security Best Practices

```
# BGP security configuration
router bgp 100
 ! Authentication
 neighbor 10.1.12.2 password SecureBGPKey123

 ! Maximum prefixes
 neighbor 10.1.12.2 maximum-prefix 1000 75 warning-only

 ! Route filtering
 neighbor 10.1.12.2 prefix-list CUSTOMER-ROUTES out
 neighbor 10.1.12.2 prefix-list PROVIDER-ROUTES in

 ! Disable unnecessary features
 no bgp default ipv4-unicast
 no synchronization

 ! BGP dampening
 bgp dampening 15 750 2000 60
```

Route Hijacking Prevention

```
# Implement route origin validation
ip prefix-list VALID-ORIGINS seq 10 permit 192.168.10.0/24
ip prefix-list VALID-ORIGINS seq 20 permit 192.168.20.0/24

route-map ORIGIN-VALIDATION permit 10
 match ip address prefix-list VALID-ORIGINS
!
route-map ORIGIN-VALIDATION deny 20

router bgp 100
 neighbor 10.1.12.2 route-map ORIGIN-VALIDATION in
```

BGP Optimization

BGP Performance Tuning

```
# Optimize BGP performance
router bgp 100
 ! Reduce convergence time
 bgp fast-external-fallover
 bgp bestpath as-path multipath-relax

 ! Memory optimization
 bgp scan-time 60
 bgp update-delay 120

 ! CPU optimization
 bgp dampening
 maximum-paths 4
```

BGP Route Aggregation

```
# Configure route aggregation
router bgp 100
 aggregate-address 192.168.0.0 255.255.252.0
 aggregate-address 192.168.0.0 255.255.252.0 summary-only
 aggregate-address 192.168.0.0 255.255.252.0 as-set
 aggregate-address 192.168.0.0 255.255.252.0 suppress-map SUPPRESS-SPECIFIC

route-map SUPPRESS-SPECIFIC permit 10
 match ip address prefix-list SPECIFIC-ROUTES
```

```
ip prefix-list SPECIFIC-ROUTES seq 10 permit 192.168.1.0/24
ip prefix-list SPECIFIC-ROUTES seq 20 permit 192.168.2.0/24
```

BGP Design Patterns

Hub-and-Spoke BGP Design

```
# Hub router configuration
router bgp 100
 neighbor 10.1.1.2 remote-as 200
 neighbor 10.1.1.2 route-reflector-client
 neighbor 10.1.2.2 remote-as 200
 neighbor 10.1.2.2 route-reflector-client
 neighbor 10.1.3.2 remote-as 200
 neighbor 10.1.3.2 route-reflector-client
```

Multi-Homed BGP Design

```
# Multi-homed customer configuration
router bgp 300
 ! Primary ISP
 neighbor 10.1.13.1 remote-as 100
 neighbor 10.1.13.1 route-map PRIMARY-ISP out

 ! Backup ISP
 neighbor 10.2.13.1 remote-as 200
 neighbor 10.2.13.1 route-map BACKUP-ISP out

route-map PRIMARY-ISP permit 10
 set as-path prepend 300

route-map BACKUP-ISP permit 10
 set as-path prepend 300 300 300
```

Summary

BGP is the foundation of Internet routing and a critical protocol for enterprise networks connecting to multiple ISPs. Understanding BGP fundamentals, path selection, attribute manipulation, and security considerations is essential for CCNP-level network design and operation.

Key concepts covered: - BGP session establishment and neighbor states - eBGP and iBGP configuration differences - BGP attributes and path selection algorithm - Route filtering using prefix lists and AS-path filters - BGP security and optimization techniques - Common troubleshooting scenarios

In the next chapter, we'll explore advanced BGP features including route reflectors, confederations, and complex policy implementations.

Review Questions

1. What are the differences between eBGP and iBGP?
2. How does BGP path selection algorithm work?
3. What are the main BGP attributes and their purposes?
4. How do you implement BGP route filtering?
5. What are common BGP security considerations?

Hands-on Exercises

Exercise 1: Basic BGP Configuration

1. Deploy the BGP fundamentals lab
2. Configure eBGP and iBGP peering
3. Verify BGP neighbor establishment
4. Test route advertisement and path selection

Exercise 2: BGP Attribute Manipulation

1. Configure route-maps to modify BGP attributes
2. Test LOCAL_PREF and MED manipulation
3. Implement AS-path prepending
4. Verify path selection changes

Exercise 3: BGP Route Filtering

1. Configure prefix lists for route filtering
2. Implement AS-path filtering
3. Use community attributes for policy
4. Test filtering effectiveness

Exercise 4: BGP Troubleshooting

1. Create various BGP problems (neighbor issues, route filtering)
2. Practice diagnostic commands and procedures
3. Develop systematic troubleshooting approaches
4. Document solutions and prevention strategies

Additional Resources

- BGP Configuration Guide
- BGP Best Practices
- BGP Security Recommendations
- BGP Troubleshooting Guide

Chapter 33: Advanced STP and MST

Learning Objectives

By the end of this chapter, you will be able to: - Configure Multiple Spanning Tree (MST) protocol - Implement advanced STP optimization techniques - Configure STP security features and protection mechanisms - Troubleshoot complex spanning tree scenarios - Design resilient Layer 2 topologies with optimal STP configuration

Multiple Spanning Tree (MST) Protocol

MST Fundamentals

Multiple Spanning Tree (MST) allows multiple VLANs to be mapped to a single spanning tree instance, reducing the number of spanning tree instances while providing load balancing across different VLANs.

MST Benefits

1. **Scalability**: Reduces number of spanning tree instances
2. **Load Balancing**: Different VLANs can use different paths
3. **Convergence**: Faster convergence than PVST+
4. **Bandwidth Efficiency**: Fewer BPDUs transmitted
5. **Interoperability**: Works with RSTP and PVST+

MST Concepts

MST Regions

- **MST Region**: Group of switches with same MST configuration
- **Region Name**: Text string identifying the region
- **Revision Number**: Configuration version number
- **VLAN-to-Instance Mapping**: Which VLANs belong to which instance

MST Instances

- **Instance 0 (IST)**: Internal Spanning Tree, always exists
- **Instance 1-4094**: User-defined instances
- **MSTI**: Multiple Spanning Tree Instance

MST Lab Environment

```
# MST demonstration lab
name: mst-advanced
prefix: mst

topology:
  nodes:
    # Core switches with MST
    core-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Core-SW1
        !
        ! Enable MST globally
        spanning-tree mode mst
        !
        ! MST configuration
        spanning-tree mst configuration
         name ENTERPRISE-REGION
         revision 1
         instance 1 vlan 10,30,50,70
         instance 2 vlan 20,40,60,80
         instance 3 vlan 100-199
        !
        ! MST instance priorities
        spanning-tree mst 0 priority 4096
        spanning-tree mst 1 priority 4096
        spanning-tree mst 2 priority 8192
        spanning-tree mst 3 priority 4096
        !
        ! VLANs
        vlan 10,20,30,40,50,60,70,80
        vlan 100-199
        !
        ! Trunk interfaces
        interface range GigabitEthernet1/0/1-4
         switchport mode trunk
         switchport trunk allowed vlan 10,20,30,40,50,60,70,80,100-199
         spanning-tree portfast trunk
         no shutdown
        !

    core-sw2:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Core-SW2
        !
        spanning-tree mode mst
        !
        spanning-tree mst configuration
         name ENTERPRISE-REGION
         revision 1
         instance 1 vlan 10,30,50,70
         instance 2 vlan 20,40,60,80
         instance 3 vlan 100-199
        !
        ! MST instance priorities (alternate root)
        spanning-tree mst 0 priority 8192
        spanning-tree mst 1 priority 8192
        spanning-tree mst 2 priority 4096
        spanning-tree mst 3 priority 8192
        !
        vlan 10,20,30,40,50,60,70,80
        vlan 100-199
        !
        interface range GigabitEthernet1/0/1-4
         switchport mode trunk
         switchport trunk allowed vlan 10,20,30,40,50,60,70,80,100-199
         no shutdown
        !

    # Distribution switches
    dist-sw1:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.12
      startup-config: |
        hostname Dist-SW1
        !
        spanning-tree mode mst
        !
        spanning-tree mst configuration
```

```
      name ENTERPRISE-REGION
      revision 1
      instance 1 vlan 10,30,50,70
      instance 2 vlan 20,40,60,80
      instance 3 vlan 100-199
     !
     ! MST priorities for load balancing
     spanning-tree mst 0 priority 16384
     spanning-tree mst 1 priority 12288
     spanning-tree mst 2 priority 16384
     spanning-tree mst 3 priority 12288
     !
     vlan 10,20,30,40,50,60,70,80
     vlan 100-199
     !
     ! Uplink trunks
     interface range GigabitEthernet1/0/1-2
      switchport mode trunk
      switchport trunk allowed vlan 10,20,30,40,50,60,70,80,100-199
      no shutdown
     !
     ! Access ports
     interface range GigabitEthernet1/0/3-10
      switchport mode access
      switchport access vlan 10
      spanning-tree portfast
      spanning-tree bpduguard enable
      no shutdown
     !

 dist-sw2:
   kind: cisco_iosxe
   image: cisco/catalyst:latest
   mgmt-ipv4: 172.20.20.13
   startup-config: |
     hostname Dist-SW2
     !
     spanning-tree mode mst
     !
     spanning-tree mst configuration
      name ENTERPRISE-REGION
      revision 1
      instance 1 vlan 10,30,50,70
      instance 2 vlan 20,40,60,80
      instance 3 vlan 100-199
     !
     spanning-tree mst 0 priority 16384
```

```
      spanning-tree mst 1 priority 16384
      spanning-tree mst 2 priority 12288
      spanning-tree mst 3 priority 16384
      !
      vlan 10,20,30,40,50,60,70,80
      vlan 100-199
      !
      interface range GigabitEthernet1/0/1-2
       switchport mode trunk
       switchport trunk allowed vlan 10,20,30,40,50,60,70,80,100-199
       no shutdown
      !
      interface range GigabitEthernet1/0/3-10
       switchport mode access
       switchport access vlan 20
       spanning-tree portfast
       spanning-tree bpduguard enable
       no shutdown
      !

  # Access switches
  access-sw1:
    kind: cisco_iosxe
    image: cisco/catalyst:latest
    mgmt-ipv4: 172.20.20.14
    startup-config: |
      hostname Access-SW1
      !
      spanning-tree mode mst
      !
      spanning-tree mst configuration
       name ENTERPRISE-REGION
       revision 1
       instance 1 vlan 10,30,50,70
       instance 2 vlan 20,40,60,80
       instance 3 vlan 100-199
      !
      vlan 10,20,30,40
      !
      ! Uplink to distribution
      interface GigabitEthernet1/0/1
       switchport mode trunk
       switchport trunk allowed vlan 10,20,30,40
       no shutdown
      !
      ! Access ports with different VLANs
      interface range GigabitEthernet1/0/2-5
```

```
          switchport mode access
          switchport access vlan 10
          spanning-tree portfast
          spanning-tree bpduguard enable
          no shutdown
         !
         interface range GigabitEthernet1/0/6-9
          switchport mode access
          switchport access vlan 20
          spanning-tree portfast
          spanning-tree bpduguard enable
          no shutdown
         !

    # Test devices
    pc1:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.10.10/24 dev eth1

    pc2:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.20.10/24 dev eth1

  links:
    # Core interconnection
    - endpoints: ["core-sw1:eth1", "core-sw2:eth1"]
    - endpoints: ["core-sw1:eth2", "core-sw2:eth2"]

    # Core to distribution
    - endpoints: ["core-sw1:eth3", "dist-sw1:eth1"]
    - endpoints: ["core-sw1:eth4", "dist-sw2:eth1"]
    - endpoints: ["core-sw2:eth3", "dist-sw1:eth2"]
    - endpoints: ["core-sw2:eth4", "dist-sw2:eth2"]

    # Distribution to access
    - endpoints: ["dist-sw1:eth3", "access-sw1:eth1"]

    # End devices
    - endpoints: ["access-sw1:eth2", "pc1:eth1"]
    - endpoints: ["access-sw1:eth6", "pc2:eth1"]
```

MST Configuration

Basic MST Setup

```
# Deploy MST lab
containerlab deploy -t mst-advanced.yml

# Configure MST on Core-SW1
docker exec -it clab-mst-core-sw1 cli

configure terminal
! Enable MST mode
spanning-tree mode mst

! Configure MST region
spanning-tree mst configuration
 name ENTERPRISE-REGION
 revision 1
 instance 1 vlan 10,30,50,70
 instance 2 vlan 20,40,60,80
 instance 3 vlan 100-199
 exit

! Set MST priorities
spanning-tree mst 0 priority 4096
spanning-tree mst 1 priority 4096
spanning-tree mst 2 priority 8192
```

MST Verification

```
# Verify MST configuration
show spanning-tree mst configuration
show spanning-tree mst
show spanning-tree mst 1
show spanning-tree mst interface GigabitEthernet1/0/1

# Check MST region consistency
show spanning-tree mst configuration digest
```

Advanced STP Features

Rapid Spanning Tree Protocol (RSTP) Enhancements

Port Roles and States

```
# RSTP port roles
# Root Port: Best path to root bridge
# Designated Port: Forwarding port on segment
# Alternate Port: Backup path to root
# Backup Port: Backup designated port

# RSTP port states
# Discarding: Not forwarding, learning, or relaying
# Learning: Not forwarding but learning MAC addresses
# Forwarding: Fully operational

# Configure RSTP
spanning-tree mode rapid-pvst
spanning-tree vlan 1-4094 priority 4096
```

Edge Ports and Link Types

```
# Configure edge ports (PortFast equivalent)
interface range GigabitEthernet1/0/1-24
 spanning-tree portfast
 spanning-tree bpduguard enable

# Configure link types
interface GigabitEthernet1/0/1
 spanning-tree link-type point-to-point

interface GigabitEthernet1/0/2
 spanning-tree link-type shared
```

STP Optimization Features

Root Guard

Prevents unauthorized switches from becoming root bridge.

```
# Configure Root Guard
interface GigabitEthernet1/0/1
 spanning-tree guard root
```

```
# Verify Root Guard
show spanning-tree interface GigabitEthernet1/0/1 detail
```

Loop Guard

Prevents alternate or root ports from becoming designated ports due to unidirectional failures.

```
# Configure Loop Guard globally
spanning-tree loopguard default

# Configure Loop Guard per interface
interface GigabitEthernet1/0/1
 spanning-tree guard loop

# Verify Loop Guard
show spanning-tree interface GigabitEthernet1/0/1 detail
```

BPDU Guard

Disables ports that receive BPDUs when they shouldn't.

```
# Configure BPDU Guard globally
spanning-tree portfast bpduguard default

# Configure BPDU Guard per interface
interface GigabitEthernet1/0/1
 spanning-tree portfast
 spanning-tree bpduguard enable

# Recover from BPDU Guard
errdisable recovery cause bpduguard
errdisable recovery interval 300
```

BPDU Filter

Prevents sending or receiving BPDUs on specific ports.

```
# Configure BPDU Filter
interface GigabitEthernet1/0/1
 spanning-tree bpdufilter enable

# Global BPDU Filter for PortFast ports
spanning-tree portfast bpdufilter default
```

Unidirectional Link Detection (UDLD)

UDLD detects and disables unidirectional links that can cause spanning tree loops.

```
# Enable UDLD globally
udld enable
udld aggressive

# Configure UDLD per interface
interface GigabitEthernet1/0/1
 udld port aggressive

# Verify UDLD
show udld
show udld interface GigabitEthernet1/0/1
```

STP Security Features

STP Attack Mitigation

BPDU Attack Protection

```
# Protect against BPDU attacks
spanning-tree portfast bpduguard default
spanning-tree loopguard default

# Rate limit BPDUs
interface GigabitEthernet1/0/1
 storm-control broadcast level 50.00
 storm-control multicast level 50.00
 storm-control action shutdown
```

Root Bridge Protection

```
# Secure root bridge selection
spanning-tree vlan 1-4094 root primary
spanning-tree vlan 1-4094 priority 0

# Monitor root bridge changes
spanning-tree logging

# Root bridge backup
spanning-tree vlan 1-4094 root secondary
```

STP Hardening Configuration

```
# Comprehensive STP security configuration
! Global settings
spanning-tree mode rapid-pvst
spanning-tree portfast bpduguard default
spanning-tree portfast bpdufilter default
spanning-tree loopguard default
udld aggressive

! Root bridge security
spanning-tree vlan 1-4094 priority 0
spanning-tree logging

! Interface security template
interface range GigabitEthernet1/0/1-24
 switchport mode access
 spanning-tree portfast
 spanning-tree bpduguard enable
 udld port aggressive
 storm-control broadcast level 50.00
 storm-control action shutdown

! Trunk interface security
interface range GigabitEthernet1/0/25-28
 switchport mode trunk
 spanning-tree guard root
 udld port aggressive
```

Advanced STP Troubleshooting

Common STP Issues

Topology Changes

```
# Monitor topology changes
show spanning-tree vlan 1 | include changes
show spanning-tree summary totals

# Debug topology changes (use carefully)
debug spanning-tree events
debug spanning-tree root
debug spanning-tree topology
```

Convergence Issues

```
# Analyze convergence problems
show spanning-tree interface GigabitEthernet1/0/1 detail
show spanning-tree vlan 1 detail
show spanning-tree blockedports

# Check for inconsistent port states
show spanning-tree inconsistentports
show spanning-tree interface GigabitEthernet1/0/1 portfast
```

MST Troubleshooting

```
# MST-specific troubleshooting
show spanning-tree mst configuration
show spanning-tree mst configuration digest
show spanning-tree mst 1 detail

# Check MST region consistency
show spanning-tree mst interface GigabitEthernet1/0/1 detail
show spanning-tree mst 1 interface GigabitEthernet1/0/1
```

STP Diagnostic Commands

```
# Essential STP show commands
show spanning-tree
show spanning-tree summary
show spanning-tree vlan 1
show spanning-tree interface GigabitEthernet1/0/1
show spanning-tree root
show spanning-tree bridge

# Advanced diagnostics
show spanning-tree pathcost method
show spanning-tree uplinkfast
show spanning-tree backbonefast
show errdisable recovery
show udld
```

STP Performance Optimization

Convergence Optimization

Timer Tuning

```
# Optimize STP timers (use carefully)
spanning-tree vlan 1-4094 hello-time 1
spanning-tree vlan 1-4094 forward-time 4
spanning-tree vlan 1-4094 max-age 6

# MST timer optimization
spanning-tree mst hello-time 1
spanning-tree mst forward-time 4
spanning-tree mst max-age 6
```

UplinkFast and BackboneFast

```
# Configure UplinkFast (access layer)
spanning-tree uplinkfast

# Configure BackboneFast (all switches)
spanning-tree backbonefast

# Verify optimization features
show spanning-tree uplinkfast
show spanning-tree backbonefast
```

Load Balancing with MST

```
# Optimize MST load balancing
spanning-tree mst 1 priority 4096
spanning-tree mst 2 priority 8192

# Per-VLAN load balancing
spanning-tree vlan 10,30,50 priority 4096
spanning-tree vlan 20,40,60 priority 8192

# Verify load balancing
show spanning-tree mst 1 | include Root
show spanning-tree mst 2 | include Root
```

STP Design Best Practices

Hierarchical STP Design

```
# Core layer configuration
spanning-tree mode mst
spanning-tree mst 0 priority 0
spanning-tree mst 1 priority 0
spanning-tree mst 2 priority 4096

# Distribution layer configuration
spanning-tree mode mst
spanning-tree mst 0 priority 4096
spanning-tree mst 1 priority 4096
spanning-tree mst 2 priority 0

# Access layer configuration
spanning-tree mode mst
spanning-tree portfast default
spanning-tree portfast bpduguard default
```

STP Scalability Guidelines

1. **Use MST**: Reduce spanning tree instances
2. **Proper Root Placement**: Place root bridges in core
3. **Load Balancing**: Distribute traffic across links
4. **Security Features**: Enable protection mechanisms
5. **Monitoring**: Implement comprehensive monitoring

STP Monitoring and Maintenance

```
# STP monitoring script
#!/bin/bash
echo "=== STP Status Report ==="
echo "Root Bridge Status:"
show spanning-tree root | include Root

echo "Topology Changes:"
show spanning-tree summary totals | include changes

echo "Blocked Ports:"
show spanning-tree blockedports

echo "Error Disabled Ports:"
```

```
show interfaces status | include err-disabled

echo "UDLD Status:"
show udld | include Port
```

Testing STP Scenarios

Failure Simulation

```
# Simulate link failure
interface GigabitEthernet1/0/1
 shutdown

# Monitor convergence
show spanning-tree vlan 1 | include Root
show spanning-tree interface GigabitEthernet1/0/2 detail

# Restore link
interface GigabitEthernet1/0/1
 no shutdown
```

Load Testing

```
# Generate traffic for load balancing test
docker exec -it clab-mst-pc1 sh
ping 192.168.20.10 &

# Monitor traffic distribution
docker exec -it clab-mst-core-sw1 cli
show interfaces GigabitEthernet1/0/1 | include rate
show interfaces GigabitEthernet1/0/2 | include rate
```

Summary

Advanced STP features and MST provide the foundation for resilient and efficient Layer 2 networks. Understanding MST configuration, STP security features, and optimization techniques is essential for designing scalable enterprise networks with proper loop prevention and load balancing.

Key concepts covered: - Multiple Spanning Tree (MST) protocol configuration - Advanced STP features (Root Guard, Loop Guard, BPDU Guard) - STP security and attack mitigation - Performance optimization and convergence tuning - Troubleshooting complex spanning tree scenarios

In the next chapter, we'll explore advanced VLAN features including Private VLANs and VTP configuration.

Review Questions

1. What are the benefits of MST over PVST+?
2. How do Root Guard and Loop Guard protect STP topology?
3. What is the purpose of BPDU Guard and BPDU Filter?
4. How do you optimize STP convergence time?
5. What are best practices for STP security hardening?

Hands-on Exercises

Exercise 1: MST Configuration

1. Deploy the MST lab topology
2. Configure MST regions and instances
3. Implement load balancing across instances
4. Verify MST operation and convergence

Exercise 2: STP Security Implementation

1. Configure Root Guard and Loop Guard
2. Implement BPDU Guard on access ports
3. Enable UDLD for unidirectional link detection
4. Test security features with attack simulations

Exercise 3: STP Optimization

1. Tune STP timers for faster convergence
2. Configure UplinkFast and BackboneFast
3. Implement proper root bridge placement
4. Monitor and verify optimization results

Exercise 4: Advanced STP Troubleshooting

1. Create complex STP problems and failures
2. Practice diagnostic commands and procedures
3. Analyze topology changes and convergence
4. Develop systematic troubleshooting approaches

Additional Resources

- MST Configuration Guide
- STP Best Practices
- STP Security Features
- Advanced STP Troubleshooting

Chapter 37: Quality of Service (QoS) Fundamentals

Learning Objectives

By the end of this chapter, you will be able to: - Understand QoS concepts and requirements - Implement traffic classification and marking - Configure queuing mechanisms and scheduling algorithms - Apply traffic shaping and policing techniques - Design QoS policies for voice, video, and data traffic

QoS Fundamentals

What is Quality of Service?

Quality of Service (QoS) is a set of technologies and techniques used to manage network resources and provide different levels of service to different types of traffic. QoS ensures that critical applications receive the network performance they require while managing bandwidth efficiently.

Why QoS is Needed

1. **Limited Bandwidth**: Network links have finite capacity
2. **Varying Traffic Types**: Different applications have different requirements
3. **Network Congestion**: Traffic bursts can overwhelm network resources
4. **Service Level Agreements**: Contractual obligations for performance
5. **User Experience**: Maintaining acceptable application performance

QoS Service Models

Best Effort

- **Default service**: No guarantees
- **FIFO queuing**: First in, first out
- **No differentiation**: All traffic treated equally
- **Suitable for**: Non-critical data applications

Integrated Services (IntServ)

- **Per-flow reservations**: RSVP protocol
- **Hard guarantees**: Strict resource allocation
- **Scalability issues**: State information per flow
- **Suitable for**: Small networks with specific requirements

Differentiated Services (DiffServ)

- **Class-based service**: Traffic aggregation
- **Scalable approach**: No per-flow state
- **Flexible policies**: Multiple service classes
- **Industry standard**: Most widely deployed

Traffic Characteristics

Voice Traffic

- **Bandwidth**: 64 Kbps (G.711) to 32 Kbps (G.729)
- **Delay**: < 150ms one-way
- **Jitter**: < 30ms
- **Loss**: < 1%
- **Characteristics**: Smooth, predictable, delay-sensitive

Video Traffic

- **Bandwidth**: 384 Kbps to 10+ Mbps
- **Delay**: < 200ms for interactive, < 5s for streaming
- **Jitter**: < 30ms for interactive
- **Loss**: < 0.1% for interactive, < 1% for streaming
- **Characteristics**: Bursty, variable bit rate

Data Traffic

- **Bandwidth**: Highly variable
- **Delay**: Generally tolerant (seconds to minutes)
- **Jitter**: Not critical
- **Loss**: Retransmission handles losses
- **Characteristics**: Bursty, elastic

QoS Lab Environment

Comprehensive QoS Lab Setup

```yaml
# QoS demonstration lab
name: qos-fundamentals
prefix: qos

topology:
  nodes:
    # Core router with QoS policies
    core-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname Core-Router
        !
        ! Enable QoS globally
        ip cef
        !
        interface GigabitEthernet0/0/0
         description To-Branch-Router
         ip address 10.1.12.1 255.255.255.252
         bandwidth 10000
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description To-Data-Center
         ip address 10.1.13.1 255.255.255.252
         bandwidth 100000
         no shutdown
        !
        interface GigabitEthernet0/0/2
         description To-Internet
         ip address 203.0.113.1 255.255.255.252
         bandwidth 50000
         no shutdown
        !
        ! QoS Class Maps
        class-map match-all VOICE
         match dscp ef
        !
        class-map match-all VIDEO
         match dscp af41 af42 af43
        !
        class-map match-all CRITICAL-DATA
         match dscp af31 af32 af33
        !
        class-map match-all BULK-DATA
         match dscp af11 af12 af13
        !
        ! QoS Policy Maps
        policy-map WAN-OUT
         class VOICE
          priority percent 20
          set dscp ef
         class VIDEO
          bandwidth percent 30
          set dscp af41
         class CRITICAL-DATA
          bandwidth percent 25
          set dscp af31
         class BULK-DATA
          bandwidth percent 15
          set dscp af11
         class class-default
          bandwidth percent 10
          fair-queue
        !
        ! Apply QoS policies
        interface GigabitEthernet0/0/0
         service-policy output WAN-OUT
        !

    # Branch router with traffic generation
    branch-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Branch-Router
        !
        interface GigabitEthernet0/0/0
         description To-Core-Router
         ip address 10.1.12.2 255.255.255.252
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description Voice-VLAN
         ip address 192.168.10.1 255.255.255.0
         no shutdown
        !
```

```
      interface GigabitEthernet0/0/2
       description Data-VLAN
       ip address 192.168.20.1 255.255.255.0
       no shutdown
      !
      ! Traffic classification
      class-map match-all VOICE-SIGNALING
       match protocol sip
      !
      class-map match-all VOICE-BEARER
       match protocol rtp
      !
      class-map match-all HTTP-TRAFFIC
       match protocol http
      !
      class-map match-all FTP-TRAFFIC
       match protocol ftp
      !
      ! Marking policy
      policy-map CLASSIFY-TRAFFIC
       class VOICE-SIGNALING
        set dscp cs3
       class VOICE-BEARER
        set dscp ef
       class HTTP-TRAFFIC
        set dscp af31
       class FTP-TRAFFIC
        set dscp af11
       class class-default
        set dscp default
      !
      ! Apply classification
      interface GigabitEthernet0/0/1
       service-policy input CLASSIFY-TRAFFIC
      !
      interface GigabitEthernet0/0/2
       service-policy input CLASSIFY-TRAFFIC
      !

  # Data center server
  dc-server:
    kind: linux
    image: ubuntu:20.04
    mgmt-ipv4: 172.20.20.12
    exec:
      - ip addr add 10.1.13.2/30 dev eth1
      - ip route add default via 10.1.13.1
```

```yaml
        - apt update && apt install -y iperf3 nginx
        - service nginx start
        - iperf3 -s -D

    # Voice phone simulation
    voice-phone:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.20
      exec:
        - ip addr add 192.168.10.10/24 dev eth1
        - ip route add default via 192.168.10.1
        - apk add --no-cache iperf3

    # Data workstation
    data-workstation:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.21
      exec:
        - ip addr add 192.168.20.10/24 dev eth1
        - ip route add default via 192.168.20.1
        - apk add --no-cache iperf3 curl

    # Internet simulation
    internet-sim:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.100
      exec:
        - ip addr add 203.0.113.2/30 dev eth1
        - ip route add default via 203.0.113.1
        - apk add --no-cache iperf3

  links:
    # Core network connections
    - endpoints: ["core-router:eth1", "branch-router:eth1"]
    - endpoints: ["core-router:eth2", "dc-server:eth1"]
    - endpoints: ["core-router:eth3", "internet-sim:eth1"]

    # Branch connections
    - endpoints: ["branch-router:eth2", "voice-phone:eth1"]
    - endpoints: ["branch-router:eth3", "data-workstation:eth1"]
```

Traffic Classification and Marking

Classification Methods

Layer 3 Classification

- **IP Precedence**: 3-bit field (0-7)
- **DSCP**: 6-bit field (0-63)
- **Source/Destination IP**: Address-based classification
- **Protocol**: TCP, UDP, ICMP, etc.

Layer 4 Classification

- **Source/Destination Port**: Application identification
- **TCP Flags**: Connection state information
- **Packet Size**: Large vs. small packets

Deep Packet Inspection

- **Application Recognition**: NBAR (Network-Based Application Recognition)
- **Protocol Analysis**: Application-specific patterns
- **Behavioral Analysis**: Traffic flow characteristics

DSCP Values and Classes

Standard DSCP Values

Class	DSCP Name	DSCP Value	Binary	Decimal	Usage
Default	Default	DF	000000	0	Best effort
Expedited Forwarding	EF	EF	101110	46	Voice
Assured Forwarding	AF41	AF41	100010	34	Video
Assured Forwarding	AF31	AF31	011010	26	Critical data
Assured Forwarding	AF21	AF21	010010	18	Standard data
Assured Forwarding	AF11	AF11	001010	10	Bulk data
Class Selector	CS6	CS6	110000	48	Network control
Class Selector	CS3	CS3	011000	24	Signaling

Classification Configuration

NBAR-Based Classification

```
# Deploy QoS lab
containerlab deploy -t qos-fundamentals.yml

# Configure NBAR on branch router
docker exec -it clab-qos-branch-router cli

configure terminal
! Enable NBAR
ip nbar port-map http tcp 8080
ip nbar port-map https tcp 8443

! Create class maps using NBAR
class-map match-all VOICE-RTP
 match protocol rtp audio
!
class-map match-all VIDEO-STREAMING
 match protocol rtsp
!
class-map match-all WEB-BROWSING
 match protocol http
 match protocol https
!
class-map match-all FILE-TRANSFER
 match protocol ftp
 match protocol sftp
!
class-map match-all EMAIL
 match protocol smtp
 match protocol pop3
 match protocol imap
!
```

ACL-Based Classification

```
# Create access lists for classification
ip access-list extended VOICE-TRAFFIC
 permit udp any any range 16384 32767
 permit tcp any any eq 5060
 permit tcp any any eq 5061

ip access-list extended VIDEO-TRAFFIC
```

```
 permit udp any any range 1024 65535 dscp af41
 permit tcp any any eq 554

ip access-list extended CRITICAL-DATA
 permit tcp any any eq 443
 permit tcp any any eq 993
 permit tcp any any eq 995

! Apply ACLs to class maps
class-map match-all VOICE-CLASS
 match access-group name VOICE-TRAFFIC
!
class-map match-all VIDEO-CLASS
 match access-group name VIDEO-TRAFFIC
!
class-map match-all CRITICAL-CLASS
 match access-group name CRITICAL-DATA
```

Marking Strategies

Trust Boundaries

```
# Configure trust boundaries
interface GigabitEthernet0/0/1
 description Trusted-Phone-Port
 mls qos trust dscp

interface GigabitEthernet0/0/2
 description Untrusted-PC-Port
 mls qos trust cos
 mls qos cos 0

! Conditional trust
mls qos map cos-dscp 0 8 16 24 32 46 48 56
```

Marking Policies

```
# Create marking policy
policy-map MARK-TRAFFIC
 class VOICE-CLASS
  set dscp ef
  set ip precedence 5
 class VIDEO-CLASS
  set dscp af41
 class CRITICAL-CLASS
  set dscp af31
 class class-default
  set dscp default

! Apply marking policy
interface GigabitEthernet0/0/1
 service-policy input MARK-TRAFFIC
```

Queuing Mechanisms

Queuing Algorithms

First In, First Out (FIFO)

- **Simple**: Single queue, no prioritization
- **Fair**: All packets treated equally
- **Limitations**: No QoS differentiation
- **Usage**: Default behavior without QoS

Priority Queuing (PQ)

- **Strict Priority**: High priority always served first
- **Starvation Risk**: Low priority may never be served
- **Usage**: Voice traffic in LLQ

Weighted Fair Queuing (WFQ)

- **Flow-based**: Separate queue per flow
- **Fairness**: Bandwidth allocated by flow weight
- **Automatic**: No configuration required
- **Limitations**: Not suitable for high-speed interfaces

Class-Based Weighted Fair Queuing (CBWFQ)

- **Class-based**: Queues based on traffic classes
- **Bandwidth Guarantees**: Minimum bandwidth per class
- **Scalable**: Suitable for high-speed interfaces
- **Flexible**: Configurable class definitions

Queuing Configuration

Basic CBWFQ Configuration

```
# Configure CBWFQ policy
policy-map CBWFQ-POLICY
 class VOICE-CLASS
  bandwidth 1000
 class VIDEO-CLASS
  bandwidth 3000
 class CRITICAL-CLASS
  bandwidth 2000
 class class-default
  bandwidth 1000
  fair-queue

! Apply to interface
interface GigabitEthernet0/0/0
 service-policy output CBWFQ-POLICY
```

Low Latency Queuing (LLQ)

```
# Configure LLQ for voice
policy-map LLQ-POLICY
 class VOICE-CLASS
  priority 1000
 class VIDEO-CLASS
  bandwidth 3000
 class CRITICAL-CLASS
  bandwidth 2000
 class class-default
  bandwidth remaining percent 20
  fair-queue

! Verify LLQ configuration
show policy-map interface GigabitEthernet0/0/0
```

Advanced Queuing Features

Bandwidth Allocation

```
# Percentage-based bandwidth allocation
policy-map PERCENT-POLICY
 class VOICE-CLASS
  priority percent 20
 class VIDEO-CLASS
  bandwidth percent 30
 class CRITICAL-CLASS
  bandwidth percent 25
 class class-default
  bandwidth percent 25

# Remaining bandwidth allocation
policy-map REMAINING-POLICY
 class VOICE-CLASS
  priority 1000
 class VIDEO-CLASS
  bandwidth remaining percent 40
 class CRITICAL-CLASS
  bandwidth remaining percent 35
 class class-default
  bandwidth remaining percent 25
```

Queue Limits and Drop Policies

```
# Configure queue limits
policy-map QUEUE-LIMITS
 class VOICE-CLASS
  priority 1000
  queue-limit 32
 class VIDEO-CLASS
  bandwidth 3000
  queue-limit 64
 class BULK-CLASS
  bandwidth 1000
  queue-limit 128
  random-detect
```

Congestion Avoidance

Random Early Detection (RED)

RED prevents global synchronization by randomly dropping packets before queues become full.

Weighted Random Early Detection (WRED)

```
# Configure WRED
policy-map WRED-POLICY
 class CRITICAL-CLASS
  bandwidth 2000
  random-detect dscp-based
  random-detect dscp af31 20 40 10
  random-detect dscp af32 15 35 10
  random-detect dscp af33 10 30 10
 class BULK-CLASS
  bandwidth 1000
  random-detect
  random-detect exponential-weighting-constant 9

! Verify WRED configuration
show policy-map interface GigabitEthernet0/0/0
show queueing interface GigabitEthernet0/0/0
```

WRED Parameters

- **Minimum Threshold**: Start dropping probability
- **Maximum Threshold**: 100% drop probability
- **Mark Probability Denominator**: Drop probability calculation
- **Exponential Weighting Constant**: Average queue depth calculation

Traffic Shaping and Policing

Traffic Shaping

Traffic shaping delays excess traffic to conform to a configured rate.

Generic Traffic Shaping (GTS)

```
# Configure traffic shaping
interface GigabitEthernet0/0/0
 traffic-shape rate 10000000 20000 20000
 # Rate: 10 Mbps, Burst: 20KB, Excess burst: 20KB

# Class-based shaping
policy-map SHAPE-POLICY
 class VIDEO-CLASS
  shape average 5000000
 class BULK-CLASS
  shape average 2000000 4000 4000
 class class-default
  shape average 1000000

interface GigabitEthernet0/0/0
 service-policy output SHAPE-POLICY
```

Hierarchical QoS

```
# Parent policy for shaping
policy-map PARENT-SHAPE
 class class-default
  shape average 10000000
  service-policy CHILD-QUEUE

# Child policy for queuing
policy-map CHILD-QUEUE
 class VOICE-CLASS
  priority percent 20
 class VIDEO-CLASS
  bandwidth percent 30
 class CRITICAL-CLASS
  bandwidth percent 25
 class class-default
  bandwidth percent 25

interface GigabitEthernet0/0/0
 service-policy output PARENT-SHAPE
```

Traffic Policing

Traffic policing drops or marks excess traffic that exceeds configured rates.

Single-Rate Policing

```
# Configure single-rate policer
policy-map POLICE-POLICY
 class BULK-CLASS
  police rate 2000000 burst 4000
   conform-action transmit
   exceed-action drop
 class class-default
  police rate 1000000 burst 2000
   conform-action transmit
   exceed-action set-dscp-transmit default

interface GigabitEthernet0/0/1
 service-policy input POLICE-POLICY
```

Dual-Rate Policing

```
# Configure dual-rate policer
policy-map DUAL-RATE-POLICE
 class CRITICAL-CLASS
  police cir 2000000 bc 4000 pir 4000000 be 8000
   conform-action transmit
   exceed-action set-dscp-transmit af32
   violate-action drop

interface GigabitEthernet0/0/1
 service-policy input DUAL-RATE-POLICE
```

QoS Monitoring and Troubleshooting

QoS Statistics

```
# Monitor QoS statistics
show policy-map interface GigabitEthernet0/0/0
show queueing interface GigabitEthernet0/0/0
show interfaces GigabitEthernet0/0/0 | include drops

# Detailed class statistics
show policy-map interface GigabitEthernet0/0/0 output class VOICE-CLASS
show policy-map interface GigabitEthernet0/0/0 input class BULK-CLASS
```

Traffic Analysis

```
# Analyze traffic patterns
show ip nbar protocol-discovery
show ip nbar protocol-discovery interface GigabitEthernet0/0/1

# Monitor DSCP distribution
show mls qos interface GigabitEthernet0/0/1 statistics
show ip cef switching-statistics

# Real-time monitoring
show interfaces GigabitEthernet0/0/0 | include rate
show policy-map interface GigabitEthernet0/0/0 | include offered
```

QoS Troubleshooting

Common QoS Issues

```
# 1. Classification problems
show class-map
show policy-map
show ip nbar protocol-discovery

# 2. Marking issues
show mls qos maps
show mls qos interface statistics

# 3. Queuing problems
show queueing interface GigabitEthernet0/0/0
show policy-map interface GigabitEthernet0/0/0

# 4. Bandwidth allocation
show interfaces GigabitEthernet0/0/0 | include BW
show policy-map interface GigabitEthernet0/0/0 | include bandwidth
```

Debug Commands

```
# Debug QoS (use carefully in production)
debug policy-map
debug qos set
debug mls qos

# Monitor specific classes
```

```
show policy-map interface GigabitEthernet0/0/0 output class VOICE-CLASS
```

QoS Design Best Practices

Voice QoS Requirements

```
# Voice QoS configuration
policy-map VOICE-OPTIMIZED
 class VOICE-BEARER
  priority percent 10
  set dscp ef
 class VOICE-SIGNALING
  bandwidth percent 5
  set dscp cs3
 class VIDEO-CLASS
  bandwidth percent 33
  set dscp af41
 class CRITICAL-DATA
  bandwidth percent 25
  set dscp af31
 class class-default
  bandwidth percent 27
  fair-queue
  random-detect dscp-based
```

Video QoS Requirements

```
# Video QoS configuration
policy-map VIDEO-OPTIMIZED
 class VOICE-CLASS
  priority percent 10
 class INTERACTIVE-VIDEO
  bandwidth percent 20
  set dscp af41
  queue-limit 64
 class STREAMING-VIDEO
  bandwidth percent 15
  set dscp af31
  random-detect dscp-based
 class class-default
  bandwidth percent 55
  fair-queue
```

Data QoS Requirements

```
# Data application QoS
policy-map DATA-OPTIMIZED
 class VOICE-CLASS
  priority percent 10
 class MISSION-CRITICAL
  bandwidth percent 25
  set dscp af31
 class TRANSACTIONAL-DATA
  bandwidth percent 20
  set dscp af21
 class BULK-DATA
  bandwidth percent 15
  set dscp af11
  random-detect
 class class-default
  bandwidth percent 30
  fair-queue
```

Testing QoS Implementation

Traffic Generation

```
# Generate test traffic from data workstation
docker exec -it clab-qos-data-workstation sh

# High-priority traffic test
iperf3 -c 10.1.13.2 -t 60 -b 5M --dscp 46

# Video traffic test
iperf3 -c 10.1.13.2 -t 60 -b 10M --dscp 34

# Bulk data test
iperf3 -c 10.1.13.2 -t 60 -b 20M --dscp 10

# Monitor QoS statistics during tests
docker exec -it clab-qos-core-router cli -c "show policy-map interface GigabitEthernet0/0/0"
```

Performance Validation

```
# Validate QoS performance
show policy-map interface GigabitEthernet0/0/0 | include offered|drop
show interfaces GigabitEthernet0/0/0 | include drops|rate

# Check queue depths
show queueing interface GigabitEthernet0/0/0

# Verify DSCP marking
show ip nbar protocol-discovery interface GigabitEthernet0/0/1 stats packet-count
```

Summary

Quality of Service is essential for managing network resources and ensuring application performance in modern networks. Understanding traffic characteristics, classification methods, queuing algorithms, and congestion management techniques enables effective QoS implementation for voice, video, and data applications.

Key concepts covered: - QoS fundamentals and service models - Traffic classification and DSCP marking - Queuing mechanisms (FIFO, PQ, WFQ, CBWFQ, LLQ) - Congestion avoidance with WRED - Traffic shaping and policing - QoS monitoring and troubleshooting

In the next chapter, we'll explore advanced QoS implementation including Modular QoS CLI (MQC) and complex policy configurations.

Review Questions

1. What are the differences between IntServ and DiffServ QoS models?
2. How do you classify traffic using NBAR and ACLs?
3. What are the characteristics of voice, video, and data traffic?
4. How does Low Latency Queuing (LLQ) work?
5. What's the difference between traffic shaping and policing?

Hands-on Exercises

Exercise 1: Basic QoS Implementation

1. Deploy the QoS fundamentals lab
2. Configure traffic classification and marking
3. Implement basic queuing policies
4. Test with traffic generation tools

Exercise 2: Advanced Queuing Configuration

1. Configure LLQ for voice traffic
2. Implement CBWFQ for different traffic classes
3. Configure WRED for congestion avoidance
4. Monitor queue statistics and performance

Exercise 3: Traffic Shaping and Policing

1. Configure traffic shaping on WAN interfaces
2. Implement traffic policing for rate limiting
3. Create hierarchical QoS policies
4. Test bandwidth allocation and limiting

Exercise 4: QoS Troubleshooting

1. Create QoS misconfigurations and issues
2. Practice diagnostic commands and procedures
3. Analyze traffic patterns and performance
4. Optimize QoS policies based on requirements

Additional Resources

- Cisco QoS Configuration Guide
- QoS Design Guide
- Voice QoS Best Practices
- DSCP and QoS Marking

Chapter 44: MPLS Fundamentals

Learning Objectives

By the end of this chapter, you will be able to: - Understand MPLS concepts and architecture - Configure MPLS label switching and LDP - Implement MPLS VPN services - Configure MPLS Traffic Engineering basics - Troubleshoot MPLS networks and services

MPLS Fundamentals

What is MPLS?

Multiprotocol Label Switching (MPLS) is a routing technique that forwards packets based on labels rather than network addresses. MPLS creates a virtual circuit-like service over packet-switched networks, enabling efficient traffic engineering and VPN services.

Key MPLS Benefits

1. **Performance**: Faster forwarding decisions
2. **Traffic Engineering**: Explicit path control
3. **VPN Services**: Layer 2 and Layer 3 VPNs
4. **Quality of Service**: Integrated QoS support
5. **Scalability**: Efficient core network operation

MPLS Architecture

MPLS Components

- **Label Switch Router (LSR)**: Forwards packets based on labels
- **Label Edge Router (LER)**: Ingress/egress points for MPLS domain
- **Label**: 32-bit identifier for forwarding decisions
- **Label Distribution Protocol (LDP)**: Distributes label bindings
- **Forwarding Equivalence Class (FEC)**: Group of packets with same treatment

MPLS Label Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Label                  | TC  |S|       TTL     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Label: 20 bits - Label value
TC: 3 bits - Traffic Class (QoS)
S: 1 bit - Bottom of Stack
TTL: 8 bits - Time to Live
```

MPLS Operations

Label Operations

1. **PUSH**: Add label to packet (ingress LER)
2. **SWAP**: Replace label with new label (LSR)
3. **POP**: Remove label from packet (egress LER or penultimate LSR)

Penultimate Hop Popping (PHP)

The second-to-last router removes the label, reducing processing at the egress LER.

MPLS Lab Environment

Basic MPLS Lab Setup

```yaml
# MPLS fundamentals lab
name: mpls-fundamentals
prefix: mpls

topology:
  nodes:
    # Provider Edge Router 1
    pe1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.10
      startup-config: |
        hostname PE1
        !
        ip cef
        mpls ip
        !
        interface Loopback0
         ip address 1.1.1.1 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description To-P1
         ip address 10.1.12.1 255.255.255.252
         mpls ip
         no shutdown
        !
        interface GigabitEthernet0/0/1
         description To-CE1
         ip address 192.168.1.1 255.255.255.252
         no shutdown
        !
        ! OSPF for IGP
        router ospf 1
         router-id 1.1.1.1
         network 1.1.1.1 0.0.0.0 area 0
         network 10.1.12.0 0.0.0.3 area 0
        !
        ! LDP configuration
        mpls ldp router-id Loopback0
        mpls ldp discovery hello interval 5
        mpls ldp discovery hello holdtime 15
        !

    # Provider Router 1
    p1:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname P1
        !
        ip cef
        mpls ip
        !
        interface Loopback0
         ip address 2.2.2.2 255.255.255.255
        !
        interface GigabitEthernet0/0/0
         description To-PE1
         ip address 10.1.12.2 255.255.255.252
         mpls ip
         no shutdown
```

```
      !
      interface GigabitEthernet0/0/1
       description To-P2
       ip address 10.1.23.2 255.255.255.252
       mpls ip
       no shutdown
      !
      ! OSPF for IGP
      router ospf 1
       router-id 2.2.2.2
       network 2.2.2.2 0.0.0.0 area 0
       network 10.1.12.0 0.0.0.3 area 0
       network 10.1.23.0 0.0.0.3 area 0
      !
      ! LDP configuration
      mpls ldp router-id Loopback0
      !

  # Provider Router 2
  p2:
    kind: cisco_iosxe
    image: cisco/iosxe:latest
    mgmt-ipv4: 172.20.20.12
    startup-config: |
      hostname P2
      !
      ip cef
      mpls ip
      !
      interface Loopback0
       ip address 3.3.3.3 255.255.255.255
      !
      interface GigabitEthernet0/0/0
       description To-P1
       ip address 10.1.23.3 255.255.255.252
       mpls ip
       no shutdown
      !
      interface GigabitEthernet0/0/1
       description To-PE2
       ip address 10.1.34.3 255.255.255.252
       mpls ip
       no shutdown
      !
      ! OSPF for IGP
      router ospf 1
       router-id 3.3.3.3
      network 3.3.3.3 0.0.0.0 area 0
      network 10.1.23.0 0.0.0.3 area 0
      network 10.1.34.0 0.0.0.3 area 0
     !
     ! LDP configuration
     mpls ldp router-id Loopback0
     !

# Provider Edge Router 2
pe2:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
  mgmt-ipv4: 172.20.20.13
  startup-config: |
    hostname PE2
    !
    ip cef
    mpls ip
    !
    interface Loopback0
     ip address 4.4.4.4 255.255.255.255
    !
    interface GigabitEthernet0/0/0
     description To-P2
     ip address 10.1.34.4 255.255.255.252
     mpls ip
     no shutdown
    !
    interface GigabitEthernet0/0/1
     description To-CE2
     ip address 192.168.2.1 255.255.255.252
     no shutdown
    !
    ! OSPF for IGP
    router ospf 1
     router-id 4.4.4.4
     network 4.4.4.4 0.0.0.0 area 0
     network 10.1.34.0 0.0.0.3 area 0
    !
    ! LDP configuration
    mpls ldp router-id Loopback0
    !

# Customer Edge Router 1
ce1:
  kind: cisco_iosxe
  image: cisco/iosxe:latest
```

```
    mgmt-ipv4: 172.20.20.20
    startup-config: |
      hostname CE1
      !
      interface Loopback0
       ip address 10.10.10.10 255.255.255.255
      !
      interface GigabitEthernet0/0/0
       description To-PE1
       ip address 192.168.1.2 255.255.255.252
       no shutdown
      !
      interface GigabitEthernet0/0/1
       description Customer-LAN
       ip address 172.16.1.1 255.255.255.0
       no shutdown
      !
      ! Static routing to provider
      ip route 0.0.0.0 0.0.0.0 192.168.1.1
      !

  # Customer Edge Router 2
  ce2:
    kind: cisco_iosxe
    image: cisco/iosxe:latest
    mgmt-ipv4: 172.20.20.21
    startup-config: |
      hostname CE2
      !
      interface Loopback0
       ip address 20.20.20.20 255.255.255.255
      !
      interface GigabitEthernet0/0/0
       description To-PE2
       ip address 192.168.2.2 255.255.255.252
       no shutdown
      !
      interface GigabitEthernet0/0/1
       description Customer-LAN
       ip address 172.16.2.1 255.255.255.0
       no shutdown
      !
      ! Static routing to provider
      ip route 0.0.0.0 0.0.0.0 192.168.2.1
      !

  # Customer devices
```

```
    customer1-pc:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 172.16.1.10/24 dev eth1
        - ip route add default via 172.16.1.1

    customer2-pc:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 172.16.2.10/24 dev eth1
        - ip route add default via 172.16.2.1

  links:
    # MPLS core network
    - endpoints: ["pe1:eth1", "p1:eth1"]
    - endpoints: ["p1:eth2", "p2:eth1"]
    - endpoints: ["p2:eth2", "pe2:eth1"]

    # Customer connections
    - endpoints: ["pe1:eth2", "ce1:eth1"]
    - endpoints: ["pe2:eth2", "ce2:eth1"]

    # Customer LANs
    - endpoints: ["ce1:eth2", "customer1-pc:eth1"]
    - endpoints: ["ce2:eth2", "customer2-pc:eth1"]
```

Label Distribution Protocol (LDP)

LDP Fundamentals

LDP is used to distribute label bindings between LSRs in an MPLS network.

LDP Messages

1. **Discovery Messages**: Hello packets for neighbor discovery
2. **Session Messages**: Establish and maintain LDP sessions
3. **Advertisement Messages**: Advertise label bindings
4. **Notification Messages**: Error and advisory information

LDP Configuration

Basic LDP Setup

```
# Deploy MPLS lab
containerlab deploy -t mpls-fundamentals.yml

# Configure LDP on PE1
docker exec -it clab-mpls-pe1 cli

configure terminal
! Enable MPLS globally
mpls ip

! Configure LDP router ID
mpls ldp router-id Loopback0

! Enable LDP on interfaces
interface GigabitEthernet0/0/0
 mpls ip

! Verify LDP neighbors
show mpls ldp neighbor
show mpls ldp discovery
```

Advanced LDP Configuration

```
# Configure LDP parameters
mpls ldp discovery hello interval 10
mpls ldp discovery hello holdtime 30
mpls ldp session holdtime 180
mpls ldp password required for 10 PASSWORD123

! LDP access control
access-list 10 permit 1.1.1.1
access-list 10 permit 2.2.2.2
access-list 10 permit 3.3.3.3
access-list 10 permit 4.4.4.4

! Targeted LDP sessions
mpls ldp neighbor 4.4.4.4 targeted ldp
```

LDP Verification

```
# Verify LDP operation
show mpls ldp neighbor
show mpls ldp discovery
show mpls ldp bindings
show mpls forwarding-table

# Detailed LDP information
show mpls ldp neighbor detail
show mpls ldp parameters
show mpls ldp statistics
```

MPLS Forwarding

Label Forwarding Information Base (LFIB)

The LFIB contains label forwarding entries used for MPLS packet forwarding.

```
# View MPLS forwarding table
show mpls forwarding-table
show mpls forwarding-table detail
show mpls forwarding-table 4.4.4.4/32

# Example LFIB entry:
# Local  Outgoing    Prefix            Bytes Label   Outgoing   Next Hop
# Label  Label       or Tunnel Id      Switched      interface
# 16     Pop Label   4.4.4.4/32        0             Gi0/0/0    10.1.12.2
```

Label Stack Operations

```
# Trace MPLS path
traceroute 4.4.4.4 source 1.1.1.1

# Monitor label operations
debug mpls packet
debug mpls ldp messages

# Verify label stack
show mpls forwarding-table 4.4.4.4/32 detail
```

MPLS VPN Services

Layer 3 MPLS VPN

Layer 3 MPLS VPNs provide IP connectivity between customer sites using VRFs (Virtual Routing and Forwarding).

VRF Configuration

```
# Configure VRF on PE1
configure terminal
ip vrf CUSTOMER-A
 rd 100:1
 route-target export 100:1
 route-target import 100:1

! Assign interface to VRF
interface GigabitEthernet0/0/1
 ip vrf forwarding CUSTOMER-A
 ip address 192.168.1.1 255.255.255.252

! BGP for VPNv4
router bgp 100
 bgp router-id 1.1.1.1
 no bgp default ipv4-unicast
 neighbor 4.4.4.4 remote-as 100
 neighbor 4.4.4.4 update-source Loopback0
 !
 address-family vpnv4
  neighbor 4.4.4.4 activate
  neighbor 4.4.4.4 send-community extended
 exit-address-family
 !
 address-family ipv4 vrf CUSTOMER-A
  redistribute connected
  neighbor 192.168.1.2 remote-as 65001
  neighbor 192.168.1.2 activate
 exit-address-family
```

MP-BGP Configuration

```
# Configure MP-BGP for VPNv4
router bgp 100
 bgp router-id 1.1.1.1
 no bgp default ipv4-unicast

 ! iBGP neighbor for VPNv4
 neighbor 4.4.4.4 remote-as 100
 neighbor 4.4.4.4 update-source Loopback0

 address-family vpnv4
  neighbor 4.4.4.4 activate
  neighbor 4.4.4.4 send-community extended
  neighbor 4.4.4.4 next-hop-self
 exit-address-family

# Verify MP-BGP
show bgp vpnv4 unicast all
show bgp vpnv4 unicast vrf CUSTOMER-A
```

Layer 2 MPLS VPN

Layer 2 MPLS VPNs provide Ethernet connectivity between customer sites.

Pseudowire Configuration

```
# Configure L2VPN pseudowire
l2vpn vfi context CUSTOMER-A
 vpn id 100
 member 4.4.4.4 encapsulation mpls

interface GigabitEthernet0/0/1
 switchport
 switchport mode access
 switchport access vlan 100

interface vlan 100
 xconnect vfi CUSTOMER-A
```

MPLS Traffic Engineering

Traffic Engineering Fundamentals

MPLS TE allows explicit path control and bandwidth reservation.

RSVP-TE Configuration

```
# Enable RSVP-TE
ip rsvp bandwidth 10000 10000

interface GigabitEthernet0/0/0
 ip rsvp bandwidth 10000 10000
 mpls traffic-eng tunnels

! Configure TE tunnel
interface Tunnel1
 ip unnumbered Loopback0
 tunnel destination 4.4.4.4
 tunnel mode mpls traffic-eng
 tunnel mpls traffic-eng bandwidth 5000
 tunnel mpls traffic-eng path-option 1 explicit name PATH-TO-PE2

! Define explicit path
ip explicit-path name PATH-TO-PE2 enable
 next-address 10.1.12.2
 next-address 10.1.23.3
 next-address 10.1.34.4
```

IS-IS for MPLS TE

```
# Configure IS-IS with TE extensions
router isis
 net 49.0001.0000.0000.0001.00
 is-type level-2-only
 metric-style wide
 mpls traffic-eng router-id Loopback0
 mpls traffic-eng level-2

interface GigabitEthernet0/0/0
 ip router isis
 isis metric 10
 mpls traffic-eng tunnels
```

MPLS QoS

QoS in MPLS Networks

MPLS supports QoS through the EXP (Experimental) bits in the label header.

EXP Bit Mapping

```
# Configure EXP bit mapping
mls qos map dscp-exp 0 8 16 24 32 40 48 56 to 0 1 2 3 4 5 6 7

! QoS policy for MPLS
policy-map MPLS-QOS
 class VOICE
  set mpls experimental topmost 5
 class VIDEO
  set mpls experimental topmost 4
 class DATA
  set mpls experimental topmost 2

interface GigabitEthernet0/0/0
 service-policy output MPLS-QOS
```

Pipe and Uniform Models

```
# Pipe model (preserve customer QoS)
policy-map PIPE-MODEL
 class class-default
  set mpls experimental topmost 0

# Uniform model (copy IP DSCP to EXP)
policy-map UNIFORM-MODEL
 class class-default
  set mpls experimental imposition dscp table
```

MPLS Troubleshooting

Common MPLS Issues

LDP Neighbor Problems

```
# Troubleshoot LDP neighbors
show mpls ldp neighbor
show mpls ldp discovery
show mpls ldp parameters

# Common issues:
# 1. LDP not enabled on interface
interface GigabitEthernet0/0/0
 mpls ip

# 2. Router ID not configured
mpls ldp router-id Loopback0

# 3. IGP connectivity issues
show ip route 4.4.4.4
ping 4.4.4.4 source 1.1.1.1
```

Label Binding Issues

```
# Check label bindings
show mpls ldp bindings
show mpls ldp bindings 4.4.4.4/32
show mpls forwarding-table

# Debug LDP
debug mpls ldp messages
debug mpls ldp bindings
```

MPLS VPN Troubleshooting

```
# VRF troubleshooting
show ip vrf
show ip vrf detail CUSTOMER-A
show ip route vrf CUSTOMER-A

# BGP VPNv4 troubleshooting
show bgp vpnv4 unicast all summary
show bgp vpnv4 unicast vrf CUSTOMER-A
show bgp vpnv4 unicast all neighbors

# Connectivity testing
ping vrf CUSTOMER-A 172.16.2.10 source 172.16.1.10
traceroute vrf CUSTOMER-A 172.16.2.10
```

MPLS Monitoring

```
# Monitor MPLS performance
show mpls forwarding-table
show mpls ldp statistics
show mpls traffic-eng tunnels

# Interface statistics
show interfaces GigabitEthernet0/0/0 | include MPLS
show mpls interfaces
show mpls interfaces detail
```

MPLS Security

MPLS Security Considerations

```
# LDP authentication
mpls ldp password required for 10 SecureLDPKey123
access-list 10 permit 1.1.1.1
access-list 10 permit 2.2.2.2
access-list 10 permit 3.3.3.3
access-list 10 permit 4.4.4.4

# BGP authentication for VPNv4
router bgp 100
 neighbor 4.4.4.4 password SecureBGPKey456

# Control plane protection
control-plane
 service-policy input CONTROL-PLANE-POLICY
```

MPLS Best Practices

```
# MPLS security hardening
! Disable MPLS on customer-facing interfaces
interface GigabitEthernet0/0/1
 no mpls ip

! Filter MPLS packets from customers
access-list 101 deny ip any any precedence 6
access-list 101 deny ip any any precedence 7
access-list 101 permit ip any any

interface GigabitEthernet0/0/1
 ip access-group 101 in

! TTL propagation control
no mpls ip propagate-ttl
```

Testing MPLS Connectivity

End-to-End Testing

```
# Test customer connectivity
docker exec -it clab-mpls-customer1-pc sh
ping 172.16.2.10

# Trace MPLS path
docker exec -it clab-mpls-ce1 cli
traceroute 20.20.20.20 source 10.10.10.10

# Monitor MPLS labels
docker exec -it clab-mpls-pe1 cli
show mpls forwarding-table 4.4.4.4/32
debug mpls packet
```

Performance Testing

```
# Bandwidth testing through MPLS
docker exec -it clab-mpls-customer1-pc sh
iperf3 -s &

docker exec -it clab-mpls-customer2-pc sh
iperf3 -c 172.16.1.10 -t 60
```

```
# Monitor MPLS statistics
docker exec -it clab-mpls-p1 cli
show mpls forwarding-table | include bytes
show interfaces GigabitEthernet0/0/0 | include rate
```

Summary

MPLS provides a powerful framework for service provider networks, enabling efficient packet forwarding, traffic engineering, and VPN services. Understanding MPLS fundamentals, LDP operation, and VPN implementation is essential for modern service provider and enterprise networks.

Key concepts covered: - MPLS architecture and label operations - Label Distribution Protocol (LDP) configuration - MPLS forwarding and label switching - Layer 3 and Layer 2 MPLS VPN services - MPLS Traffic Engineering basics - MPLS QoS and security considerations

In the next chapter, we'll explore advanced service provider technologies including advanced MPLS VPN features and service implementations.

Review Questions

1. How does MPLS forwarding differ from traditional IP routing?
2. What is the role of LDP in MPLS networks?
3. How do Layer 3 MPLS VPNs provide customer isolation?
4. What are the benefits of MPLS Traffic Engineering?
5. How do you troubleshoot MPLS connectivity issues?

Hands-on Exercises

Exercise 1: Basic MPLS Configuration

1. Deploy the MPLS fundamentals lab
2. Configure LDP on all provider routers
3. Verify label distribution and forwarding
4. Test end-to-end connectivity

Exercise 2: MPLS VPN Implementation

1. Configure VRFs on PE routers
2. Implement MP-BGP for VPNv4 routes
3. Configure customer routing protocols
4. Test VPN isolation and connectivity

Exercise 3: MPLS Traffic Engineering

1. Configure RSVP-TE tunnels
2. Implement explicit path constraints
3. Test traffic engineering functionality
4. Monitor bandwidth utilization

Exercise 4: MPLS Troubleshooting

1. Create various MPLS problems (LDP issues, VPN misconfigurations)
2. Practice diagnostic commands and procedures
3. Develop systematic troubleshooting approaches
4. Document solutions and prevention strategies

Additional Resources

- MPLS Configuration Guide
- MPLS VPN Configuration Guide
- MPLS Traffic Engineering Guide
- MPLS Best Practices

Chapter 48: Advanced Network Automation

Learning Objectives

By the end of this chapter, you will be able to: - Implement advanced Ansible automation for network infrastructure - Develop Python scripts for network management and monitoring - Configure NETCONF and RESTCONF for programmable networks - Build CI/CD pipelines for network configuration management - Create automated testing frameworks for network validation

Advanced Ansible for Network Automation

Ansible Architecture for Networks

Ansible provides agentless automation for network devices using SSH, NETCONF, or API connections.

Ansible Components

- **Control Node**: Machine running Ansible
- **Managed Nodes**: Network devices being automated
- **Inventory**: List of managed devices
- **Playbooks**: Automation scripts in YAML
- **Modules**: Task execution units
- **Plugins**: Extend Ansible functionality

Advanced Ansible Lab Setup

```yaml
# Advanced network automation lab
name: network-automation
prefix: auto

topology:
  nodes:
    # Ansible control node
    ansible-controller:
      kind: linux
      image: ubuntu:20.04
      mgmt-ipv4: 172.20.20.10
      exec:
        - apt update && apt install -y python3 python3-pip openssh-client git
        - pip3 install ansible netmiko napalm jinja2 paramiko
        - mkdir -p /etc/ansible /opt/automation
        - echo "[defaults]" > /etc/ansible/ansible.cfg
        - echo "host_key_checking = False" >> /etc/ansible/ansible.cfg
        - echo "timeout = 30" >> /etc/ansible/ansible.cfg

    # Network devices for automation
    core-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      mgmt-ipv4: 172.20.20.11
      startup-config: |
        hostname Core-Router
        !
        username ansible privilege 15 secret ansible123
        !
        ip domain-name automation.lab
        crypto key generate rsa modulus 2048
        !
        line vty 0 15
         login local
         transport input ssh
        !
        interface GigabitEthernet0/0/0
         description To-Distribution
         ip address 10.1.12.1 255.255.255.252
         no shutdown
        !
        interface Loopback0
         ip address 1.1.1.1 255.255.255.255
        !

    dist-switch:
      kind: cisco_iosxe
      image: cisco/catalyst:latest
      mgmt-ipv4: 172.20.20.12
      startup-config: |
        hostname Dist-Switch
        !
        username ansible privilege 15 secret ansible123
        !
        ip domain-name automation.lab
        crypto key generate rsa modulus 2048
        !
        line vty 0 15
```

```
        login local
        transport input ssh
       !
       interface GigabitEthernet1/0/1
        description To-Core
        switchport mode trunk
        no shutdown
       !
       interface Loopback0
        ip address 2.2.2.2 255.255.255.255
       !

   access-switch:
     kind: cisco_iosxe
     image: cisco/catalyst:latest
     mgmt-ipv4: 172.20.20.13
     startup-config: |
       hostname Access-Switch
       !
       username ansible privilege 15 secret ansible123
       !
       ip domain-name automation.lab
       crypto key generate rsa modulus 2048
       !
       line vty 0 15
        login local
        transport input ssh
       !
       interface GigabitEthernet1/0/1
        description To-Distribution
        switchport mode trunk
        no shutdown
       !
       interface Loopback0
        ip address 3.3.3.3 255.255.255.255
       !

links:
  - endpoints: ["core-router:eth1", "dist-switch:eth1"]
  - endpoints: ["dist-switch:eth2", "access-switch:eth1"]
```

Ansible Inventory Management

Dynamic Inventory

```python
#!/usr/bin/env python3
# dynamic_inventory.py
import json
import sys

def get_inventory():
    inventory = {
        'all': {
            'hosts': ['172.20.20.11', '172.20.20.12', '172.20.20.13'],
            'vars': {
                'ansible_user': 'ansible',
                'ansible_password': 'ansible123',
                'ansible_network_os': 'ios',
                'ansible_connection': 'network_cli'
            }
        },
        'routers': {
            'hosts': ['172.20.20.11'],
            'vars': {
                'device_type': 'router'
            }
        },
        'switches': {
            'hosts': ['172.20.20.12', '172.20.20.13'],
            'vars': {
                'device_type': 'switch'
            }
        },
        '_meta': {
            'hostvars': {
                '172.20.20.11': {
                    'hostname': 'Core-Router',
                    'device_role': 'core'
                },
                '172.20.20.12': {
                    'hostname': 'Dist-Switch',
                    'device_role': 'distribution'
                },
                '172.20.20.13': {
                    'hostname': 'Access-Switch',
                    'device_role': 'access'
                }
            }
        }
    }
    return inventory

if __name__ == '__main__':
    if len(sys.argv) == 2 and sys.argv[1] == '--list':
        print(json.dumps(get_inventory(), indent=2))
    elif len(sys.argv) == 3 and sys.argv[1] == '--host':
        print(json.dumps({}))
    else:
        print("Usage: %s --list or %s --host <hostname>" % (sys.argv[0], sys.argv[0]))
```

Static Inventory with Groups

```
# inventory/hosts.ini
[all:vars]
ansible_user=ansible
ansible_password=ansible123
ansible_network_os=ios
ansible_connection=network_cli

[routers]
core-router ansible_host=172.20.20.11 hostname=Core-Router

[switches]
dist-switch ansible_host=172.20.20.12 hostname=Dist-Switch
access-switch ansible_host=172.20.20.13 hostname=Access-Switch

[core]
core-router

[distribution]
dist-switch

[access]
access-switch
```

Advanced Ansible Playbooks

Configuration Management Playbook

```
# playbooks/configure_network.yml
---
- name: Configure Network Infrastructure
  hosts: all
  gather_facts: no
  vars:
    backup_dir: "./backups/{{ inventory_hostname }}"

  tasks:
    - name: Create backup directory
      file:
        path: "{{ backup_dir }}"
        state: directory
      delegate_to: localhost
      run_once: true

    - name: Backup current configuration
      ios_config:
        backup: yes
        backup_options:
          filename: "{{ inventory_hostname }}_{{ ansible_date_time.epoch }}.cfg"
          dir_path: "{{ backup_dir }}"

    - name: Configure global settings
      ios_config:
        lines:
          - service timestamps debug datetime msec
          - service timestamps log datetime msec
          - service password-encryption
          - no ip domain-lookup
          - ip domain-name {{ domain_name | default('automation.lab') }}
        save_when: modified

    - name: Configure NTP
      ios_config:
        lines:
          - ntp server {{ ntp_server | default('pool.ntp.org') }}
        save_when: modified

    - name: Configure SNMP
      ios_config:
        lines:
          - snmp-server community {{ snmp_community | default('public') }} RO
          - snmp-server location {{ snmp_location | default('Data Center') }}
          - snmp-server contact {{ snmp_contact | default('admin@company.com') }}
        save_when: modified

    - name: Configure logging
      ios_config:
        lines:
          - logging buffered 16384 informational
          - logging console critical
          - logging monitor informational
          - logging {{ syslog_server | default('172.20.20.100') }}
        save_when: modified

    - name: Verify configuration
      ios_command:
        commands:
          - show running-config | include ntp
          - show running-config | include snmp
          - show running-config | include logging
      register: config_verification

    - name: Display verification results
      debug:
        var: config_verification.stdout_lines
```

VLAN Management Playbook

```
# playbooks/manage_vlans.yml
---
- name: Manage VLANs on Switches
  hosts: switches
  gather_facts: no
  vars:
    vlans:
      - { id: 10, name: "USERS" }
      - { id: 20, name: "SERVERS" }
      - { id: 30, name: "MANAGEMENT" }
      - { id: 99, name: "NATIVE" }

  tasks:
    - name: Configure VLANs
      ios_vlans:
        config:
          - vlan_id: "{{ item.id }}"
          name: "{{ item.name }}"
          state: active
      state: merged
    loop: "{{ vlans }}"

  - name: Configure trunk interfaces
    ios_l2_interfaces:
      config:
        - name: "{{ item }}"
          trunk:
            allowed_vlans: "10,20,30,99"
            native_vlan: 99
      state: merged
    loop:
      - GigabitEthernet1/0/1
      - GigabitEthernet1/0/2
    when: inventory_hostname in groups['switches']

  - name: Configure access interfaces
    ios_l2_interfaces:
      config:
        - name: "GigabitEthernet1/0/{{ item.port }}"
          access:
            vlan: "{{ item.vlan }}"
      state: merged
    loop:
      - { port: 3, vlan: 10 }
      - { port: 4, vlan: 10 }
      - { port: 5, vlan: 20 }
      - { port: 6, vlan: 20 }
    when: inventory_hostname in groups['access']

  - name: Verify VLAN configuration
    ios_command:
      commands:
        - show vlan brief
        - show interfaces trunk
    register: vlan_status

  - name: Save VLAN status to file
    copy:
      content: "{{ vlan_status.stdout[0] }}"
      dest: "./reports/{{ inventory_hostname }}_vlans.txt"
    delegate_to: localhost
```

Jinja2 Templates for Configuration

Router Configuration Template

```
{# templates/router_config.j2 #}
hostname {{ hostname }}
!
{% if domain_name is defined %}
ip domain-name {{ domain_name }}
{% endif %}
!
{% for interface in interfaces %}
interface {{ interface.name }}
{% if interface.description is defined %}
 description {{ interface.description }}
{% endif %}
{% if interface.ip_address is defined %}
 ip address {{ interface.ip_address }} {{ interface.subnet_mask }}
{% endif %}
{% if interface.shutdown is not defined or not interface.shutdown %}
 no shutdown
{% endif %}
!
{% endfor %}
!
{% if ospf is defined %}
router ospf {{ ospf.process_id }}
 router-id {{ ospf.router_id }}
{% for network in ospf.networks %}
 network {{ network.network }} {{ network.wildcard }} area {{ network.area }}
{% endfor %}
!
{% endif %}
!
{% if static_routes is defined %}
{% for route in static_routes %}
ip route {{ route.network }} {{ route.mask }} {{ route.next_hop }}
{% endfor %}
{% endif %}
```

Switch Configuration Template

```
{# templates/switch_config.j2 #}
hostname {{ hostname }}
!
{% if vlans is defined %}
{% for vlan in vlans %}
vlan {{ vlan.id }}
 name {{ vlan.name }}
!
{% endfor %}
{% endif %}
!
{% if interfaces is defined %}
{% for interface in interfaces %}
interface {{ interface.name }}
{% if interface.description is defined %}
 description {{ interface.description }}
{% endif %}
{% if interface.mode == 'access' %}
 switchport mode access
 switchport access vlan {{ interface.vlan }}
{% elif interface.mode == 'trunk' %}
 switchport mode trunk
{% if interface.allowed_vlans is defined %}
 switchport trunk allowed vlan {{ interface.allowed_vlans }}
{% endif %}
{% if interface.native_vlan is defined %}
 switchport trunk native vlan {{ interface.native_vlan }}
{% endif %}
{% endif %}
{% if interface.portfast is defined and interface.portfast %}
 spanning-tree portfast
{% endif %}
 no shutdown
!
{% endfor %}
{% endif %}
```

Python Network Automation

Advanced Python Scripting

Network Device Management Class

```
#!/usr/bin/env python3
# network_manager.py
import netmiko
import json
import logging
from datetime import datetime
from concurrent.futures import ThreadPoolExecutor, as_completed

class NetworkManager:
    def __init__(self, devices_file):
        self.devices = self.load_devices(devices_file)
        self.setup_logging()

    def load_devices(self, devices_file):
        with open(devices_file, 'r') as f:
            return json.load(f)

    def setup_logging(self):
        logging.basicConfig(
            level=logging.INFO,
            format='%(asctime)s - %(levelname)s - %(message)s',
            handlers=[
                logging.FileHandler('network_automation.log'),
                logging.StreamHandler()
            ]
        )
        self.logger = logging.getLogger(__name__)

    def connect_device(self, device):
        try:
            connection = netmiko.ConnectHandler(**device)
            self.logger.info(f"Connected to {device['host']}")
            return connection
        except Exception as e:
            self.logger.error(f"Failed to connect to {device['host']}: {str(e)}")
            return None

    def execute_command(self, device, command):
        connection = self.connect_device(device)
        if connection:
            try:
                output = connection.send_command(command)
                connection.disconnect()
                return {
                    'host': device['host'],
                    'command': command,
                    'output': output,
                    'status': 'success'
                }
            except Exception as e:
                connection.disconnect()
                return {
                    'host': device['host'],
                    'command': command,
                    'error': str(e),
                    'status': 'failed'
                }
        return {
            'host': device['host'],
            'command': command,
            'error': 'Connection failed',
            'status': 'failed'
        }

    def execute_commands_parallel(self, commands, max_workers=5):
        results = []
        with ThreadPoolExecutor(max_workers=max_workers) as executor:
            futures = []
            for device in self.devices:
                for command in commands:
                    future = executor.submit(self.execute_command, device, command)
                    futures.append(future)

            for future in as_completed(futures):
                results.append(future.result())

        return results

    def backup_configurations(self, backup_dir='./backups'):
        import os
        os.makedirs(backup_dir, exist_ok=True)

        results = []
        for device in self.devices:
            connection = self.connect_device(device)
            if connection:
                try:
```

```
                config = connection.send_command('show running-config')
                timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
                filename = f"{backup_dir}/{device['host']}_{timestamp}.cfg"

                with open(filename, 'w') as f:
                    f.write(config)

                results.append({
                    'host': device['host'],
                    'backup_file': filename,
                    'status': 'success'
                })
                connection.disconnect()
            except Exception as e:
                results.append({
                    'host': device['host'],
                    'error': str(e),
                    'status': 'failed'
                })
                connection.disconnect()

    return results

def deploy_configuration(self, device, config_lines):
    connection = self.connect_device(device)
    if connection:
        try:
            output = connection.send_config_set(config_lines)
            connection.save_config()
            connection.disconnect()
            return {
                'host': device['host'],
                'output': output,
                'status': 'success'
            }
        except Exception as e:
            connection.disconnect()
            return {
                'host': device['host'],
                'error': str(e),
                'status': 'failed'
            }
    return {
        'host': device['host'],
        'error': 'Connection failed',
        'status': 'failed'
    }
```

```
# Usage example
if __name__ == '__main__':
    nm = NetworkManager('devices.json')

    # Backup all configurations
    backup_results = nm.backup_configurations()
    print("Backup Results:", json.dumps(backup_results, indent=2))

    # Execute commands in parallel
    commands = ['show version', 'show ip interface brief', 'show running-config | include ho
    results = nm.execute_commands_parallel(commands)

    # Save results to file
    with open('command_results.json', 'w') as f:
        json.dump(results, f, indent=2)
```

Network Monitoring Script

```
#!/usr/bin/env python3
# network_monitor.py
import netmiko
import json
import time
import smtplib
from email.mime.text import MimeText
from email.mime.multipart import MimeMultipart
import threading

class NetworkMonitor:
    def __init__(self, config_file):
        with open(config_file, 'r') as f:
            self.config = json.load(f)
        self.devices = self.config['devices']
        self.thresholds = self.config['thresholds']
        self.email_config = self.config['email']
        self.monitoring = True

    def check_interface_utilization(self, device):
        try:
            connection = netmiko.ConnectHandler(**device)
            output = connection.send_command('show interfaces | include rate')
            connection.disconnect()

            alerts = []
            for line in output.split('\n'):
                if 'input rate' in line and 'output rate' in line:
                    # Parse interface utilization
                    parts = line.split()
                    input_rate = int(parts[4])
                    output_rate = int(parts[9])

                    if input_rate > self.thresholds['interface_utilization']:
                        alerts.append(f"High input rate on {device['host']}: {input_rate} bp
                    if output_rate > self.thresholds['interface_utilization']:
                        alerts.append(f"High output rate on {device['host']}: {output_rate}

            return alerts
        except Exception as e:
            return [f"Error checking {device['host']}: {str(e)}"]

    def check_cpu_utilization(self, device):
        try:
            connection = netmiko.ConnectHandler(**device)
            output = connection.send_command('show processes cpu | include CPU')
            connection.disconnect()

            # Parse CPU utilization
            for line in output.split('\n'):
                if 'CPU utilization' in line:
                    cpu_percent = int(line.split('%')[0].split()[-1])
                    if cpu_percent > self.thresholds['cpu_utilization']:
                        return [f"High CPU utilization on {device['host']}: {cpu_percent}%"]

            return []
        except Exception as e:
            return [f"Error checking CPU on {device['host']}: {str(e)}"]

    def check_memory_utilization(self, device):
        try:
            connection = netmiko.ConnectHandler(**device)
            output = connection.send_command('show memory statistics')
            connection.disconnect()

            # Parse memory utilization
            for line in output.split('\n'):
                if 'Processor' in line and 'Used' in line:
                    parts = line.split()
                    used = int(parts[2])
                    total = int(parts[1])
                    utilization = (used / total) * 100

                    if utilization > self.thresholds['memory_utilization']:
                        return [f"High memory utilization on {device['host']}: {utilization:

            return []
        except Exception as e:
            return [f"Error checking memory on {device['host']}: {str(e)}"]

    def send_alert(self, alerts):
        if not alerts:
            return

        msg = MimeMultipart()
        msg['From'] = self.email_config['from']
        msg['To'] = ', '.join(self.email_config['to'])
        msg['Subject'] = 'Network Alert'

        body = '\n'.join(alerts)
        msg.attach(MimeText(body, 'plain'))

        try:
            server = smtplib.SMTP(self.email_config['smtp_server'], self.email_config['smtp_
            server.starttls()
            server.login(self.email_config['username'], self.email_config['password'])
            server.send_message(msg)
            server.quit()
            print(f"Alert sent: {len(alerts)} issues")
        except Exception as e:
            print(f"Failed to send alert: {str(e)}")

    def monitor_device(self, device):
        all_alerts = []

        # Check various metrics
        all_alerts.extend(self.check_interface_utilization(device))
        all_alerts.extend(self.check_cpu_utilization(device))
        all_alerts.extend(self.check_memory_utilization(device))

        return all_alerts

    def start_monitoring(self, interval=300):  # 5 minutes
        print(f"Starting network monitoring (interval: {interval}s)")

        while self.monitoring:
            all_alerts = []

            # Monitor all devices
            for device in self.devices:
                alerts = self.monitor_device(device)
                all_alerts.extend(alerts)

            # Send alerts if any issues found
            if all_alerts:
                self.send_alert(all_alerts)

            time.sleep(interval)

    def stop_monitoring(self):
        self.monitoring = False
        print("Monitoring stopped")

# Configuration file example
monitor_config = {
    "devices": [
        {
            "device_type": "cisco_ios",
            "host": "172.20.20.11",
            "username": "ansible",
            "password": "ansible123"
        }
    ],
    "thresholds": {
        "interface_utilization": 80000000,  # 80 Mbps
        "cpu_utilization": 80,              # 80%
        "memory_utilization": 80            # 80%
    },
    "email": {
        "smtp_server": "smtp.gmail.com",
        "smtp_port": 587,
        "from": "alerts@company.com",
        "to": ["admin@company.com"],
        "username": "alerts@company.com",
        "password": "app_password"
    }
}

# Save configuration
with open('monitor_config.json', 'w') as f:
    json.dump(monitor_config, f, indent=2)
```

NETCONF and RESTCONF

NETCONF Implementation

NETCONF Client Script

```python
#!/usr/bin/env python3
# netconf_client.py
from ncclient import manager
import xml.etree.ElementTree as ET
import json

class NetconfClient:
    def __init__(self, host, username, password, port=830):
        self.host = host
        self.username = username
        self.password = password
        self.port = port
        self.connection = None

    def connect(self):
        try:
            self.connection = manager.connect(
                host=self.host,
                port=self.port,
                username=self.username,
                password=self.password,
                hostkey_verify=False,
                device_params={'name': 'iosxe'}
            )
            print(f"Connected to {self.host} via NETCONF")
            return True
        except Exception as e:
            print(f"Failed to connect: {str(e)}")
            return False

    def get_config(self, source='running'):
        if not self.connection:
            return None

        try:
            config = self.connection.get_config(source=source)
            return config.data_xml
        except Exception as e:
            print(f"Failed to get config: {str(e)}")
            return None
    def get_interfaces(self):
        filter_xml = """
        <filter>
            <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
                <interface>
                    <name/>
                    <type/>
                    <enabled/>
                    <oper-status/>
                </interface>
            </interfaces>
        </filter>
        """

        try:
            result = self.connection.get(filter=filter_xml)
            return result.data_xml
        except Exception as e:
            print(f"Failed to get interfaces: {str(e)}")
            return None

    def configure_interface(self, interface_name, ip_address, subnet_mask):
        config_xml = f"""
        <config>
            <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
                <interface>
                    <name>{interface_name}</name>
                    <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
                        ianaift:ethernetCsmacd
                    </type>
                    <enabled>true</enabled>
                    <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                        <address>
                            <ip>{ip_address}</ip>
                            <netmask>{subnet_mask}</netmask>
                        </address>
                    </ipv4>
                </interface>
            </interfaces>
        </config>
        """

        try:
            result = self.connection.edit_config(target='running', config=config_xml)
            print(f"Interface {interface_name} configured successfully")
            return True
        except Exception as e:
            print(f"Failed to configure interface: {str(e)}")
            return False

    def disconnect(self):
        if self.connection:
            self.connection.close_session()
            print("NETCONF session closed")

# Usage example
if __name__ == '__main__':
    client = NetconfClient('172.20.20.11', 'ansible', 'ansible123')

    if client.connect():
        # Get interface information
        interfaces = client.get_interfaces()
        if interfaces:
            print("Current interfaces:")
            print(interfaces)

        # Configure an interface
        client.configure_interface('GigabitEthernet0/0/1', '10.1.1.1', '255.255.255.0')

        client.disconnect()
```

RESTCONF Implementation

RESTCONF Client Script

```
#!/usr/bin/env python3
# restconf_client.py
import requests
import json
from requests.auth import HTTPBasicAuth
import urllib3

# Disable SSL warnings
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

class RestconfClient:
    def __init__(self, host, username, password, port=443):
        self.host = host
        self.username = username
        self.password = password
        self.port = port
        self.base_url = f"https://{host}:{port}/restconf"
        self.auth = HTTPBasicAuth(username, password)
        self.headers = {
            'Accept': 'application/yang-data+json',
            'Content-Type': 'application/yang-data+json'
        }

    def get_capabilities(self):
        url = f"{self.base_url}/data/ietf-restconf-monitoring:restconf-state/capabilities"
        try:
            response = requests.get(url, auth=self.auth, headers=self.headers, verify=False)
            if response.status_code == 200:
                return response.json()
            else:
                print(f"Failed to get capabilities: {response.status_code}")
                return None
        except Exception as e:
            print(f"Error getting capabilities: {str(e)}")
            return None

    def get_interfaces(self):
        url = f"{self.base_url}/data/ietf-interfaces:interfaces"
        try:
            response = requests.get(url, auth=self.auth, headers=self.headers, verify=False)
            if response.status_code == 200:
                return response.json()
            else:
                print(f"Failed to get interfaces: {response.status_code}")
                return None
        except Exception as e:
            print(f"Error getting interfaces: {str(e)}")
            return None

    def create_interface(self, interface_data):
        url = f"{self.base_url}/data/ietf-interfaces:interfaces"
        try:
            response = requests.post(
                url,
                auth=self.auth,
                headers=self.headers,
                data=json.dumps(interface_data),
                verify=False
            )
            if response.status_code in [200, 201, 204]:
                print("Interface created successfully")
                return True
            else:
                print(f"Failed to create interface: {response.status_code}")
                print(response.text)
                return False
        except Exception as e:
            print(f"Error creating interface: {str(e)}")
            return False

    def update_interface(self, interface_name, interface_data):
        url = f"{self.base_url}/data/ietf-interfaces:interfaces/interface={interface_name}"
        try:
            response = requests.put(
                url,
                auth=self.auth,
                headers=self.headers,
                data=json.dumps(interface_data),
                verify=False
            )
            if response.status_code in [200, 204]:
                print(f"Interface {interface_name} updated successfully")
                return True
            else:
                print(f"Failed to update interface: {response.status_code}")
                return False
        except Exception as e:
            print(f"Error updating interface: {str(e)}")
            return False

    def delete_interface(self, interface_name):
        url = f"{self.base_url}/data/ietf-interfaces:interfaces/interface={interface_name}"
        try:
            response = requests.delete(url, auth=self.auth, headers=self.headers, verify=Fal
            if response.status_code in [200, 204]:
                print(f"Interface {interface_name} deleted successfully")
                return True
            else:
                print(f"Failed to delete interface: {response.status_code}")
                return False
        except Exception as e:
            print(f"Error deleting interface: {str(e)}")
            return False

# Usage example
if __name__ == '__main__':
    client = RestconfClient('172.20.20.11', 'ansible', 'ansible123')

    # Get current interfaces
```

```python
    interfaces = client.get_interfaces()
    if interfaces:
        print("Current interfaces:")
        print(json.dumps(interfaces, indent=2))

    # Create new interface configuration
    new_interface = {
        "ietf-interfaces:interface": {
            "name": "Loopback100",
            "type": "iana-if-type:softwareLoopback",
            "enabled": True,
            "ietf-ip:ipv4": {
                "address": [
                    {
                        "ip": "100.100.100.100",
                        "netmask": "255.255.255.255"
                    }
                ]
            }
        }
    }

    # Create the interface
    client.create_interface(new_interface)
```

CI/CD for Network Configuration

GitLab CI/CD Pipeline

```yaml
# .gitlab-ci.yml
stages:
  - validate
  - test
  - deploy
  - verify

variables:
  ANSIBLE_HOST_KEY_CHECKING: "False"
  ANSIBLE_STDOUT_CALLBACK: "yaml"

validate_syntax:
  stage: validate
  image: python:3.9
  before_script:
    - pip install ansible yamllint ansible-lint
  script:
    - yamllint playbooks/
    - ansible-lint playbooks/
    - ansible-playbook --syntax-check playbooks/site.yml
  only:
    - merge_requests
    - master

test_configurations:
  stage: test
  image: python:3.9
  before_script:
    - pip install ansible netmiko pytest
  script:
    - python -m pytest tests/ -v
    - ansible-playbook playbooks/site.yml --check --diff
  only:
    - merge_requests
    - master

deploy_staging:
  stage: deploy
  image: python:3.9
  before_script:
    - pip install ansible netmiko
  script:
    - ansible-playbook -i inventory/staging playbooks/site.yml
  environment:
    name: staging
  only:
    - master
  when: manual

deploy_production:
  stage: deploy
  image: python:3.9
  before_script:
    - pip install ansible netmiko
  script:
    - ansible-playbook -i inventory/production playbooks/site.yml
  environment:
    name: production
  only:
    - master
  when: manual
```

```
verify_deployment:
  stage: verify
  image: python:3.9
  before_script:
    - pip install ansible netmiko pytest
  script:
    - python -m pytest tests/integration/ -v
    - ansible-playbook playbooks/verify.yml
  only:
    - master
  dependencies:
    - deploy_production
```

Network Testing Framework

```
#!/usr/bin/env python3
# tests/test_network.py
import pytest
import netmiko
import json
import re

class TestNetworkConnectivity:
    @pytest.fixture(scope="class")
    def devices(self):
        with open('inventory/devices.json', 'r') as f:
            return json.load(f)

    @pytest.fixture(scope="class")
    def connections(self, devices):
        connections = {}
        for device in devices:
            try:
                conn = netmiko.ConnectHandler(**device)
                connections[device['host']] = conn
            except Exception as e:
                pytest.fail(f"Failed to connect to {device['host']}: {str(e)}")
        yield connections

        # Cleanup
        for conn in connections.values():
            conn.disconnect()

    def test_device_reachability(self, connections):
        """Test that all devices are reachable"""
        for host, conn in connections.items():
            output = conn.send_command("show version")
            assert "uptime" in output.lower(), f"Device {host} not responding properly"

    def test_interface_status(self, connections):
        """Test that critical interfaces are up"""
        critical_interfaces = {
            '172.20.20.11': ['GigabitEthernet0/0/0'],
            '172.20.20.12': ['GigabitEthernet1/0/1'],
            '172.20.20.13': ['GigabitEthernet1/0/1']
        }

        for host, conn in connections.items():
            if host in critical_interfaces:
                output = conn.send_command("show ip interface brief")
                for interface in critical_interfaces[host]:
                    assert interface in output, f"Interface {interface} not found on {host}"
                    # Check if interface is up
                    pattern = rf"{interface}\s+\S+\s+\S+\s+up\s+up"
                    assert re.search(pattern, output), f"Interface {interface} is down on {h

    def test_routing_table(self, connections):
        """Test that routing tables contain expected routes"""
        for host, conn in connections.items():
            output = conn.send_command("show ip route")
            # Should have at least connected routes
            assert "C " in output, f"No connected routes found on {host}"

    def test_vlan_configuration(self, connections):
        """Test VLAN configuration on switches"""
        switch_hosts = ['172.20.20.12', '172.20.20.13']
        expected_vlans = ['10', '20', '30']

        for host in switch_hosts:
            if host in connections:
                conn = connections[host]
                output = conn.send_command("show vlan brief")
                for vlan in expected_vlans:
                    assert vlan in output, f"VLAN {vlan} not found on switch {host}"

    def test_spanning_tree(self, connections):
        """Test spanning tree status"""
        switch_hosts = ['172.20.20.12', '172.20.20.13']

        for host in switch_hosts:
            if host in connections:
                conn = connections[host]
                output = conn.send_command("show spanning-tree summary")
                assert "forwarding" in output.lower(), f"No forwarding ports on {host}"

    def test_ntp_synchronization(self, connections):
        """Test NTP synchronization"""
        for host, conn in connections.items():
            output = conn.send_command("show ntp status")
            # Should show synchronized or at least configured
            assert "Clock is" in output, f"NTP not configured on {host}"

class TestNetworkSecurity:
    @pytest.fixture(scope="class")
    def connections(self):
        # Same as above
        pass

    def test_ssh_access_only(self, connections):
        """Test that only SSH is enabled for remote access"""
        for host, conn in connections.items():
            output = conn.send_command("show running-config | include line vty")
            assert "transport input ssh" in output, f"Telnet may be enabled on {host}"

    def test_snmp_community(self, connections):
        """Test SNMP community strings"""
        for host, conn in connections.items():
            output = conn.send_command("show running-config | include snmp-server community"
            # Should not have default communities
            assert "public" not in output, f"Default SNMP community found on {host}"
            assert "private" not in output, f"Default SNMP community found on {host}"

if __name__ == '__main__':
    pytest.main(['-v', __file__])
```

Summary

Advanced network automation combines multiple technologies and methodologies to create robust, scalable, and maintainable network infrastructure. Understanding Ansible automation, Python scripting, NETCONF/RESTCONF protocols, and CI/CD practices enables modern network operations and DevNetOps workflows.

Key concepts covered: - Advanced Ansible playbooks and inventory management - Python scripting for network management and monitoring - NETCONF and RESTCONF for programmable networks - CI/CD pipelines for network configuration management - Automated testing frameworks for network validation

In the next chapter, we'll explore network orchestration and Infrastructure as Code (IaC) concepts for large-scale network deployments.

Review Questions

1. How do you implement dynamic inventory in Ansible for network devices?
2. What are the advantages of NETCONF over traditional CLI-based management?
3. How do you create automated testing for network configurations?
4. What are best practices for CI/CD in network automation?
5. How do you implement parallel execution in Python network scripts?

Hands-on Exercises

Exercise 1: Advanced Ansible Automation

1. Deploy the network automation lab
2. Create dynamic inventory scripts
3. Develop comprehensive playbooks for device configuration
4. Implement Jinja2 templates for configuration generation

Exercise 2: Python Network Management

1. Create a network monitoring script with alerting
2. Implement parallel device management
3. Build configuration backup and restore functionality
4. Develop network discovery and documentation tools

Exercise 3: NETCONF/RESTCONF Implementation

1. Configure NETCONF on network devices
2. Create NETCONF client scripts for configuration management
3. Implement RESTCONF API interactions
4. Compare NETCONF vs RESTCONF performance and capabilities

Exercise 4: CI/CD Pipeline Development

1. Set up GitLab CI/CD for network automation
2. Create automated testing frameworks
3. Implement staging and production deployment workflows
4. Develop rollback and recovery procedures

Additional Resources

- Ansible Network Automation Guide
- Python Network Programming
- NETCONF Protocol RFC
- RESTCONF Protocol RFC
- Network DevOps Best Practices

Opensource

Chapter 53: FRRouting (FRR) - Open Source Routing Suite

Learning Objectives

By the end of this chapter, you will be able to: - Understand FRRouting architecture and capabilities - Deploy FRR containers in ContainerLab environments - Configure multiple routing protocols using FRR - Implement advanced routing features with open source tools - Integrate FRR with commercial network equipment

Introduction to FRRouting (FRR)

What is FRRouting?

FRRouting (FRR) is a free and open source Internet routing protocol suite for Linux and Unix platforms. It implements BGP, OSPF, RIP, IS-IS, PIM, LDP, BFD, and Babel routing protocols. FRR is a fork of Quagga and is actively developed by a community of network engineers and developers.

Key FRR Features

- **Multi-Protocol Support**: BGP, OSPF, IS-IS, RIP, PIM, LDP, BFD
- **High Performance**: Optimized for modern hardware
- **Standards Compliant**: Implements RFC standards
- **Extensible**: Plugin architecture for new features
- **Production Ready**: Used by major cloud providers and ISPs
- **Container Native**: Excellent ContainerLab integration

FRR Architecture

Core Components

- **zebra**: Kernel routing manager and interface to other daemons
- **bgpd**: BGP daemon
- **ospfd**: OSPF daemon
- **ospf6d**: OSPFv3 daemon
- **ripd**: RIP daemon
- **ripngd**: RIPng daemon

- **isisd**: IS-IS daemon
- **pimd**: PIM daemon
- **ldpd**: LDP daemon
- **bfdd**: BFD daemon

FRR Lab Environment

Basic FRR Lab Setup

```
# FRR comprehensive lab
name: frr-routing-lab
prefix: frr

topology:
  nodes:
    # FRR routers
    frr-r1:
      kind: linux
      image: frrouting/frr:latest
      mgmt-ipv4: 172.20.20.10
      exec:
        - ip addr add 10.1.12.1/30 dev eth1
        - ip addr add 10.1.13.1/30 dev eth2
        - ip addr add 192.168.1.1/24 dev eth3
        - ip link set eth1 up
        - ip link set eth2 up
        - ip link set eth3 up
      binds:
        - ./configs/frr-r1:/etc/frr

    frr-r2:
      kind: linux
      image: frrouting/frr:latest
      mgmt-ipv4: 172.20.20.11
      exec:
        - ip addr add 10.1.12.2/30 dev eth1
        - ip addr add 10.1.24.1/30 dev eth2
        - ip addr add 192.168.2.1/24 dev eth3
        - ip link set eth1 up
        - ip link set eth2 up
        - ip link set eth3 up
      binds:
        - ./configs/frr-r2:/etc/frr

    frr-r3:
```

```yaml
      kind: linux
      image: frrouting/frr:latest
      mgmt-ipv4: 172.20.20.12
      exec:
        - ip addr add 10.1.13.2/30 dev eth1
        - ip addr add 10.1.34.1/30 dev eth2
        - ip addr add 192.168.3.1/24 dev eth3
        - ip link set eth1 up
        - ip link set eth2 up
        - ip link set eth3 up
      binds:
        - ./configs/frr-r3:/etc/frr

    frr-r4:
      kind: linux
      image: frrouting/frr:latest
      mgmt-ipv4: 172.20.20.13
      exec:
        - ip addr add 10.1.24.2/30 dev eth1
        - ip addr add 10.1.34.2/30 dev eth2
        - ip addr add 192.168.4.1/24 dev eth3
        - ip link set eth1 up
        - ip link set eth2 up
        - ip link set eth3 up
      binds:
        - ./configs/frr-r4:/etc/frr

    # BGP route reflector
    frr-rr:
      kind: linux
      image: frrouting/frr:latest
      mgmt-ipv4: 172.20.20.14
      exec:
        - ip addr add 10.1.15.1/30 dev eth1
        - ip addr add 10.1.25.1/30 dev eth2
        - ip addr add 10.1.35.1/30 dev eth3
        - ip addr add 10.1.45.1/30 dev eth4
        - ip link set eth1 up
        - ip link set eth2 up
        - ip link set eth3 up
        - ip link set eth4 up
      binds:
        - ./configs/frr-rr:/etc/frr

    # Test clients
    client1:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.1.10/24 dev eth1
        - ip route add default via 192.168.1.1

    client2:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.2.10/24 dev eth1
        - ip route add default via 192.168.2.1

    client3:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.3.10/24 dev eth1
        - ip route add default via 192.168.3.1

    client4:
      kind: linux
      image: alpine:latest
      exec:
        - ip addr add 192.168.4.10/24 dev eth1
        - ip route add default via 192.168.4.1

  links:
    # Core network mesh
    - endpoints: ["frr-r1:eth1", "frr-r2:eth1"]
    - endpoints: ["frr-r1:eth2", "frr-r3:eth1"]
    - endpoints: ["frr-r2:eth2", "frr-r4:eth1"]
    - endpoints: ["frr-r3:eth2", "frr-r4:eth2"]

    # Route reflector connections
    - endpoints: ["frr-rr:eth1", "frr-r1:eth4"]
    - endpoints: ["frr-rr:eth2", "frr-r2:eth4"]
    - endpoints: ["frr-rr:eth3", "frr-r3:eth4"]
    - endpoints: ["frr-rr:eth4", "frr-r4:eth4"]

    # Client connections
    - endpoints: ["frr-r1:eth3", "client1:eth1"]
    - endpoints: ["frr-r2:eth3", "client2:eth1"]
    - endpoints: ["frr-r3:eth3", "client3:eth1"]
    - endpoints: ["frr-r4:eth3", "client4:eth1"]
```

FRR Configuration Files

Basic FRR Configuration Structure

```
# Create configuration directories
mkdir -p configs/frr-r1 configs/frr-r2 configs/frr-r3 configs/frr-r4 configs/frr-rr

# FRR daemon configuration
cat > configs/frr-r1/daemons << 'EOF'
# FRR daemon configuration
bgpd=yes
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
isisd=no
pimd=no
ldpd=no
nhrpd=no
eigrpd=no
babeld=no
sharpd=no
pbrd=no
bfdd=yes
fabricd=no
vrrpd=no

# Integrated config file
vtysh_enable=yes
zebra_options="  -A 127.0.0.1 -s 90000000"
bgpd_options="   -A 127.0.0.1"
ospfd_options="  -A 127.0.0.1"
bfdd_options="   -A 127.0.0.1"
EOF
```

OSPF Configuration

```
# FRR-R1 OSPF Configuration
cat > configs/frr-r1/frr.conf << 'EOF'
frr version 8.4
frr defaults traditional
hostname frr-r1
log syslog informational
service integrated-vtysh-config
!
```

```
interface eth1
 description To-FRR-R2
 ip address 10.1.12.1/30
 ip ospf area 0
 ip ospf hello-interval 5
 ip ospf dead-interval 20
!
interface eth2
 description To-FRR-R3
 ip address 10.1.13.1/30
 ip ospf area 0
!
interface eth3
 description LAN-Network
 ip address 192.168.1.1/24
 ip ospf area 1
!
interface lo
 ip address 1.1.1.1/32
 ip ospf area 0
!
router ospf
 ospf router-id 1.1.1.1
 log-adjacency-changes detail
 area 1 range 192.168.1.0/24
 passive-interface eth3
!
line vty
!
EOF
```

BGP Configuration

```
# FRR-R1 BGP Configuration
cat > configs/frr-r1/frr.conf << 'EOF'
frr version 8.4
frr defaults traditional
hostname frr-r1
log syslog informational
service integrated-vtysh-config
!
interface eth1
 description To-FRR-R2
 ip address 10.1.12.1/30
!
```

```
interface eth2
 description To-FRR-R3
 ip address 10.1.13.1/30
!
interface eth3
 description LAN-Network
 ip address 192.168.1.1/24
!
interface eth4
 description To-Route-Reflector
 ip address 10.1.15.2/30
!
interface lo
 ip address 1.1.1.1/32
!
router bgp 65001
 bgp router-id 1.1.1.1
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 neighbor 10.1.15.1 remote-as 65001
 neighbor 10.1.15.1 description Route-Reflector
 neighbor 10.1.15.1 update-source lo
 !
 address-family ipv4 unicast
  network 192.168.1.0/24
  neighbor 10.1.15.1 activate
  neighbor 10.1.15.1 route-reflector-client
 exit-address-family
!
line vty
!
EOF
```

FRR Protocol Configuration

OSPF with FRR

Multi-Area OSPF Setup

```
# Deploy FRR lab
containerlab deploy -t frr-routing-lab.yml

# Connect to FRR-R1 and configure OSPF
docker exec -it clab-frr-frr-r1 vtysh
```

```
# Enter configuration mode
configure terminal

# Configure OSPF
router ospf
 ospf router-id 1.1.1.1
 log-adjacency-changes detail
 area 1 range 192.168.0.0/22
 area 1 stub
 passive-interface eth3
 network 10.1.12.0/30 area 0
 network 10.1.13.0/30 area 0
 network 192.168.1.0/24 area 1
 network 1.1.1.1/32 area 0

# Verify OSPF configuration
show ip ospf neighbor
show ip ospf database
show ip route ospf
```

OSPF Authentication

```
# Configure OSPF authentication
interface eth1
 ip ospf message-digest-key 1 md5 SecureOSPFKey123

router ospf
 area 0 authentication message-digest

# Verify authentication
show ip ospf interface eth1
```

BGP with FRR

eBGP Configuration

```
# Configure eBGP on FRR-R1
configure terminal

router bgp 65001
 bgp router-id 1.1.1.1
 bgp log-neighbor-changes
 no bgp default ipv4-unicast
 # eBGP neighbor
 neighbor 10.1.12.2 remote-as 65002
 neighbor 10.1.12.2 description FRR-R2-eBGP

 address-family ipv4 unicast
  network 192.168.1.0/24
  neighbor 10.1.12.2 activate
  neighbor 10.1.12.2 soft-reconfiguration inbound
 exit-address-family

# Verify BGP
show bgp summary
show bgp neighbors
show ip route bgp
```

Route Reflector Configuration

```
# Configure route reflector on FRR-RR
configure terminal

router bgp 65001
 bgp router-id 5.5.5.5
 bgp cluster-id 1.1.1.1

 # Route reflector clients
 neighbor 1.1.1.1 remote-as 65001
 neighbor 1.1.1.1 update-source lo
 neighbor 2.2.2.2 remote-as 65001
 neighbor 2.2.2.2 update-source lo
 neighbor 3.3.3.3 remote-as 65001
 neighbor 3.3.3.3 update-source lo
 neighbor 4.4.4.4 remote-as 65001
 neighbor 4.4.4.4 update-source lo

 address-family ipv4 unicast
  neighbor 1.1.1.1 activate
  neighbor 1.1.1.1 route-reflector-client
  neighbor 2.2.2.2 activate
  neighbor 2.2.2.2 route-reflector-client
  neighbor 3.3.3.3 activate
  neighbor 3.3.3.3 route-reflector-client
  neighbor 4.4.4.4 activate
  neighbor 4.4.4.4 route-reflector-client
 exit-address-family
```

IS-IS with FRR

Basic IS-IS Configuration

```
# Enable IS-IS daemon
# Edit /etc/frr/daemons: isisd=yes

# Configure IS-IS
configure terminal

router isis 1
 net 49.0001.0000.0000.0001.00
 is-type level-2-only
 log-adjacency-changes

interface eth1
 ip router isis 1
 isis circuit-type level-2-only
 isis hello-interval 3
 isis hello-multiplier 3

interface lo
 ip router isis 1
 isis circuit-type level-2-only
 isis passive

# Verify IS-IS
show isis neighbor
show isis database
show ip route isis
```

Advanced FRR Features

BFD (Bidirectional Forwarding Detection)

```
# Configure BFD for fast convergence
configure terminal

# Enable BFD globally
bfd
 peer 10.1.12.2
  detect-multiplier 3
  receive-interval 300
  transmit-interval 300
 exit

# Enable BFD for OSPF
router ospf
 bfd all-interfaces

# Enable BFD for BGP
router bgp 65001
 neighbor 10.1.12.2 bfd

# Verify BFD
show bfd peers
show bfd peer 10.1.12.2
```

Route Maps and Filtering

```
# Configure route maps
configure terminal

# Create access lists
ip prefix-list CUSTOMER-ROUTES seq 10 permit 192.168.0.0/16 le 24
ip prefix-list PROVIDER-ROUTES seq 10 permit 0.0.0.0/0 le 32

# Create route map
route-map CUSTOMER-IN permit 10
 match ip address prefix-list CUSTOMER-ROUTES
 set local-preference 200
route-map CUSTOMER-IN permit 20
 set local-preference 100

# Apply to BGP neighbor
router bgp 65001
 neighbor 10.1.12.2 route-map CUSTOMER-IN in

# Verify route map
show route-map
show bgp neighbors 10.1.12.2 received-routes
```

Multicast with PIM

```
# Enable PIM daemon
# Edit /etc/frr/daemons: pimd=yes

# Configure PIM
```

```
configure terminal

# Enable multicast forwarding
ip multicast-routing

# Configure PIM on interfaces
interface eth1
 ip pim sparse-mode
 ip igmp

interface eth2
 ip pim sparse-mode
 ip igmp

# Configure RP (Rendezvous Point)
ip pim rp 1.1.1.1 224.0.0.0/4

# Verify PIM
show ip pim neighbor
show ip pim rp-info
show ip mroute
```

FRR Integration with Commercial Equipment

FRR with Cisco Integration

```
# Mixed FRR and Cisco lab
name: frr-cisco-integration
topology:
  nodes:
    frr-router:
      kind: linux
      image: frrouting/frr:latest
      exec:
        - ip addr add 10.1.12.1/30 dev eth1
        - ip link set eth1 up
      binds:
        - ./configs/frr-mixed:/etc/frr

    cisco-router:
      kind: cisco_iosxe
      image: cisco/iosxe:latest
      startup-config: |
        hostname Cisco-Router
        !
        interface GigabitEthernet0/0/0
         ip address 10.1.12.2 255.255.255.252
         no shutdown
        !
        router bgp 65002
         bgp router-id 2.2.2.2
         neighbor 10.1.12.1 remote-as 65001
         network 203.0.113.0 mask 255.255.255.0
        !

  links:
    - endpoints: ["frr-router:eth1", "cisco-router:eth1"]
```

Protocol Interoperability Testing

```
# Test OSPF interoperability
docker exec -it clab-frr-cisco-integration-frr-router vtysh -c "show ip ospf neighbor"
docker exec -it clab-frr-cisco-integration-cisco-router cli -c "show ip ospf neighbor"

# Test BGP interoperability
docker exec -it clab-frr-cisco-integration-frr-router vtysh -c "show bgp summary"
docker exec -it clab-frr-cisco-integration-cisco-router cli -c "show bgp summary"
```

FRR Monitoring and Troubleshooting

FRR Diagnostic Commands

```
# System information
show version
show memory
show thread cpu

# Interface information
show interface
show interface eth1
show ip interface

# Routing information
show ip route
show ip route summary
show ip route 192.168.1.0/24

# Protocol-specific commands
```

```
show ip ospf
show ip ospf neighbor detail
show ip ospf database
show bgp summary
show bgp neighbors
show isis neighbor
show bfd peers
```

FRR Logging and Debugging

```
# Configure logging
configure terminal
log file /var/log/frr/frr.log informational
log syslog informational

# Enable debugging (use carefully)
debug ospf packet all
debug bgp updates
debug isis adj-packets

# View logs
show logging
```

Performance Monitoring

```
# Monitor FRR performance
show thread cpu
show memory
show zebra

# Interface statistics
show interface eth1
show ip route summary

# Protocol statistics
show ip ospf statistics
show bgp statistics
```

FRR Automation and Scripting

FRR Configuration Automation

```python
#!/usr/bin/env python3
# frr_config_manager.py
import subprocess
import json
import time

class FRRManager:
    def __init__(self, container_name):
        self.container_name = container_name

    def execute_vtysh_command(self, command):
        """Execute vtysh command in FRR container"""
        cmd = f"docker exec {self.container_name} vtysh -c '{command}'"
        result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
        return result.stdout, result.stderr

    def configure_ospf(self, router_id, networks):
        """Configure OSPF on FRR router"""
        commands = [
            "configure terminal",
            f"router ospf",
            f"ospf router-id {router_id}",
            "log-adjacency-changes detail"
        ]

        for network in networks:
            commands.append(f"network {network['network']} area {network['area']}")

        commands.append("exit")

        for cmd in commands:
            stdout, stderr = self.execute_vtysh_command(cmd)
            if stderr:
                print(f"Error executing '{cmd}': {stderr}")

    def configure_bgp(self, asn, router_id, neighbors):
        """Configure BGP on FRR router"""
        commands = [
            "configure terminal",
            f"router bgp {asn}",
            f"bgp router-id {router_id}",
            "bgp log-neighbor-changes",
            "no bgp default ipv4-unicast"
        ]

        for neighbor in neighbors:
            commands.extend([
                f"neighbor {neighbor['ip']} remote-as {neighbor['asn']}",
                f"neighbor {neighbor['ip']} description {neighbor.get('description', '')}",
                "address-family ipv4 unicast",
                f"neighbor {neighbor['ip']} activate",
                "exit-address-family"
            ])

        commands.append("exit")

        for cmd in commands:
            stdout, stderr = self.execute_vtysh_command(cmd)
            if stderr:
                print(f"Error executing '{cmd}': {stderr}")

    def get_routing_table(self):
        """Get routing table from FRR"""
        stdout, stderr = self.execute_vtysh_command("show ip route json")
        if not stderr:
            try:
                return json.loads(stdout)
            except json.JSONDecodeError:
                return None
        return None

    def get_bgp_summary(self):
        """Get BGP summary from FRR"""
        stdout, stderr = self.execute_vtysh_command("show bgp summary json")
        if not stderr:
            try:
                return json.loads(stdout)
            except json.JSONDecodeError:
                return None
        return None

# Usage example
if __name__ == '__main__':
    frr = FRRManager('clab-frr-frr-r1')

    # Configure OSPF
    networks = [
        {'network': '10.1.12.0/30', 'area': '0'},
        {'network': '192.168.1.0/24', 'area': '1'}
    ]
    frr.configure_ospf('1.1.1.1', networks)

    # Configure BGP
    neighbors = [
        {'ip': '10.1.12.2', 'asn': '65002', 'description': 'FRR-R2'}
    ]
    frr.configure_bgp('65001', '1.1.1.1', neighbors)

    # Get routing information
    routes = frr.get_routing_table()
    if routes:
        print("Routing table:", json.dumps(routes, indent=2))
```

FRR Best Practices

Configuration Management

1. **Version Control**: Store FRR configurations in Git
2. **Templating**: Use Jinja2 templates for consistent configs
3. **Validation**: Test configurations before deployment
4. **Backup**: Regular configuration backups
5. **Documentation**: Document network design and changes

Performance Optimization

```
# Optimize FRR performance
configure terminal

# Increase BGP keepalive frequency
router bgp 65001
 timers bgp 30 90

# Optimize OSPF timers
router ospf
 timers throttle spf 200 400 10000
 timers throttle lsa 200 400 10000

# Enable BFD for fast convergence
bfd
 peer 10.1.12.2
  detect-multiplier 3
  receive-interval 100
  transmit-interval 100
```

Security Hardening

```
# FRR security configuration
configure terminal

# BGP security
router bgp 65001
 neighbor 10.1.12.2 password SecureBGPKey123
 neighbor 10.1.12.2 ttl-security hops 1

# OSPF security
interface eth1
 ip ospf message-digest-key 1 md5 SecureOSPFKey456

router ospf
 area 0 authentication message-digest

# Access control
access-list 10 permit 10.1.12.0 0.0.0.3
line vty
 access-class 10 in
```

Summary

FRRouting provides a powerful, open-source alternative to commercial routing platforms. Its comprehensive protocol support, container-native design, and active development community make it an excellent choice for learning, testing, and production deployments. Understanding FRR capabilities enables cost-effective network implementations while maintaining enterprise-grade functionality.

Key concepts covered: - FRR architecture and daemon structure - Multi-protocol routing configuration (OSPF, BGP, IS-IS) - Advanced features (BFD, route maps, multicast) - Integration with commercial equipment - Automation and monitoring techniques - Performance optimization and security

In the next chapter, we'll explore VyOS, another powerful open-source network operating system with comprehensive routing and security features.

Review Questions

1. What are the main advantages of FRR over commercial routing platforms?
2. How do you configure multi-area OSPF in FRR?
3. What is the role of the zebra daemon in FRR architecture?
4. How do you implement BGP route reflection with FRR?
5. What are best practices for FRR security hardening?

Hands-on Exercises

Exercise 1: Basic FRR Deployment

1. Deploy the FRR routing lab
2. Configure OSPF on all routers
3. Verify neighbor relationships and routing tables
4. Test end-to-end connectivity

Exercise 2: BGP Configuration

1. Configure eBGP between different autonomous systems
2. Implement route reflector for iBGP scaling
3. Apply route maps for traffic engineering
4. Monitor BGP convergence and path selection

Exercise 3: Advanced Features

1. Configure BFD for fast convergence
2. Implement IS-IS as alternative IGP
3. Set up PIM for multicast routing
4. Test protocol interoperability with commercial equipment

Exercise 4: FRR Automation

1. Create Python scripts for FRR configuration management
2. Implement automated monitoring and alerting
3. Develop configuration templates and validation
4. Build CI/CD pipeline for FRR deployments

Additional Resources

- FRRouting Official Documentation
- FRR GitHub Repository
- FRR Configuration Examples
- FRR Community and Support

Chapter 54: VyOS - Open Source Network Operating System

Learning Objectives

By the end of this chapter, you will be able to: - Deploy and configure VyOS in ContainerLab environments - Implement routing, switching, and security features with VyOS - Configure VPN services and firewall policies - Integrate VyOS with existing network infrastructure - Automate VyOS configuration and management

Introduction to VyOS

What is VyOS?

VyOS is a Linux-based network operating system that provides software-based network routing, firewall, and VPN functionality. It's based on Debian Linux and uses a unified configuration interface similar to Juniper JunOS. VyOS is the open-source continuation of the Vyatta project.

Key VyOS Features

- **Unified CLI**: Consistent command structure across all features
- **Routing Protocols**: OSPF, BGP, RIP, IS-IS, EIGRP support
- **VPN Services**: IPSec, OpenVPN, WireGuard, L2TP
- **Firewall**: Stateful packet filtering and NAT
- **Load Balancing**: WAN load balancing and failover
- **High Availability**: VRRP and clustering support
- **Container Ready**: Excellent ContainerLab integration

VyOS Architecture

Core Components

- **Configuration System**: Hierarchical configuration tree
- **Routing Engine**: FRRouting integration
- **Firewall Engine**: netfilter/iptables based
- **VPN Engine**: strongSwan and OpenVPN integration
- **Management Interface**: Web GUI and API

VyOS Lab Environment

Comprehensive VyOS Lab Setup

```yaml
# VyOS comprehensive lab
name: vyos-network-lab
prefix: vyos

topology:
  nodes:
    # VyOS routers
    vyos-r1:
      kind: linux
      image: vyos/vyos:1.4-rolling
      mgmt-ipv4: 172.20.20.10
      cmd: /sbin/init
      binds:
        - ./configs/vyos-r1:/opt/vyatta/etc/config
      env:
        VYOS_STARTUP_CONFIG: /opt/vyatta/etc/config/config.boot

    vyos-r2:
      kind: linux
      image: vyos/vyos:1.4-rolling
      mgmt-ipv4: 172.20.20.11
      cmd: /sbin/init
      binds:
        - ./configs/vyos-r2:/opt/vyatta/etc/config
      env:
        VYOS_STARTUP_CONFIG: /opt/vyatta/etc/config/config.boot

    vyos-r3:
      kind: linux
      image: vyos/vyos:1.4-rolling
      mgmt-ipv4: 172.20.20.12
      cmd: /sbin/init
      binds:
        - ./configs/vyos-r3:/opt/vyatta/etc/config
      env:
        VYOS_STARTUP_CONFIG: /opt/vyatta/etc/config/config.boot

    # VyOS firewall/gateway
    vyos-fw:
      kind: linux
      image: vyos/vyos:1.4-rolling
      mgmt-ipv4: 172.20.20.13
      cmd: /sbin/init
      binds:
        - ./configs/vyos-fw:/opt/vyatta/etc/config
      env:
        VYOS_STARTUP_CONFIG: /opt/vyatta/etc/config/config.boot

    # Internal networks
    internal-server:
      kind: linux
      image: nginx:alpine
      mgmt-ipv4: 172.20.20.20
      exec:
        - ip addr add 192.168.10.10/24 dev eth1
        - ip route add default via 192.168.10.1

    dmz-server:
      kind: linux
      image: nginx:alpine
      mgmt-ipv4: 172.20.20.21
      exec:
        - ip addr add 192.168.20.10/24 dev eth1
        - ip route add default via 192.168.20.1

    external-client:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.22
      exec:
        - ip addr add 203.0.113.10/24 dev eth1
        - ip route add default via 203.0.113.1
        - apk add --no-cache curl iperf3

    # Branch office
    branch-client:
      kind: linux
      image: alpine:latest
      mgmt-ipv4: 172.20.20.23
      exec:
        - ip addr add 192.168.30.10/24 dev eth1
        - ip route add default via 192.168.30.1

  links:
    # Core network
    - endpoints: ["vyos-r1:eth1", "vyos-r2:eth1"]
    - endpoints: ["vyos-r1:eth2", "vyos-r3:eth1"]
    - endpoints: ["vyos-r2:eth2", "vyos-r3:eth2"]
    # Firewall connections
    - endpoints: ["vyos-r1:eth3", "vyos-fw:eth1"]
    - endpoints: ["vyos-fw:eth2", "internal-server:eth1"]
    - endpoints: ["vyos-fw:eth3", "dmz-server:eth1"]
    - endpoints: ["vyos-fw:eth4", "external-client:eth1"]

    # Branch office
    - endpoints: ["vyos-r3:eth3", "branch-client:eth1"]
```

VyOS Configuration Structure

Basic VyOS Configuration

```
# Create configuration directories
mkdir -p configs/vyos-r1 configs/vyos-r2 configs/vyos-r3 configs/vyos-fw

# VyOS-R1 Configuration
cat > configs/vyos-r1/config.boot << 'EOF'
interfaces {
    ethernet eth1 {
        address 10.1.12.1/30
        description "To-VyOS-R2"
    }
    ethernet eth2 {
        address 10.1.13.1/30
        description "To-VyOS-R3"
    }
    ethernet eth3 {
        address 10.1.14.1/30
        description "To-Firewall"
    }
    loopback lo {
        address 1.1.1.1/32
    }
}

protocols {
    ospf {
        area 0 {
            network 10.1.12.0/30
            network 10.1.13.0/30
            network 10.1.14.0/30
            network 1.1.1.1/32
        }
        log-adjacency-changes
        parameters {
            router-id 1.1.1.1
        }
    }
}

system {
    config-management {
        commit-revisions 100
    }
    console {
        device ttyS0 {
            speed 115200
        }
    }
    host-name vyos-r1
    login {
        user vyos {
            authentication {
                encrypted-password $6$rounds=656000$YxM3u8HhkvEm0x7C$w2S9GPSKVVppNHqcq8Qg.7Q
                plaintext-password ""
            }
            level admin
        }
    }
    ntp {
        server time1.vyos.net
        server time2.vyos.net
    }
    syslog {
        global {
            facility all {
                level info
            }
        }
    }
}
EOF
```

VyOS Routing Configuration

OSPF Configuration

```
# Deploy VyOS lab
containerlab deploy -t vyos-network-lab.yml

# Connect to VyOS-R1
docker exec -it clab-vyos-vyos-r1 vbash

# Enter configuration mode
configure

# Configure OSPF
set protocols ospf area 0 network 10.1.12.0/30
set protocols ospf area 0 network 10.1.13.0/30
set protocols ospf area 0 network 1.1.1.1/32
set protocols ospf parameters router-id 1.1.1.1
set protocols ospf log-adjacency-changes

# Configure interfaces
set interfaces ethernet eth1 address 10.1.12.1/30
set interfaces ethernet eth1 description "To-VyOS-R2"
set interfaces ethernet eth2 address 10.1.13.1/30
set interfaces ethernet eth2 description "To-VyOS-R3"
set interfaces loopback lo address 1.1.1.1/32

# Commit and save
commit
save

# Verify OSPF
show ip ospf neighbor
show ip route ospf
```

BGP Configuration

```
# Configure BGP on VyOS-R1
configure

# Basic BGP setup
set protocols bgp 65001 parameters router-id 1.1.1.1
set protocols bgp 65001 parameters log-neighbor-changes

# eBGP neighbor
```

```
set protocols bgp 65001 neighbor 10.1.12.2 remote-as 65002
set protocols bgp 65001 neighbor 10.1.12.2 description "VyOS-R2-eBGP"
set protocols bgp 65001 neighbor 10.1.12.2 address-family ipv4-unicast

# Network advertisement
set protocols bgp 65001 address-family ipv4-unicast network 1.1.1.1/32
set protocols bgp 65001 address-family ipv4-unicast network 192.168.1.0/24

# Route maps
set policy route-map LOCAL-PREF rule 10 action permit
set policy route-map LOCAL-PREF rule 10 match ip address prefix-list CUSTOMER-ROUTES
set policy route-map LOCAL-PREF rule 10 set local-preference 200

set protocols bgp 65001 neighbor 10.1.12.2 address-family ipv4-unicast route-map import LOCA

commit
save

# Verify BGP
show ip bgp summary
show ip bgp neighbors
show ip route bgp
```

Static Routing

```
# Configure static routes
configure

# Default route
set protocols static route 0.0.0.0/0 next-hop 203.0.113.1

# Specific static routes
set protocols static route 192.168.100.0/24 next-hop 10.1.14.2
set protocols static route 192.168.100.0/24 description "Internal Network"

# Route with administrative distance
set protocols static route 192.168.200.0/24 next-hop 10.1.14.3 distance 200

commit
save

# Verify static routes
show ip route static
show ip route
```

VyOS Firewall Configuration

Zone-Based Firewall

```
# Configure firewall zones
configure

# Define zones
set zone-policy zone LAN description "Internal LAN"
set zone-policy zone DMZ description "DMZ Network"
set zone-policy zone WAN description "External WAN"

# Assign interfaces to zones
set zone-policy zone LAN interface eth2
set zone-policy zone DMZ interface eth3
set zone-policy zone WAN interface eth4

# Configure zone rules
set zone-policy zone LAN from DMZ firewall name DMZ-to-LAN
set zone-policy zone LAN from WAN firewall name WAN-to-LAN
set zone-policy zone DMZ from LAN firewall name LAN-to-DMZ
set zone-policy zone DMZ from WAN firewall name WAN-to-DMZ
set zone-policy zone WAN from LAN firewall name LAN-to-WAN
set zone-policy zone WAN from DMZ firewall name DMZ-to-WAN

commit
save
```

Firewall Rules

```
# Configure firewall rules
configure

# LAN to WAN (allow most traffic)
set firewall name LAN-to-WAN default-action accept
set firewall name LAN-to-WAN rule 10 action drop
set firewall name LAN-to-WAN rule 10 destination port 23,135-139,445
set firewall name LAN-to-WAN rule 10 protocol tcp
set firewall name LAN-to-WAN rule 10 description "Block dangerous ports"

# WAN to LAN (restrictive)
set firewall name WAN-to-LAN default-action drop
set firewall name WAN-to-LAN rule 10 action accept
set firewall name WAN-to-LAN rule 10 state established enable
set firewall name WAN-to-LAN rule 10 state related enable
set firewall name WAN-to-LAN rule 10 description "Allow established connections"

# WAN to DMZ (allow specific services)
set firewall name WAN-to-DMZ default-action drop
set firewall name WAN-to-DMZ rule 10 action accept
set firewall name WAN-to-DMZ rule 10 destination port 80,443
set firewall name WAN-to-DMZ rule 10 protocol tcp
set firewall name WAN-to-DMZ rule 10 description "Allow HTTP/HTTPS to DMZ"

set firewall name WAN-to-DMZ rule 20 action accept
set firewall name WAN-to-DMZ rule 20 state established enable
set firewall name WAN-to-DMZ rule 20 state related enable

# DMZ to LAN (very restrictive)
set firewall name DMZ-to-LAN default-action drop
set firewall name DMZ-to-LAN rule 10 action accept
set firewall name DMZ-to-LAN rule 10 destination address 192.168.10.100
set firewall name DMZ-to-LAN rule 10 destination port 3306
set firewall name DMZ-to-LAN rule 10 protocol tcp
set firewall name DMZ-to-LAN rule 10 description "Allow DMZ to database server"

commit
save

# Verify firewall
show firewall
show zone-policy
```

NAT Configuration

```
# Configure NAT
configure

# Source NAT (masquerade)
set nat source rule 100 outbound-interface eth4
set nat source rule 100 source address 192.168.10.0/24
set nat source rule 100 translation address masquerade
set nat source rule 100 description "LAN to WAN NAT"

set nat source rule 110 outbound-interface eth4
set nat source rule 110 source address 192.168.20.0/24
set nat source rule 110 translation address masquerade
set nat source rule 110 description "DMZ to WAN NAT"

# Destination NAT (port forwarding)
```

```
set nat destination rule 10 inbound-interface eth4
set nat destination rule 10 destination port 80
set nat destination rule 10 protocol tcp
set nat destination rule 10 translation address 192.168.20.10
set nat destination rule 10 translation port 80
set nat destination rule 10 description "HTTP to DMZ server"

set nat destination rule 20 inbound-interface eth4
set nat destination rule 20 destination port 443
set nat destination rule 20 protocol tcp
set nat destination rule 20 translation address 192.168.20.10
set nat destination rule 20 translation port 443
set nat destination rule 20 description "HTTPS to DMZ server"

commit
save

# Verify NAT
show nat source statistics
show nat destination statistics
```

VyOS VPN Configuration

IPSec Site-to-Site VPN

```
# Configure IPSec VPN
configure

# IPSec configuration
set vpn ipsec esp-group ESP-GROUP compression disable
set vpn ipsec esp-group ESP-GROUP lifetime 3600
set vpn ipsec esp-group ESP-GROUP mode tunnel
set vpn ipsec esp-group ESP-GROUP pfs dh-group2
set vpn ipsec esp-group ESP-GROUP proposal 1 encryption aes256
set vpn ipsec esp-group ESP-GROUP proposal 1 hash sha1

set vpn ipsec ike-group IKE-GROUP dead-peer-detection action restart
set vpn ipsec ike-group IKE-GROUP dead-peer-detection interval 30
set vpn ipsec ike-group IKE-GROUP dead-peer-detection timeout 120
set vpn ipsec ike-group IKE-GROUP lifetime 28800
set vpn ipsec ike-group IKE-GROUP proposal 1 dh-group 2
set vpn ipsec ike-group IKE-GROUP proposal 1 encryption aes256
set vpn ipsec ike-group IKE-GROUP proposal 1 hash sha1

# Site-to-site tunnel
```

```
set vpn ipsec site-to-site peer 203.0.113.100 authentication mode pre-shared-secret
set vpn ipsec site-to-site peer 203.0.113.100 authentication pre-shared-secret SecretVPNKey1
set vpn ipsec site-to-site peer 203.0.113.100 connection-type initiate
set vpn ipsec site-to-site peer 203.0.113.100 default-esp-group ESP-GROUP
set vpn ipsec site-to-site peer 203.0.113.100 ike-group IKE-GROUP
set vpn ipsec site-to-site peer 203.0.113.100 local-address 203.0.113.1

set vpn ipsec site-to-site peer 203.0.113.100 tunnel 1 local prefix 192.168.10.0/24
set vpn ipsec site-to-site peer 203.0.113.100 tunnel 1 remote prefix 192.168.30.0/24

commit
save

# Verify IPSec
show vpn ipsec sa
show vpn ipsec status
```

OpenVPN Configuration

```
# Configure OpenVPN server
configure

# Generate certificates (simplified for lab)
set pki ca CA certificate "-----BEGIN CERTIFICATE-----
...certificate content...
-----END CERTIFICATE-----"

set pki certificate server certificate "-----BEGIN CERTIFICATE-----
...certificate content...
-----END CERTIFICATE-----"

set pki certificate server private key "-----BEGIN PRIVATE KEY-----
...private key content...
-----END PRIVATE KEY-----"

# OpenVPN server configuration
set interfaces openvpn vtun0 mode server
set interfaces openvpn vtun0 server subnet 10.8.0.0/24
set interfaces openvpn vtun0 server push-route 192.168.10.0/24
set interfaces openvpn vtun0 server push-route 192.168.20.0/24
set interfaces openvpn vtun0 tls ca-cert-file /config/auth/ca.crt
set interfaces openvpn vtun0 tls cert-file /config/auth/server.crt
set interfaces openvpn vtun0 tls key-file /config/auth/server.key
set interfaces openvpn vtun0 tls dh-file /config/auth/dh2048.pem
```

```
commit
save

# Verify OpenVPN
show interfaces openvpn
show openvpn status server vtun0
```

WireGuard Configuration

```
# Configure WireGuard
configure

# Generate keys
run generate wireguard keypair

# WireGuard interface
set interfaces wireguard wg0 address 10.9.0.1/24
set interfaces wireguard wg0 description "WireGuard VPN"
set interfaces wireguard wg0 port 51820
set interfaces wireguard wg0 private-key "private-key-here"

# Peer configuration
set interfaces wireguard wg0 peer client1 allowed-ips 10.9.0.2/32
set interfaces wireguard wg0 peer client1 allowed-ips 192.168.40.0/24
set interfaces wireguard wg0 peer client1 public-key "client-public-key-here"

commit
save

# Verify WireGuard
show interfaces wireguard
show wireguard keypairs
```

VyOS High Availability

VRRP Configuration

```
# Configure VRRP for high availability
configure

# VRRP group
set high-availability vrrp group LAN vrid 10
set high-availability vrrp group LAN interface eth2
set high-availability vrrp group LAN virtual-address 192.168.10.1/24
set high-availability vrrp group LAN priority 200
set high-availability vrrp group LAN preempt true
set high-availability vrrp group LAN authentication type plaintext-password
set high-availability vrrp group LAN authentication password VRRPSecret123

# Sync group for multiple interfaces
set high-availability vrrp sync-group MAIN member LAN
set high-availability vrrp sync-group MAIN member DMZ

commit
save

# Verify VRRP
show vrrp
show vrrp detail
```

Configuration Synchronization

```
# Configure config sync between VRRP peers
configure

set service config-sync mode load-balance
set service config-sync secondary 192.168.10.2
set service config-sync section firewall
set service config-sync section nat
set service config-sync section vpn

commit
save
```

VyOS Load Balancing

WAN Load Balancing

```
# Configure WAN load balancing
configure

# Load balancing rules
set load-balancing wan interface-health eth3 nexthop 203.0.113.1
set load-balancing wan interface-health eth4 nexthop 198.51.100.1

set load-balancing wan rule 1 inbound-interface eth2
set load-balancing wan rule 1 interface eth3 weight 1
set load-balancing wan rule 1 interface eth4 weight 1
set load-balancing wan rule 1 protocol all
set load-balancing wan rule 1 description "Load balance LAN traffic"

# Failover configuration
set load-balancing wan interface-health eth3 failure-count 3
set load-balancing wan interface-health eth3 success-count 3
set load-balancing wan interface-health eth3 test 10 type ping
set load-balancing wan interface-health eth3 test 10 target 8.8.8.8

commit
save

# Verify load balancing
show load-balancing wan
show load-balancing wan interface-health
```

VyOS Monitoring and Troubleshooting

System Monitoring

```
# System information
show version
show system uptime
show system memory
show system storage

# Interface monitoring
show interfaces
show interfaces ethernet eth1
show interfaces statistics

# Protocol monitoring
show ip route
show ip ospf neighbor
show ip bgp summary
show vpn ipsec sa
```

Logging and Debugging

```
# Configure logging
configure

set system syslog global facility all level info
set system syslog host 192.168.10.100 facility all level info
set system syslog file /var/log/vyos.log facility all level debug

commit
save

# View logs
show log
show log tail 50
show log | match "ospf"

# Debug commands
debug ip ospf packet all
debug bgp updates
```

Performance Monitoring

```
# Monitor system performance
show system processes
show system processes extensive
show interfaces counters

# Network testing
ping 8.8.8.8 count 10
traceroute 8.8.8.8
monitor traffic interface eth1
```

VyOS Automation

Configuration Management

```
#!/usr/bin/env python3
# vyos_config_manager.py
import subprocess
import json
import time

```

```
class VyOSManager:
    def __init__(self, container_name):
        self.container_name = container_name

    def execute_command(self, command, config_mode=False):
        """Execute command on VyOS"""
        if config_mode:
            cmd = f"docker exec {self.container_name} vbash -c 'source /opt/vyatta/etc/funct
        else:
            cmd = f"docker exec {self.container_name} vbash -c 'source /opt/vyatta/etc/funct

        result = subprocess.run(cmd, shell=True, capture_output=True, text=True)
        return result.stdout, result.stderr

    def configure_interface(self, interface, address, description=None):
        """Configure interface"""
        commands = [
            f"set interfaces ethernet {interface} address {address}"
        ]

        if description:
            commands.append(f"set interfaces ethernet {interface} description '{description}

        for cmd in commands:
            stdout, stderr = self.execute_command(cmd, config_mode=True)
            if stderr:
                print(f"Error: {stderr}")

    def configure_ospf(self, router_id, networks):
        """Configure OSPF"""
        commands = [
            f"set protocols ospf parameters router-id {router_id}",
            "set protocols ospf log-adjacency-changes"
        ]

        for network in networks:
            commands.append(f"set protocols ospf area {network['area']} network {network['ne

        for cmd in commands:
            stdout, stderr = self.execute_command(cmd, config_mode=True)
            if stderr:
                print(f"Error: {stderr}")

    def get_routing_table(self):
        """Get routing table"""
        stdout, stderr = self.execute_command("show ip route")
        return stdout
```

```python
    def get_interface_status(self):
        """Get interface status"""
        stdout, stderr = self.execute_command("show interfaces")
        return stdout

# Usage example
if __name__ == '__main__':
    vyos = VyOSManager('clab-vyos-vyos-r1')

    # Configure interface
    vyos.configure_interface('eth1', '10.1.12.1/30', 'To-VyOS-R2')

    # Configure OSPF
    networks = [
        {'network': '10.1.12.0/30', 'area': '0'},
        {'network': '1.1.1.1/32', 'area': '0'}
    ]
    vyos.configure_ospf('1.1.1.1', networks)

    # Get status
    print("Routing table:")
    print(vyos.get_routing_table())
```

Ansible Integration

```yaml
# vyos_playbook.yml
---
- name: Configure VyOS Network
  hosts: vyos_routers
  gather_facts: no
  connection: network_cli
  vars:
    ansible_network_os: vyos
    ansible_user: vyos
    ansible_password: vyos

  tasks:
    - name: Configure interfaces
      vyos_interfaces:
        config:
          - name: eth1
            description: "To-Core-Network"
            enabled: true
        state: merged
    - name: Configure OSPF
      vyos_ospfv2:
        config:
          router_id: "{{ router_id }}"
          log_adjacency_changes: true
          areas:
            - area_id: "0"
              networks:
                - address: "{{ ospf_networks }}"
        state: merged

    - name: Configure firewall
      vyos_firewall_rules:
        config:
          - afi: ipv4
            rule_sets:
              - name: "LAN-to-WAN"
                default_action: accept
                rules:
                  - number: 10
                    action: drop
                    destination:
                      port: "23,135-139,445"
                    protocol: tcp
        state: merged

    - name: Save configuration
      vyos_config:
        save: true
```

VyOS Best Practices

Security Hardening

```
# Security configuration
configure

# Strong authentication
set system login user admin authentication encrypted-password '$6$rounds=656000$...'
set system login user admin level admin

# SSH hardening
set service ssh port 2222
set service ssh protocol-version v2
set service ssh client-keepalive-interval 60
```

```
# Disable unnecessary services
delete service telnet
delete service ftp

# Firewall logging
set firewall all-ping enable
set firewall broadcast-ping disable
set firewall config-trap disable
set firewall twa-hazards-protection disable

commit
save
```

Performance Optimization

```
# Performance tuning
configure

# Interface optimization
set interfaces ethernet eth1 offload gro
set interfaces ethernet eth1 offload gso
set interfaces ethernet eth1 offload sg
set interfaces ethernet eth1 offload tso

# System optimization
set system option performance throughput
set system option kernel disable-power-saving

commit
save
```

Backup and Recovery

```
# Configuration backup
show configuration commands | save /config/backup-$(date +%Y%m%d).conf

# System image backup
add system image http://example.com/vyos-image.iso

# Configuration archive
set system config-management commit-archive location 'scp://backup-server/vyos-configs'
```

Summary

VyOS provides a comprehensive, open-source network operating system with enterprise-grade features. Its unified configuration interface, extensive protocol support, and container-native design make it an excellent choice for both learning and production deployments. Understanding VyOS capabilities enables cost-effective implementation of routing, security, and VPN services.

Key concepts covered: - VyOS architecture and configuration system - Routing protocols (OSPF, BGP, static) - Zone-based firewall and NAT configuration - VPN services (IPSec, OpenVPN, WireGuard) - High availability with VRRP - Load balancing and performance optimization - Automation and management techniques

In the next chapter, we'll explore OpenWrt, a Linux-based operating system for embedded devices and wireless access points.

Review Questions

1. What are the main advantages of VyOS over traditional router operating systems?
2. How do you configure zone-based firewall policies in VyOS?
3. What VPN technologies does VyOS support and how do they differ?
4. How do you implement high availability with VRRP in VyOS?
5. What are best practices for VyOS security hardening?

Hands-on Exercises

Exercise 1: Basic VyOS Deployment

1. Deploy the VyOS network lab
2. Configure interfaces and basic routing
3. Verify connectivity and routing tables
4. Test configuration persistence

Exercise 2: Firewall and NAT Configuration

1. Configure zone-based firewall policies
2. Implement NAT rules for different scenarios
3. Test firewall rule effectiveness
4. Monitor firewall logs and statistics

Exercise 3: VPN Implementation

1. Configure IPSec site-to-site VPN
2. Set up OpenVPN server for remote access
3. Implement WireGuard for modern VPN
4. Test VPN connectivity and performance

Exercise 4: High Availability Setup

1. Configure VRRP for gateway redundancy
2. Implement configuration synchronization
3. Test failover scenarios
4. Monitor HA status and performance

Additional Resources

- VyOS Documentation
- VyOS GitHub Repository
- VyOS Community Forum
- VyOS Configuration Examples

Chapter 55: OpenWrt - Open Source Wireless and Embedded Networking

Learning Objectives

By the end of this chapter, you will be able to: - Deploy OpenWrt in ContainerLab for network simulation - Configure wireless access points and routing with OpenWrt - Implement advanced networking features using OpenWrt - Integrate OpenWrt with enterprise network infrastructure - Customize and extend OpenWrt functionality

Introduction to OpenWrt

What is OpenWrt?

OpenWrt is a Linux-based operating system targeting embedded devices, primarily wireless routers and access points. It provides a fully writable filesystem with package management, enabling users to customize the device through the use of packages to suit any application.

Key OpenWrt Features

- **Linux-based**: Full Linux distribution for embedded devices
- **Package Management**: opkg package manager with thousands of packages
- **Web Interface**: LuCI web configuration interface
- **Wireless Support**: Comprehensive 802.11 protocol support
- **Network Services**: Routing, switching, firewall, VPN, QoS
- **Extensible**: Custom applications and kernel modules
- **Container Ready**: Can run in containers for testing and development

OpenWrt Architecture

Core Components

- **Kernel**: Linux kernel optimized for embedded devices
- **Base System**: Essential system utilities and libraries
- **Network Stack**: Advanced networking capabilities
- **Wireless Stack**: mac80211 and cfg80211 wireless frameworks
- **Package System**: opkg package management
- **Web Interface**: LuCI configuration interface

- **UCI**: Unified Configuration Interface

OpenWrt Lab Environment

OpenWrt Container Lab Setup

```yaml
# OpenWrt comprehensive lab
name: openwrt-network-lab
prefix: owrt

topology:
  nodes:
    # OpenWrt routers/APs
    openwrt-main:
      kind: linux
      image: openwrt/rootfs:latest
      mgmt-ipv4: 172.20.20.10
      cmd: /sbin/init
      binds:
        - ./configs/openwrt-main:/etc/config
      env:
        - OPENWRT_HOSTNAME=openwrt-main

    openwrt-ap1:
      kind: linux
      image: openwrt/rootfs:latest
      mgmt-ipv4: 172.20.20.11
      cmd: /sbin/init
      binds:
        - ./configs/openwrt-ap1:/etc/config
      env:
        - OPENWRT_HOSTNAME=openwrt-ap1

    openwrt-ap2:
      kind: linux
      image: openwrt/rootfs:latest
      mgmt-ipv4: 172.20.20.12
      cmd: /sbin/init
      binds:
        - ./configs/openwrt-ap2:/etc/config
      env:
        - OPENWRT_HOSTNAME=openwrt-ap2

    # Core network switch
    core-switch:
```

```
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 172.20.20.13
    exec:
      - apk add --no-cache bridge-utils
      - brctl addbr br0
      - brctl stp br0 off
      - ip link set br0 up
      - ip addr add 10.1.1.1/24 dev br0

  # Client devices
  wireless-client1:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 172.20.20.20
    exec:
      - apk add --no-cache wireless-tools wpa_supplicant dhcpcd
      - ip addr add 192.168.1.100/24 dev eth1

  wireless-client2:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 172.20.20.21
    exec:
      - apk add --no-cache wireless-tools wpa_supplicant dhcpcd
      - ip addr add 192.168.2.100/24 dev eth1

  wired-client:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 172.20.20.22
    exec:
      - ip addr add 192.168.10.100/24 dev eth1
      - ip route add default via 192.168.10.1

  # Internet simulation
  internet-gw:
    kind: linux
    image: alpine:latest
    mgmt-ipv4: 172.20.20.100
    exec:
      - ip addr add 203.0.113.1/30 dev eth1
      - ip route add 192.168.0.0/16 via 203.0.113.2
      - apk add --no-cache iperf3 nginx
      - nginx

links:
  # Core network connections
  - endpoints: ["openwrt-main:eth1", "core-switch:eth1"]
  - endpoints: ["openwrt-ap1:eth1", "core-switch:eth2"]
  - endpoints: ["openwrt-ap2:eth1", "core-switch:eth3"]

  # WAN connection
  - endpoints: ["openwrt-main:eth2", "internet-gw:eth1"]

  # Wired client
  - endpoints: ["openwrt-main:eth3", "wired-client:eth1"]

  # Wireless simulation (using wired for lab)
  - endpoints: ["openwrt-ap1:eth2", "wireless-client1:eth1"]
  - endpoints: ["openwrt-ap2:eth2", "wireless-client2:eth1"]
```

OpenWrt Configuration Files

Network Configuration

```
# Create configuration directories
mkdir -p configs/openwrt-main configs/openwrt-ap1 configs/openwrt-ap2

# OpenWrt Main Router Network Configuration
cat > configs/openwrt-main/network << 'EOF'
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'fd12:3456:789a::/48'

config interface 'lan'
    option type 'bridge'
    option ifname 'eth1 eth3'
    option proto 'static'
    option ipaddr '192.168.10.1'
    option netmask '255.255.255.0'
    option ip6assign '60'

config interface 'wan'
    option ifname 'eth2'
    option proto 'dhcp'
```

```
config interface 'wan6'
    option ifname 'eth2'
    option proto 'dhcpv6'

config switch
    option name 'switch0'
    option ports '0 1 2 3 6'
    option blinkrate '2'

config switch_vlan
    option device 'switch0'
    option vlan '1'
    option ports '0 1 2 3 6t'

config switch_vlan
    option device 'switch0'
    option vlan '2'
    option ports '4 6t'
EOF

# OpenWrt AP1 Network Configuration
cat > configs/openwrt-ap1/network << 'EOF'
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
    option ipaddr '127.0.0.1'
    option netmask '255.0.0.0'

config interface 'lan'
    option ifname 'eth1 eth2'
    option type 'bridge'
    option proto 'static'
    option ipaddr '192.168.1.1'
    option netmask '255.255.255.0'
    option ip6assign '60'

config interface 'uplink'
    option ifname 'eth1'
    option proto 'static'
    option ipaddr '10.1.1.11'
    option netmask '255.255.255.0'
    option gateway '10.1.1.1'
    option dns '8.8.8.8 8.8.4.4'
EOF
```

Wireless Configuration

```
# OpenWrt Main Router Wireless Configuration
cat > configs/openwrt-main/wireless << 'EOF'
config wifi-device 'radio0'
    option type 'mac80211'
    option channel '11'
    option hwmode '11g'
    option path 'platform/10180000.wmac'
    option htmode 'HT20'
    option disabled '0'
    option country 'US'
    option txpower '20'

config wifi-iface 'default_radio0'
    option device 'radio0'
    option network 'lan'
    option mode 'ap'
    option ssid 'OpenWrt-Main'
    option encryption 'psk2'
    option key 'SecureWiFiPassword123'
    option hidden '0'
    option isolate '0'

config wifi-device 'radio1'
    option type 'mac80211'
    option channel '36'
    option hwmode '11a'
    option path 'pci0000:00/0000:00:00.0'
    option htmode 'VHT80'
    option disabled '0'
    option country 'US'
    option txpower '23'

config wifi-iface 'default_radio1'
    option device 'radio1'
    option network 'lan'
    option mode 'ap'
    option ssid 'OpenWrt-Main-5G'
    option encryption 'psk2'
    option key 'SecureWiFiPassword123'
    option hidden '0'
    option isolate '0'
EOF

# OpenWrt AP1 Wireless Configuration
```

```
cat > configs/openwrt-ap1/wireless << 'EOF'
config wifi-device 'radio0'
    option type 'mac80211'
    option channel '6'
    option hwmode '11g'
    option path 'platform/10180000.wmac'
    option htmode 'HT20'
    option disabled '0'
    option country 'US'
    option txpower '20'

config wifi-iface 'default_radio0'
    option device 'radio0'
    option network 'lan'
    option mode 'ap'
    option ssid 'OpenWrt-Guest'
    option encryption 'psk2'
    option key 'GuestWiFiPassword456'
    option hidden '0'
    option isolate '1'
EOF
```

Firewall Configuration

```
# OpenWrt Main Router Firewall Configuration
cat > configs/openwrt-main/firewall << 'EOF'
config defaults
    option syn_flood '1'
    option input 'ACCEPT'
    option output 'ACCEPT'
    option forward 'REJECT'

config zone
    option name 'lan'
    list network 'lan'
    option input 'ACCEPT'
    option output 'ACCEPT'
    option forward 'ACCEPT'

config zone
    option name 'wan'
    list network 'wan'
    list network 'wan6'
    option input 'REJECT'
    option output 'ACCEPT'
    option forward 'REJECT'
    option masq '1'
    option mtu_fix '1'

config forwarding
    option src 'lan'
    option dest 'wan'

config rule
    option name 'Allow-DHCP-Renew'
    option src 'wan'
    option proto 'udp'
    option dest_port '68'
    option target 'ACCEPT'
    option family 'ipv4'

config rule
    option name 'Allow-Ping'
    option src 'wan'
    option proto 'icmp'
    option icmp_type 'echo-request'
    option family 'ipv4'
    option target 'ACCEPT'

config rule
    option name 'Allow-IGMP'
    option src 'wan'
    option proto 'igmp'
    option family 'ipv4'
    option target 'ACCEPT'

config rule
    option name 'Allow-DHCPv6'
    option src 'wan'
    option proto 'udp'
    option src_ip 'fc00::/6'
    option dest_ip 'fc00::/6'
    option dest_port '546'
    option family 'ipv6'
    option target 'ACCEPT'

config rule
    option name 'Allow-MLD'
    option src 'wan'
    option proto 'icmp'
    option src_ip 'fe80::/10'
    option icmp_type '130/0 131/0 132/0 143/0'
    option family 'ipv6'
    option target 'ACCEPT'

config rule
    option name 'Allow-ICMPv6-Input'
    option src 'wan'
    option proto 'icmp'
    option icmp_type '128 129 135 136'
    option family 'ipv6'
    option target 'ACCEPT'

config rule
    option name 'Allow-ICMPv6-Forward'
    option src 'wan'
    option dest '*'
    option proto 'icmp'
    option icmp_type '128 129 135 136'
    option family 'ipv6'
    option target 'ACCEPT'

config include
    option path '/etc/firewall.user'
EOF
```

OpenWrt Network Configuration

Basic Network Setup

```
# Deploy OpenWrt lab
containerlab deploy -t openwrt-network-lab.yml

# Connect to OpenWrt main router
docker exec -it clab-owrt-openwrt-main sh

# Check network configuration
uci show network
cat /etc/config/network

# Configure network interface
uci set network.lan.ipaddr='192.168.10.1'
uci set network.lan.netmask='255.255.255.0'
uci commit network
/etc/init.d/network restart

# Verify network configuration
```

```
ip addr show
ip route show
```

VLAN Configuration

```
# Configure VLANs
uci set network.@switch_vlan[0].vlan='10'
uci set network.@switch_vlan[0].ports='0 1 6t'
uci add network switch_vlan
uci set network.@switch_vlan[1].device='switch0'
uci set network.@switch_vlan[1].vlan='20'
uci set network.@switch_vlan[1].ports='2 3 6t'

# Create VLAN interfaces
uci set network.vlan10=interface
uci set network.vlan10.ifname='eth0.10'
uci set network.vlan10.proto='static'
uci set network.vlan10.ipaddr='192.168.10.1'
uci set network.vlan10.netmask='255.255.255.0'

uci set network.vlan20=interface
uci set network.vlan20.ifname='eth0.20'
uci set network.vlan20.proto='static'
uci set network.vlan20.ipaddr='192.168.20.1'
uci set network.vlan20.netmask='255.255.255.0'

uci commit network
/etc/init.d/network restart

# Verify VLAN configuration
swconfig dev switch0 show
```

Bridge Configuration

```
# Configure bridge
uci set network.br_lan=interface
uci set network.br_lan.type='bridge'
uci set network.br_lan.proto='static'
uci set network.br_lan.ipaddr='192.168.1.1'
uci set network.br_lan.netmask='255.255.255.0'
uci add_list network.br_lan.ifname='eth1'
uci add_list network.br_lan.ifname='eth2'
```

```
# Bridge options
uci set network.br_lan.stp='1'
uci set network.br_lan.forward_delay='2'
uci set network.br_lan.hello_time='1'
uci set network.br_lan.max_age='10'

uci commit network
/etc/init.d/network restart

# Verify bridge
brctl show
```

OpenWrt Wireless Configuration

Access Point Configuration

```
# Configure wireless access point
uci set wireless.radio0.disabled='0'
uci set wireless.radio0.channel='11'
uci set wireless.radio0.htmode='HT20'
uci set wireless.radio0.country='US'
uci set wireless.radio0.txpower='20'

# Configure wireless interface
uci set wireless.default_radio0.ssid='OpenWrt-Lab'
uci set wireless.default_radio0.encryption='psk2'
uci set wireless.default_radio0.key='SecurePassword123'
uci set wireless.default_radio0.network='lan'
uci set wireless.default_radio0.mode='ap'
uci set wireless.default_radio0.hidden='0'

uci commit wireless
wifi reload

# Verify wireless configuration
iwconfig
iw dev
```

Guest Network Configuration

```
# Create guest network interface
uci set network.guest=interface
uci set network.guest.proto='static'
uci set network.guest.ipaddr='192.168.100.1'
uci set network.guest.netmask='255.255.255.0'

# Create guest wireless interface
uci set wireless.guest=wifi-iface
uci set wireless.guest.device='radio0'
uci set wireless.guest.mode='ap'
uci set wireless.guest.ssid='OpenWrt-Guest'
uci set wireless.guest.encryption='psk2'
uci set wireless.guest.key='GuestPassword456'
uci set wireless.guest.network='guest'
uci set wireless.guest.isolate='1'

uci commit network
uci commit wireless
/etc/init.d/network restart
wifi reload

# Configure guest network firewall
uci add firewall zone
uci set firewall.@zone[-1].name='guest'
uci add_list firewall.@zone[-1].network='guest'
uci set firewall.@zone[-1].input='REJECT'
uci set firewall.@zone[-1].output='ACCEPT'
uci set firewall.@zone[-1].forward='REJECT'

# Allow guest to WAN
uci add firewall forwarding
uci set firewall.@forwarding[-1].src='guest'
uci set firewall.@forwarding[-1].dest='wan'

# Block guest to LAN
uci add firewall rule
uci set firewall.@rule[-1].name='Block-Guest-to-LAN'
uci set firewall.@rule[-1].src='guest'
uci set firewall.@rule[-1].dest='lan'
uci set firewall.@rule[-1].target='REJECT'

uci commit firewall
/etc/init.d/firewall restart
```

Wireless Security

```
# Configure WPA3 security
uci set wireless.default_radio0.encryption='sae'
uci set wireless.default_radio0.key='SecureWPA3Password123'

# Configure enterprise security (WPA2-Enterprise)
uci set wireless.default_radio0.encryption='wpa2'
uci set wireless.default_radio0.server='192.168.1.100'
uci set wireless.default_radio0.port='1812'
uci set wireless.default_radio0.key='RadiusSecret123'

# MAC address filtering
uci set wireless.default_radio0.macfilter='allow'
uci add_list wireless.default_radio0.maclist='aa:bb:cc:dd:ee:ff'
uci add_list wireless.default_radio0.maclist='11:22:33:44:55:66'

uci commit wireless
wifi reload
```

OpenWrt Routing Configuration

Static Routing

```
# Configure static routes
uci add network route
uci set network.@route[-1].interface='wan'
uci set network.@route[-1].target='192.168.200.0'
uci set network.@route[-1].netmask='255.255.255.0'
uci set network.@route[-1].gateway='203.0.113.1'

# Policy-based routing
uci add network rule
uci set network.@rule[-1].src='192.168.10.0/24'
uci set network.@rule[-1].lookup='100'

uci add network route
uci set network.@route[-1].interface='wan'
uci set network.@route[-1].target='0.0.0.0'
uci set network.@route[-1].netmask='0.0.0.0'
uci set network.@route[-1].gateway='203.0.113.1'
uci set network.@route[-1].table='100'

uci commit network
```

```
/etc/init.d/network restart

# Verify routing
ip route show
ip rule show
```

Dynamic Routing with BIRD

```
# Install BIRD routing daemon
opkg update
opkg install bird2 bird2-client

# Configure BIRD for OSPF
cat > /etc/bird.conf << 'EOF'
log syslog all;

router id 1.1.1.1;

protocol device {
    scan time 10;
}

protocol kernel {
    ipv4 {
        import none;
        export all;
    };
}

protocol static {
    ipv4;
    route 192.168.10.0/24 via "br-lan";
}

protocol ospf v2 {
    ipv4 {
        import all;
        export all;
    };

    area 0.0.0.0 {
        interface "eth1" {
            cost 10;
            hello 5;
            dead 20;
        };
        interface "br-lan" {
            stub yes;
        };
    };
}
EOF

# Start BIRD
/etc/init.d/bird enable
/etc/init.d/bird start

# Verify BIRD
birdc show protocols
birdc show route
```

OpenWrt Services Configuration

DHCP Server

```
# Configure DHCP server
uci set dhcp.lan.start='100'
uci set dhcp.lan.limit='150'
uci set dhcp.lan.leasetime='12h'
uci add_list dhcp.lan.dhcp_option='6,8.8.8.8,8.8.4.4'
uci add_list dhcp.lan.dhcp_option='3,192.168.10.1'

# Static DHCP reservations
uci add dhcp host
uci set dhcp.@host[-1].name='server1'
uci set dhcp.@host[-1].mac='aa:bb:cc:dd:ee:ff'
uci set dhcp.@host[-1].ip='192.168.10.100'

# DHCP for guest network
uci set dhcp.guest=dhcp
uci set dhcp.guest.interface='guest'
uci set dhcp.guest.start='50'
uci set dhcp.guest.limit='50'
uci set dhcp.guest.leasetime='2h'

uci commit dhcp
/etc/init.d/dnsmasq restart

# Verify DHCP
cat /var/dhcp.leases
```

DNS Configuration

```
# Configure DNS
uci add_list dhcp.@dnsmasq[0].server='8.8.8.8'
uci add_list dhcp.@dnsmasq[0].server='1.1.1.1'
uci set dhcp.@dnsmasq[0].domain='local.lan'
uci set dhcp.@dnsmasq[0].local='/local.lan/'

# DNS filtering
uci add_list dhcp.@dnsmasq[0].address='/ads.example.com/127.0.0.1'
uci add_list dhcp.@dnsmasq[0].address='/malware.example.com/127.0.0.1'

# Custom DNS entries
echo "192.168.10.100 server1.local.lan server1" >> /etc/hosts

uci commit dhcp
/etc/init.d/dnsmasq restart

# Verify DNS
nslookup server1.local.lan
dig @127.0.0.1 google.com
```

VPN Services

OpenVPN Server

```
# Install OpenVPN
opkg update
opkg install openvpn-openssl openvpn-easy-rsa luci-app-openvpn

# Generate certificates
cd /etc/easy-rsa
./easyrsa init-pki
./easyrsa build-ca nopass
./easyrsa gen-req server nopass
./easyrsa sign-req server server
./easyrsa gen-dh

# Configure OpenVPN server
cat > /etc/openvpn/server.conf << 'EOF'
port 1194
proto udp
dev tun
ca /etc/easy-rsa/pki/ca.crt
cert /etc/easy-rsa/pki/issued/server.crt
```

```
key /etc/easy-rsa/pki/private/server.key
dh /etc/easy-rsa/pki/dh.pem
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
push "route 192.168.10.0 255.255.255.0"
push "dhcp-option DNS 192.168.10.1"
keepalive 10 120
comp-lzo
persist-key
persist-tun
status openvpn-status.log
verb 3
EOF

# Start OpenVPN
/etc/init.d/openvpn enable
/etc/init.d/openvpn start

# Configure firewall for VPN
uci add firewall rule
uci set firewall.@rule[-1].name='Allow-OpenVPN'
uci set firewall.@rule[-1].src='wan'
uci set firewall.@rule[-1].dest_port='1194'
uci set firewall.@rule[-1].proto='udp'
uci set firewall.@rule[-1].target='ACCEPT'

uci commit firewall
/etc/init.d/firewall restart
```

WireGuard VPN

```
# Install WireGuard
opkg update
opkg install wireguard-tools kmod-wireguard luci-app-wireguard

# Generate keys
wg genkey | tee /etc/wireguard/server_private.key | wg pubkey > /etc/wireguard/server_public
wg genkey | tee /etc/wireguard/client_private.key | wg pubkey > /etc/wireguard/client_public

# Configure WireGuard interface
uci set network.wg0=interface
uci set network.wg0.proto='wireguard'
uci set network.wg0.private_key="$(cat /etc/wireguard/server_private.key)"
uci add_list network.wg0.addresses='10.9.0.1/24'
```

```
# Add WireGuard peer
uci add network wireguard_wg0
uci set network.@wireguard_wg0[-1].public_key="$(cat /etc/wireguard/client_public.key)"
uci add_list network.@wireguard_wg0[-1].allowed_ips='10.9.0.2/32'
uci add_list network.@wireguard_wg0[-1].allowed_ips='192.168.100.0/24'

uci commit network
/etc/init.d/network restart

# Configure firewall
uci add firewall zone
uci set firewall.@zone[-1].name='wg'
uci add_list firewall.@zone[-1].network='wg0'
uci set firewall.@zone[-1].input='ACCEPT'
uci set firewall.@zone[-1].output='ACCEPT'
uci set firewall.@zone[-1].forward='ACCEPT'

uci add firewall forwarding
uci set firewall.@forwarding[-1].src='wg'
uci set firewall.@forwarding[-1].dest='lan'

uci commit firewall
/etc/init.d/firewall restart
```

OpenWrt Quality of Service

Traffic Shaping

```
# Install QoS packages
opkg update
opkg install tc kmod-sched-core kmod-ifb luci-app-qos

# Configure QoS
uci set qos.wan=interface
uci set qos.wan.classgroup='Default'
uci set qos.wan.enabled='1'
uci set qos.wan.upload='1000'
uci set qos.wan.download='10000'

# QoS rules
uci add qos rule
uci set qos.@rule[-1].target='Priority'
uci set qos.@rule[-1].proto='tcp'
uci set qos.@rule[-1].ports='22,53,80,443'
uci add qos rule
uci set qos.@rule[-1].target='Express'
uci set qos.@rule[-1].proto='udp'
uci set qos.@rule[-1].ports='53,123'

uci add qos rule
uci set qos.@rule[-1].target='Bulk'
uci set qos.@rule[-1].proto='tcp'
uci set qos.@rule[-1].ports='20,21,25,110,143,993,995'

uci commit qos
/etc/init.d/qos enable
/etc/init.d/qos start

# Verify QoS
tc qdisc show
tc class show dev eth2
```

Bandwidth Monitoring

```
# Install monitoring tools
opkg install luci-app-statistics collectd-mod-interface collectd-mod-iwinfo

# Configure statistics collection
uci set luci_statistics.collectd.Hostname='OpenWrt-Main'
uci set luci_statistics.collectd.BaseDir='/tmp/rrd'
uci set luci_statistics.collectd.Include='/etc/collectd/conf.d'
uci set luci_statistics.collectd.PIDFile='/var/run/collectd.pid'
uci set luci_statistics.collectd.PluginDir='/usr/lib/collectd'
uci set luci_statistics.collectd.TypesDB='/usr/share/collectd/types.db'
uci set luci_statistics.collectd.Interval='60'
uci set luci_statistics.collectd.ReadThreads='2'

# Enable interface monitoring
uci set luci_statistics.collectd_interface.enable='1'
uci add_list luci_statistics.collectd_interface.Interfaces='br-lan'
uci add_list luci_statistics.collectd_interface.Interfaces='eth2'

uci commit luci_statistics
/etc/init.d/luci_statistics enable
/etc/init.d/luci_statistics start
```

OpenWrt Security Features

Firewall Advanced Configuration

```
# Advanced firewall rules
uci add firewall rule
uci set firewall.@rule[-1].name='Block-Tor'
uci set firewall.@rule[-1].src='lan'
uci set firewall.@rule[-1].dest='wan'
uci set firewall.@rule[-1].dest_port='9001,9030'
uci set firewall.@rule[-1].target='REJECT'

# Rate limiting
uci add firewall rule
uci set firewall.@rule[-1].name='SSH-Rate-Limit'
uci set firewall.@rule[-1].src='wan'
uci set firewall.@rule[-1].dest_port='22'
uci set firewall.@rule[-1].proto='tcp'
uci set firewall.@rule[-1].extra='--limit 3/min --limit-burst 5'
uci set firewall.@rule[-1].target='ACCEPT'

# Port knocking
uci add firewall rule
uci set firewall.@rule[-1].name='Port-Knock-1'
uci set firewall.@rule[-1].src='wan'
uci set firewall.@rule[-1].dest_port='1234'
uci set firewall.@rule[-1].proto='tcp'
uci set firewall.@rule[-1].extra='-m recent --name knock1 --set'
uci set firewall.@rule[-1].target='DROP'

uci commit firewall
/etc/init.d/firewall restart
```

Intrusion Detection

```
# Install Suricata IDS
opkg update
opkg install suricata

# Configure Suricata
cat > /etc/suricata/suricata.yaml << 'EOF'
vars:
  address-groups:
    HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"
    EXTERNAL_NET: "!$HOME_NET"
```

```
af-packet:
  - interface: eth2
    cluster-id: 99
    cluster-type: cluster_flow

rule-files:
  - suricata.rules

logging:
  default-log-level: notice
  outputs:
    - console:
        enabled: yes
    - file:
        enabled: yes
        filename: /var/log/suricata/suricata.log
    - syslog:
        enabled: yes
        facility: local5
        format: "[%i] <%d> -- "
EOF

# Start Suricata
/etc/init.d/suricata enable
/etc/init.d/suricata start

# Monitor alerts
tail -f /var/log/suricata/fast.log
```

OpenWrt Monitoring and Management

System Monitoring

```
# System information
cat /proc/version
cat /proc/cpuinfo
cat /proc/meminfo
df -h

# Network monitoring
ip addr show
ip route show
iptables -L -n
iwconfig
```

```
# Process monitoring
top
ps aux
netstat -tulpn
```

Remote Management

```
# SSH configuration
uci set dropbear.@dropbear[0].Port='2222'
uci set dropbear.@dropbear[0].PasswordAuth='off'
uci set dropbear.@dropbear[0].RootPasswordAuth='off'

# Add SSH key
echo "ssh-rsa AAAAB3NzaC1yc2E... user@host" >> /etc/dropbear/authorized_keys

uci commit dropbear
/etc/init.d/dropbear restart

# SNMP configuration
opkg install snmpd
uci set snmpd.agent.agentaddress='UDP:161'
uci set snmpd.com2sec.secname='ro'
uci set snmpd.com2sec.source='default'
uci set snmpd.com2sec.community='public'

uci commit snmpd
/etc/init.d/snmpd enable
/etc/init.d/snmpd start
```

Backup and Recovery

```
# Configuration backup
sysupgrade -b /tmp/backup-$(date +%Y%m%d).tar.gz

# System backup
tar -czf /tmp/system-backup.tar.gz /etc /root

# Restore configuration
sysupgrade -r /tmp/backup-20231201.tar.gz

# Factory reset
firstboot && reboot
```

OpenWrt Automation

UCI Scripting

```
#!/bin/sh
# openwrt_config_script.sh

# Function to configure basic network
configure_network() {
    local lan_ip="$1"
    local lan_mask="$2"

    uci set network.lan.ipaddr="$lan_ip"
    uci set network.lan.netmask="$lan_mask"
    uci commit network
    /etc/init.d/network restart
}

# Function to configure wireless
configure_wireless() {
    local ssid="$1"
    local password="$2"
    local channel="$3"

    uci set wireless.radio0.disabled='0'
    uci set wireless.radio0.channel="$channel"
    uci set wireless.default_radio0.ssid="$ssid"
    uci set wireless.default_radio0.encryption='psk2'
    uci set wireless.default_radio0.key="$password"
    uci commit wireless
    wifi reload
}

# Function to configure firewall rule
add_firewall_rule() {
    local name="$1"
    local src="$2"
    local dest_port="$3"
    local proto="$4"
    local target="$5"

    uci add firewall rule
    uci set firewall.@rule[-1].name="$name"
    uci set firewall.@rule[-1].src="$src"
    uci set firewall.@rule[-1].dest_port="$dest_port"
    uci set firewall.@rule[-1].proto="$proto"
    uci set firewall.@rule[-1].target="$target"
    uci commit firewall
    /etc/init.d/firewall restart
}

# Usage examples
configure_network "192.168.1.1" "255.255.255.0"
configure_wireless "MyNetwork" "SecurePassword123" "11"
add_firewall_rule "Allow-HTTP" "wan" "80" "tcp" "ACCEPT"
```

API Integration

```
#!/usr/bin/env python3
# openwrt_api_client.py
import requests
import json
import base64

class OpenWrtAPI:
    def __init__(self, host, username, password):
        self.host = host
        self.username = username
        self.password = password
        self.session = requests.Session()
        self.token = None
        self.login()

    def login(self):
        """Login to OpenWrt LuCI"""
        auth_data = {
            'luci_username': self.username,
            'luci_password': self.password
        }

        response = self.session.post(
            f"http://{self.host}/cgi-bin/luci/admin/uci",
            data=auth_data
        )

        if response.status_code == 200:
            print("Login successful")
        else:
            print("Login failed")

    def uci_get(self, config, section=None, option=None):
        """Get UCI configuration"""
        url = f"http://{self.host}/cgi-bin/luci/admin/uci/{config}"
        if section:
            url += f"/{section}"
        if option:
            url += f"/{option}"

        response = self.session.get(url)
        return response.json() if response.status_code == 200 else None

    def uci_set(self, config, section, option, value):
        """Set UCI configuration"""
        data = {
            'config': config,
            'section': section,
            'option': option,
            'value': value
        }

        response = self.session.post(
            f"http://{self.host}/cgi-bin/luci/admin/uci",
            data=data
        )

        return response.status_code == 200

    def uci_commit(self, config):
        """Commit UCI changes"""
        data = {'config': config}
        response = self.session.post(
            f"http://{self.host}/cgi-bin/luci/admin/uci/commit",
            data=data
        )
        return response.status_code == 200

    def get_system_info(self):
        """Get system information"""
        response = self.session.get(f"http://{self.host}/cgi-bin/luci/admin/status/overview"
        return response.json() if response.status_code == 200 else None

# Usage example
if __name__ == '__main__':
    api = OpenWrtAPI('192.168.1.1', 'root', 'password')

    # Get network configuration
    network_config = api.uci_get('network')
    print("Network config:", json.dumps(network_config, indent=2))
    # Set wireless SSID
    api.uci_set('wireless', 'default_radio0', 'ssid', 'NewSSID')
    api.uci_commit('wireless')

    # Get system info
    system_info = api.get_system_info()
    print("System info:", json.dumps(system_info, indent=2))
```

OpenWrt Best Practices

Security Hardening

```
# Change default passwords
passwd root

# Disable WPS
uci set wireless.radio0.wps_pushbutton='0'
uci commit wireless

# Enable strong wireless encryption
uci set wireless.default_radio0.encryption='sae'
uci commit wireless

# Disable unnecessary services
/etc/init.d/uhttpd disable
/etc/init.d/telnet disable

# Configure automatic updates
opkg update
opkg install auc
echo "0 4 * * * /usr/bin/auc -c" >> /etc/crontabs/root
```

Performance Optimization

```
# Optimize wireless settings
uci set wireless.radio0.txpower='20'
uci set wireless.radio0.htmode='HT40'
uci set wireless.radio0.noscan='1'

# Optimize network buffers
echo 'net.core.rmem_max = 134217728' >> /etc/sysctl.conf
echo 'net.core.wmem_max = 134217728' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_rmem = 4096 87380 134217728' >> /etc/sysctl.conf

# Enable hardware acceleration
uci set network.@device[0].multicast_querier='1'
uci set network.@device[0].igmp_snooping='1'
```

Maintenance Procedures

```
# Regular maintenance script
#!/bin/sh
# maintenance.sh

# Update package lists
opkg update

# Clean temporary files
rm -rf /tmp/*

# Rotate logs
logrotate /etc/logrotate.conf

# Check filesystem
fsck.ext4 -f /dev/sda1

# Restart services if needed
/etc/init.d/network restart
/etc/init.d/firewall restart
/etc/init.d/dnsmasq restart

echo "Maintenance completed at $(date)"
```

Summary

OpenWrt provides a powerful, flexible platform for network infrastructure with extensive customization capabilities. Its open-source nature, comprehensive package ecosystem, and container compatibility make it an excellent choice for learning, prototyping, and production deployments in various network scenarios.

Key concepts covered: - OpenWrt architecture and configuration system (UCI) - Network and wireless configuration - Routing protocols and services - Security features and VPN services - Quality of Service and monitoring - Automation and API integration - Best practices for deployment and maintenance

In the next chapter, we'll explore Mininet, a network emulator that creates realistic virtual networks for research and education.

Review Questions

1. What are the main advantages of OpenWrt over commercial router firmware?
2. How does the UCI configuration system work in OpenWrt?
3. What wireless security options are available in OpenWrt?
4. How do you configure VPN services on OpenWrt?
5. What are best practices for OpenWrt security hardening?

Hands-on Exercises

Exercise 1: Basic OpenWrt Deployment

1. Deploy the OpenWrt network lab
2. Configure basic network and wireless settings
3. Set up DHCP and DNS services
4. Test connectivity and wireless functionality

Exercise 2: Advanced Networking Features

1. Configure VLANs and bridging
2. Set up guest networks with isolation
3. Implement QoS and traffic shaping
4. Configure dynamic routing with BIRD

Exercise 3: Security Implementation

1. Configure zone-based firewall
2. Set up VPN services (OpenVPN and WireGuard)
3. Implement intrusion detection
4. Apply security hardening measures

Exercise 4: Automation and Management

1. Create UCI configuration scripts
2. Develop API integration tools
3. Implement monitoring and alerting
4. Build automated deployment procedures

Additional Resources

- OpenWrt Official Documentation
- OpenWrt Package Repository
- OpenWrt Forum and Community
- UCI Configuration System

Chapter 56: Mininet - Network Emulation and SDN Testing

Learning Objectives

By the end of this chapter, you will be able to: - Deploy Mininet for network emulation in ContainerLab - Create custom network topologies with Mininet - Integrate OpenFlow controllers with Mininet networks - Implement Software-Defined Networking (SDN) concepts - Develop and test network applications using Mininet

Introduction to Mininet

What is Mininet?

Mininet is a network emulator that creates a realistic virtual network, running real kernel, switch, and application code, on a single machine. It's particularly useful for developing, testing, and researching Software-Defined Networking (SDN) applications and OpenFlow controllers.

Key Mininet Features

- **Realistic Network Emulation**: Real Linux network stack
- **Scalable**: Supports hundreds of hosts and switches
- **OpenFlow Support**: Native OpenFlow switch support
- **Programmable**: Python API for custom topologies
- **Interactive**: Command-line interface for network interaction
- **Reproducible**: Consistent network behavior
- **Container Integration**: Works well with ContainerLab

Mininet Architecture

Core Components

- **Hosts**: Linux network namespaces acting as end hosts
- **Switches**: Software switches (Open vSwitch by default)
- **Links**: Virtual Ethernet pairs connecting components
- **Controllers**: OpenFlow controllers for SDN functionality
- **CLI**: Interactive command-line interface
- **API**: Python API for programmatic control

Summary

Mininet provides a powerful platform for network emulation and SDN development. Its ability to create realistic virtual networks with real kernel and application code makes it invaluable for research, education, and development. Integration with ContainerLab extends its capabilities for comprehensive network testing and validation.

Key concepts covered: - Mininet architecture and components - Custom topology development - OpenFlow controller integration - SDN application development - Performance testing and optimization - Network failure simulation - Hybrid topology integration - Testing and validation frameworks

Additional Resources

- Mininet Official Documentation
- Mininet GitHub Repository
- OpenFlow Specification
- Ryu SDN Framework
- ONOS SDN Controller

Interactive

Chapter 61: Interactive Lab Exercises and Simulations

Learning Objectives

By the end of this chapter, you will be able to: - Access and use interactive ContainerLab simulations - Complete guided lab exercises with real-time feedback - Use web-based network topology visualizers - Participate in collaborative network design exercises

Interactive Lab Platform

Web-Based ContainerLab Interface

```
<!-- Interactive lab launcher -->
<!DOCTYPE html>
<html>
<head>
    <title>ContainerLab Interactive Labs</title>
    <script src="https://unpkg.com/vis-network/standalone/umd/vis-network.min.js"></script>
</head>
<body>
    <div id="lab-selector">
        <h2>Choose Your Lab Exercise</h2>
        <button onclick="launchLab('basic-switching')">Basic Switching Lab</button>
        <button onclick="launchLab('ospf-routing')">OSPF Routing Lab</button>
        <button onclick="launchLab('bgp-peering')">BGP Peering Lab</button>
    </div>

    <div id="topology-viewer"></div>
    <div id="lab-instructions"></div>
    <div id="terminal-interface"></div>

    <script>
        function launchLab(labType) {
            // Launch ContainerLab topology
            fetch(`/api/labs/${labType}/start`, {method: 'POST'})
                .then(response => response.json())
                .then(data => {
                    displayTopology(data.topology);
                    loadInstructions(labType);
                    connectTerminal(data.containers);
                });
        }

        function displayTopology(topology) {
            const container = document.getElementById('topology-viewer');
            const data = {
                nodes: topology.nodes,
                edges: topology.links
            };
            const options = {
                physics: {enabled: true},
                interaction: {hover: true}
            };
            new vis.Network(container, data, options);
        }
    </script>
</body>
</html>
```

Guided Lab Exercises

```
#!/usr/bin/env python3
# Interactive lab guide system

class LabGuide:
    def __init__(self, lab_name):
        self.lab_name = lab_name
        self.current_step = 0
        self.steps = self.load_lab_steps()

    def load_lab_steps(self):
        """Load lab steps from configuration"""
        lab_configs = {
            'basic-switching': [
                {
                    'title': 'Deploy the Lab',
                    'instruction': 'Run: containerlab deploy -t basic-switching.yml',
                    'validation': self.validate_deployment,
                    'hint': 'Make sure Docker is running and you have the topology file'
                },
                {
                    'title': 'Configure VLANs',
```

```python
                    'instruction': 'Create VLANs 10 and 20 on switch1',
                    'validation': self.validate_vlans,
                    'hint': 'Use: vlan 10 name USERS'
                },
                {
                    'title': 'Test Connectivity',
                    'instruction': 'Verify hosts can communicate within same VLAN',
                    'validation': self.validate_connectivity,
                    'hint': 'Use ping between hosts in same VLAN'
                }
            ]
        }
        return lab_configs.get(self.lab_name, [])

    def get_current_step(self):
        """Get current step information"""
        if self.current_step < len(self.steps):
            return self.steps[self.current_step]
        return None

    def validate_step(self):
        """Validate current step completion"""
        step = self.get_current_step()
        if step and step['validation']():
            self.current_step += 1
            return True
        return False

    def validate_deployment(self):
        """Validate lab deployment"""
        import subprocess
        try:
            result = subprocess.run(['containerlab', 'inspect'],
                                  capture_output=True, text=True)
            return 'basic-switching' in result.stdout
        except:
            return False

    def validate_vlans(self):
        """Validate VLAN configuration"""
        # Implementation for VLAN validation
        return True

    def validate_connectivity(self):
        """Validate network connectivity"""
        # Implementation for connectivity validation
        return True
```

```
# Usage in web interface
lab_guide = LabGuide('basic-switching')
current_step = lab_guide.get_current_step()
```

Summary

Interactive learning components enhance the educational experience by providing: - Real-time feedback and validation - Visual topology representation - Guided step-by-step exercises - Collaborative learning opportunities - Progress tracking and assessment

These features make the learning process more engaging and effective for students at all levels.