CANONICAL LIVING OVERVIEW GUIDE

Segment Routing, SR-MPLS, and SRv6 Engineering

A modern transport guide covering segment-routing architecture, SRGB and SIDs, SR-MPLS, SRv6, network programming, SR Policies, steering, PCE and BGP control, TI-LFA, security, telemetry, interworking, and migration.

PERMANENT ID
MYTHOS-FG-NET-0007

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Segment Routing, SR-MPLS, and SRv6 Engineering

Document ID
MYTHOS-FG-NET-0007

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-NET - Network Engineering & Architecture
Secondary domain	MYTHOS-SEC; MYTHOS-CLD; MYTHOS-EMT
Audience	Service-provider, data-center, WAN, cloud, network automation, and future-network engineers.
Prerequisites	Strong IP routing knowledge and familiarity with MPLS, BGP, and link-state IGPs.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Explain segment lists, SIDs, forwarding, steering, and policy across SR-MPLS and SRv6.

Design locator, SID, SRGB, and policy allocation with operational ownership and scale.

Build constraint-aware paths and fast reroute without per-flow signaling in the core.

Diagnose SR control-plane, policy, data-plane, MTU, security, and interoperability faults.

Plan safe migration from LDP or RSVP-TE while preserving services and rollback.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Segment-routing architecture

Understand source-routed instruction lists and distributed segment semantics.

Chapter operating position
Understand source-routed instruction lists and distributed segment semantics.

1.1 Segments, SIDs, and segment lists

Segments, SIDs, and segment lists is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Represent topology or service instructions as ordered identifiers and understand active-segment processing. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for segments, sids, and segment lists.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating segments, sids, and segment lists as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for segments, sids, and segment lists.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

1.2 Global and local significance

Global and local significance is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Differentiate globally meaningful prefix or node SIDs from locally meaningful adjacency or binding SIDs. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for global and local significance.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Implementation sketch

```
sr_policy:
  color: 100
  endpoint: 2001:db8:ffff::9
  binding_sid: 16050
  candidate_paths:
    - preference: 200
      source: pce
      constraints:
        max_latency_ms: 20
        exclude_affinity: maintenance
    - preference: 100
      source: dynamic
validation:
  - active_candidate_path
  - steering_matches_color
  - segment_list_within_hardware_limit
  - protected_failure_path
  - mtu_and_security_boundary
```

Failure patterns

- Treating global and local significance as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for global and local significance.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

1.3 Control-plane distribution

Control-plane distribution is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Advertise capabilities and SIDs through OSPF, IS-IS, BGP, controllers, or policy systems. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing,

tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for control-plane distribution.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating control-plane distribution as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for control-plane distribution.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

1.4 Steering and policy

Steering and policy is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Classify traffic into an SR Policy by destination, color, service, application, or explicit rule. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

Document the authoritative design intent, ownership, and scope for steering and policy.

Predict protocol and forwarding state before implementation or troubleshooting.

Validate control-plane state, installed forwarding state, and real traffic independently.

Test normal, boundary, degraded, failed, recovery, and rollback conditions.

Retain packet, configuration, counter, log, topology, timing, and service evidence.

Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

Treating steering and policy as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for steering and policy.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

Chapter 01 laboratory and review

Practical laboratory
Build a small SR domain, assign node and adjacency SIDs, calculate segment lists for shortest and constrained paths, and prove forwarding.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore intent-aware multi-domain paths, programmable service chains, and machine-verifiable segment semantics.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

SR-MPLS data plane

Apply segment routing to existing MPLS forwarding hardware and operations.

Chapter operating position
Apply segment routing to existing MPLS forwarding hardware and operations.

2.1 SRGB and prefix SIDs

SRGB and prefix SIDs is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Plan label ranges, index values, consistency, allocation, and collision prevention. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for srgb and prefix sids.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating srgb and prefix sids as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for srgb and prefix sids.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

2.2 Adjacency, anycast, and binding SIDs

Adjacency, anycast, and binding SIDs is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Use local instructions, redundancy, path compression, policy indirection, and operational ownership. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for adjacency, anycast, and binding sids.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Implementation sketch

```
sr_policy:
  color: 100
  endpoint: 2001:db8:ffff::9
  binding_sid: 16050
  candidate_paths:
    - preference: 200
      source: pce
      constraints:
        max_latency_ms: 20
        exclude_affinity: maintenance
    - preference: 100
      source: dynamic
validation:
  - active_candidate_path
  - steering_matches_color
  - segment_list_within_hardware_limit
  - protected_failure_path
  - mtu_and_security_boundary
```

Failure patterns

- Treating adjacency, anycast, and binding sids as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for adjacency, anycast, and binding sids.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

2.3 Label-stack processing

Label-stack processing is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Trace push, active label, swap or pop behavior, penultimate-hop behavior, and service labels. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing,

tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for label-stack processing.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating label-stack processing as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for label-stack processing.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

2.4 IGP extensions and topology

IGP extensions and topology is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Carry algorithms, metrics, capabilities, SIDs, and constraints in the link-state database. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

Document the authoritative design intent, ownership, and scope for igp extensions and topology.

Predict protocol and forwarding state before implementation or troubleshooting.

Validate control-plane state, installed forwarding state, and real traffic independently.

Test normal, boundary, degraded, failed, recovery, and rollback conditions.

Retain packet, configuration, counter, log, topology, timing, and service evidence.

Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

Treating igp extensions and topology as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for igp extensions and topology.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

Chapter 02 laboratory and review

Practical laboratory
Deploy SR-MPLS with prefix and adjacency SIDs, force a non-shortest path, test anycast and binding SIDs, and inspect the label stack at each hop.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate deep-stack limits, compressed instruction models, hardware offload, and co-designed optical transport.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

SRv6 architecture and network programming

Use IPv6 addresses as locators and executable network behaviors.

Chapter operating position
Use IPv6 addresses as locators and executable network behaviors.

3.1 SRH and segment processing

SRH and segment processing is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Interpret segment list, segments-left, flags, TLVs, destination changes, and policy boundaries. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for srh and segment processing.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating srh and segment processing as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for srh and segment processing.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

3.2 Locator and function design

Locator and function design is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Allocate locator blocks, function bits, arguments, aggregation, ownership, and route advertisement. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for locator and function design.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Implementation sketch

```
sr_policy:
  color: 100
  endpoint: 2001:db8:ffff::9
  binding_sid: 16050
  candidate_paths:
    - preference: 200
      source: pce
      constraints:
        max_latency_ms: 20
        exclude_affinity: maintenance
    - preference: 100
      source: dynamic
validation:
  - active_candidate_path
  - steering_matches_color
  - segment_list_within_hardware_limit
  - protected_failure_path
  - mtu_and_security_boundary
```

Failure patterns

- Treating locator and function design as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for locator and function design.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

3.3 Endpoint behaviors

Endpoint behaviors is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Understand End, End.X, decapsulation, lookup, cross-connect, binding, and service behaviors. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing,

tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for endpoint behaviors.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating endpoint behaviors as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for endpoint behaviors.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

3.4 Reduced encapsulation and insertion choices

Reduced encapsulation and insertion choices is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Compare encapsulation, inline or insertion constraints, header overhead, and endpoint capability. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

Document the authoritative design intent, ownership, and scope for reduced encapsulation and insertion choices.

Predict protocol and forwarding state before implementation or troubleshooting.

Validate control-plane state, installed forwarding state, and real traffic independently.

Test normal, boundary, degraded, failed, recovery, and rollback conditions.

Retain packet, configuration, counter, log, topology, timing, and service evidence.

Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

Treating reduced encapsulation and insertion choices as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for reduced encapsulation and insertion choices.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

Chapter 03 laboratory and review

Practical laboratory
Build an SRv6 lab with transit and service behaviors, trace SRH changes, test decapsulation and VPN lookup, and calculate overhead.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore compressed SRv6 SIDs, micro-segments, programmable compute services, and 6G network slicing.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

SR Policies and candidate paths

Model explicit intent, path preference, constraints, and operational state.

Chapter operating position
Model explicit intent, path preference, constraints, and operational state.

4.1 Color, endpoint, and policy identity

Color, endpoint, and policy identity is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Define headend, endpoint, color, distinguisher, binding SID, and policy lifecycle. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for color, endpoint, and policy identity.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating color, endpoint, and policy identity as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for color, endpoint, and policy identity.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

4.2 Candidate paths and preference

Candidate paths and preference is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Combine explicit, dynamic, PCE, BGP, and local candidate paths with deterministic selection. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for candidate paths and preference.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Implementation sketch

```
sr_policy:
  color: 100
  endpoint: 2001:db8:ffff::9
  binding_sid: 16050
  candidate_paths:
    - preference: 200
      source: pce
      constraints:
        max_latency_ms: 20
        exclude_affinity: maintenance
    - preference: 100
      source: dynamic
validation:
  - active_candidate_path
  - steering_matches_color
  - segment_list_within_hardware_limit
  - protected_failure_path
  - mtu_and_security_boundary
```

Failure patterns

Treating candidate paths and preference as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for candidate paths and preference.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

4.3 Segment-list optimization

Segment-list optimization is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Balance intent, stack depth, resilience, fate sharing, ECMP, and hardware limits. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing,

tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for segment-list optimization.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating segment-list optimization as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for segment-list optimization.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

4.4 Traffic steering

Traffic steering is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Use route policy, color communities, tunnel resolution, service routes, and application classification. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

Document the authoritative design intent, ownership, and scope for traffic steering.

Predict protocol and forwarding state before implementation or troubleshooting.

Validate control-plane state, installed forwarding state, and real traffic independently.

Test normal, boundary, degraded, failed, recovery, and rollback conditions.

Retain packet, configuration, counter, log, topology, timing, and service evidence.

Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

Treating traffic steering as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for traffic steering.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

Chapter 04 laboratory and review

Practical laboratory
Create two SR Policies with multiple candidate paths, fail the preferred path, observe fallback, and validate traffic steering and binding SID behavior.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore BGP-distributed SR Policies, intent-aware routing, closed-loop optimization, and independently verified candidate paths.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Traffic engineering and controllers

Use topology, constraints, telemetry, and computation to create purposeful transport paths.

Chapter operating position
Use topology, constraints, telemetry, and computation to create purposeful transport paths.

5.1 TE metrics and constraints

TE metrics and constraints is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Model latency, bandwidth, affinity, disjointness, loss, maintenance, and administrative objectives. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for te metrics and constraints.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating te metrics and constraints as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for te metrics and constraints.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

5.2 PCE and controller architecture

PCE and controller architecture is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Separate path computation, policy authority, device programming, delegation, state, and failure handling. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for pce and controller architecture.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Implementation sketch

```
sr_policy:
  color: 100
  endpoint: 2001:db8:ffff::9
  binding_sid: 16050
  candidate_paths:
    - preference: 200
      source: pce
      constraints:
        max_latency_ms: 20
        exclude_affinity: maintenance
    - preference: 100
      source: dynamic
validation:
  - active_candidate_path
  - steering_matches_color
  - segment_list_within_hardware_limit
  - protected_failure_path
  - mtu_and_security_boundary
```

Failure patterns

- Treating pce and controller architecture as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for pce and controller architecture.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

5.3 BGP SR Policy distribution

BGP SR Policy distribution is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Advertise policy NLRI, tunnel encapsulation attributes, candidate paths, preferences, and segment types. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing,

tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for bgp sr policy distribution.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating bgp sr policy distribution as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for bgp sr policy distribution.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

5.4 Closed-loop optimization

Closed-loop optimization is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Use telemetry to propose reoptimization while protecting stability, capacity, policy, and human authority. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

Document the authoritative design intent, ownership, and scope for closed-loop optimization.

Predict protocol and forwarding state before implementation or troubleshooting.

Validate control-plane state, installed forwarding state, and real traffic independently.

Test normal, boundary, degraded, failed, recovery, and rollback conditions.

Retain packet, configuration, counter, log, topology, timing, and service evidence.

Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

Treating closed-loop optimization as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for closed-loop optimization.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

Chapter 05 laboratory and review

Practical laboratory
Compute and deploy latency- and affinity-constrained policies through a controller or offline solver. Inject stale topology and prove the safety checks.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate AI-assisted path optimization, digital twins, cross-layer optical coordination, and verifiable autonomous TE.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Fast reroute and resilience

Protect traffic locally while preserving loop freedom and policy intent.

Chapter operating position
Protect traffic locally while preserving loop freedom and policy intent.

6.1 Loop-free alternates and remote LFAs

Loop-free alternates and remote LFAs is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Understand local repair, topology conditions, repair nodes, tunnels, and coverage limits. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for loop-free alternates and remote lfas.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating loop-free alternates and remote lfas as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for loop-free alternates and remote lfas.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

6.2 TI-LFA with segment routing

TI-LFA with segment routing is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Calculate post-convergence paths, encode repairs, protect links, nodes, and shared-risk groups. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for ti-lfa with segment routing.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Implementation sketch

```
sr_policy:
  color: 100
  endpoint: 2001:db8:ffff::9
  binding_sid: 16050
  candidate_paths:
    - preference: 200
      source: pce
      constraints:
        max_latency_ms: 20
        exclude_affinity: maintenance
    - preference: 100
      source: dynamic
validation:
  - active_candidate_path
  - steering_matches_color
  - segment_list_within_hardware_limit
  - protected_failure_path
  - mtu_and_security_boundary
```

Failure patterns

- Treating ti-lfa with segment routing as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for ti-lfa with segment routing.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

6.3 Microloops and convergence

Microloops and convergence is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Coordinate local repair, IGP convergence, ordered updates, policy changes, and traffic stability. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing,

tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for microloops and convergence.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating microloops and convergence as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for microloops and convergence.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

6.4 Failure-domain testing

Failure-domain testing is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Test node, link, adjacency, controller, policy, and multi-failure behavior with packet-loss measurement. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

Document the authoritative design intent, ownership, and scope for failure-domain testing.

Predict protocol and forwarding state before implementation or troubleshooting.

Validate control-plane state, installed forwarding state, and real traffic independently.

Test normal, boundary, degraded, failed, recovery, and rollback conditions.

Retain packet, configuration, counter, log, topology, timing, and service evidence.

Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

Treating failure-domain testing as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for failure-domain testing.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

Chapter 06 laboratory and review

Practical laboratory
Deploy TI-LFA or simulate its repair list, fail protected links and nodes, and compare local repair time with final convergence and application impact.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore predictive failure, photonic-layer signals, deterministic repair, and resilient network slicing.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Security, observability, and troubleshooting

Protect source-routing authority and prove packets follow intended instructions.

Chapter operating position
Protect source-routing authority and prove packets follow intended instructions.

7.1 Domain boundaries and filtering

Domain boundaries and filtering is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Prevent unauthorized SRH, labels, or policies from entering, leaving, or bypassing enforcement boundaries. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for domain boundaries and filtering.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating domain boundaries and filtering as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for domain boundaries and filtering.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

7.2 SID and policy governance

SID and policy governance is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Control allocation, ownership, reuse, advertisements, stale state, and policy authorization. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for sid and policy governance.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Implementation sketch

```
sr_policy:
  color: 100
  endpoint: 2001:db8:ffff::9
  binding_sid: 16050
  candidate_paths:
    - preference: 200
      source: pce
      constraints:
        max_latency_ms: 20
        exclude_affinity: maintenance
    - preference: 100
      source: dynamic
validation:
  - active_candidate_path
  - steering_matches_color
  - segment_list_within_hardware_limit
  - protected_failure_path
  - mtu_and_security_boundary
```

Failure patterns

- Treating sid and policy governance as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for sid and policy governance.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

7.3 Telemetry and path verification

Telemetry and path verification is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Collect policy state, candidate paths, segment lists, counters, latency, drops, SRH, and actual path evidence. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing,

tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for telemetry and path verification.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating telemetry and path verification as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for telemetry and path verification.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

7.4 Fault isolation

Fault isolation is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Trace topology, SID distribution, policy selection, steering, stack or header processing, MTU, and service behavior. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

Document the authoritative design intent, ownership, and scope for fault isolation.

Predict protocol and forwarding state before implementation or troubleshooting.

Validate control-plane state, installed forwarding state, and real traffic independently.

Test normal, boundary, degraded, failed, recovery, and rollback conditions.

Retain packet, configuration, counter, log, topology, timing, and service evidence.

Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

Treating fault isolation as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for fault isolation.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

Chapter 07 laboratory and review

Practical laboratory
Troubleshoot a missing SID, inactive candidate path, incorrect color steering, MTU failure, and rejected SRH. Preserve control and data-plane proof.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate cryptographic path attestation, in-situ OAM, zero-trust SR domains, and post-quantum controller channels.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Interworking, migration, and future transport

Adopt segment routing without destabilizing existing services or operations.

Chapter operating position
Adopt segment routing without destabilizing existing services or operations.

8.1 LDP and SR-MPLS interworking

LDP and SR-MPLS interworking is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Use mapping, preference, coexistence, label allocation, targeted migration, and failure validation. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for ldp and sr-mpls interworking.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating ldp and sr-mpls interworking as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for ldp and sr-mpls interworking.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

8.2 RSVP-TE and SR Policy coexistence

RSVP-TE and SR Policy coexistence is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Preserve explicit services, protection, bandwidth expectations, OAM, and operational tooling. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for rsvp-te and sr policy coexistence.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Implementation sketch

```
sr_policy:
  color: 100
  endpoint: 2001:db8:ffff::9
  binding_sid: 16050
  candidate_paths:
    - preference: 200
      source: pce
      constraints:
        max_latency_ms: 20
        exclude_affinity: maintenance
    - preference: 100
      source: dynamic
validation:
  - active_candidate_path
  - steering_matches_color
  - segment_list_within_hardware_limit
  - protected_failure_path
  - mtu_and_security_boundary
```

Failure patterns

- Treating rsvp-te and sr policy coexistence as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for rsvp-te and sr policy coexistence.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

8.3 VPN and service continuity

VPN and service continuity is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Maintain L3VPN, EVPN, pseudowire, cloud, and customer behavior across transport changes. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing,

tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

- Document the authoritative design intent, ownership, and scope for vpn and service continuity.
- Predict protocol and forwarding state before implementation or troubleshooting.
- Validate control-plane state, installed forwarding state, and real traffic independently.
- Test normal, boundary, degraded, failed, recovery, and rollback conditions.
- Retain packet, configuration, counter, log, topology, timing, and service evidence.
- Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

- Treating vpn and service continuity as a configuration recipe without understanding protocol state.
- Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.
- Assuming an established adjacency or installed route proves end-to-end forwarding.
- Changing multiple variables before preserving the original failure evidence.
- Using vendor-default timers, trust, or policy without documenting the resulting behavior.
- Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for vpn and service continuity.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

8.4 SRv6 and future-network evolution

SRv6 and future-network evolution is an operational mechanism with observable inputs, state transitions, forwarding consequences, and failure signatures. Evaluate service programming, mobile user planes, network slicing, edge compute, and programmable fabrics. The engineer should be able to predict the expected state before opening a command line, then compare that prediction with device, endpoint, packet, and service evidence.

In this guide, the subject is traced through the topology, segment, policy, steering, and data-plane chain. Configuration alone is not accepted as proof: intended state must be reconciled with protocol state, installed forwarding state, traffic counters, packet behavior, and the application or receiver outcome. This prevents a healthy control plane from concealing a broken data plane, and prevents a working test flow from concealing an unsafe policy.

A production implementation begins by defining authority, scope, identifiers, timers, capacity, security boundaries, telemetry, and rollback. Establish a small known-good baseline; exercise normal operation; inject one failure at a time; preserve timestamps and raw evidence; and only then introduce optimization, automation, scale, or autonomous control. Every abstraction must retain a path back to the underlying protocol state.

Troubleshooting should identify the first point where observed behavior diverges from the expected transaction. Inspect both directions, distinguish absence from rejection, and account for caching, stale state, recursive dependencies, load sharing, tunneling, policy, and hardware offload. A final conclusion should state what failed, why it failed, what evidence proves it, what restored service, and what prevents recurrence.

Engineering practices

Document the authoritative design intent, ownership, and scope for srv6 and future-network evolution.

Predict protocol and forwarding state before implementation or troubleshooting.

Validate control-plane state, installed forwarding state, and real traffic independently.

Test normal, boundary, degraded, failed, recovery, and rollback conditions.

Retain packet, configuration, counter, log, topology, timing, and service evidence.

Automate only after the manual behavior, invariants, and failure handling are understood.

Failure patterns

Treating srv6 and future-network evolution as a configuration recipe without understanding protocol state.

Checking only the local device while ignoring the neighbor, endpoint, reverse path, or remote policy.

Assuming an established adjacency or installed route proves end-to-end forwarding.

Changing multiple variables before preserving the original failure evidence.

Using vendor-default timers, trust, or policy without documenting the resulting behavior.

Closing the incident after one successful probe without negative tests or recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Topology, addressing, policy, configuration, protocol state, counters, and source authority for srv6 and future-network evolution.
Capture	Relevant packets or telemetry at enough observation points to distinguish sender, path, policy, and receiver behavior.
Test	Nominal traffic, invalid input, dependency loss, partial failure, convergence, restoration, and rollback.
Measure	Reachability, loss, latency, convergence, resource use, state scale, error rates, and application outcome.
Reconcile	Intended state against observed endpoint, control-plane, forwarding, security, and service evidence.

Chapter 08 laboratory and review

Practical laboratory
Design a staged LDP-to-SR migration and a separate SR-MPLS-to-SRv6 evaluation. Define gates, telemetry, rollback, service proof, and training needs.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore compressed SRv6, BGP CAR, semantic networking, quantum-safe control, photonic fabrics, and AI-native 6G transport.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. IETF / RFC Editor - RFC 8402 - Segment Routing Architecture
Role: Segment-routing architecture.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc8402

02. IETF / RFC Editor - RFC 8660 - Segment Routing with the MPLS Data Plane
Role: SR-MPLS data-plane behavior.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc8660

03. IETF / RFC Editor - RFC 8754 - IPv6 Segment Routing Header
Role: IPv6 SRH format and behavior.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc8754

04. IETF / RFC Editor - RFC 8986 - SRv6 Network Programming
Role: SRv6 behaviors and network-programming model.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc8986

05. IETF / RFC Editor - RFC 9256 - Segment Routing Policy Architecture
Role: SR Policy architecture and candidate paths.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9256

06. IETF / RFC Editor - RFC 9602 - SRv6 SIDs in the IPv6 Addressing Architecture
Role: Current SRv6 SID addressing guidance.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9602

07. IETF / RFC Editor - RFC 9830 - Advertising Segment Routing Policies in BGP
Role: BGP distribution of SR Policies.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9830

08. IETF / RFC Editor - RFC 9855 - Topology Independent Fast Reroute Using Segment Routing
Role: TI-LFA fast-reroute architecture.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9855

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-NET - Network Engineering & Architecture
Secondary domain	MYTHOS-SEC; MYTHOS-CLD; MYTHOS-EMT
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	segment routing, SR-MPLS, SRv6, SR Policy, TI-LFA, PCE, traffic engineering, SIDs, network programming, future networks
Canonical route	/library/field-guides/mythos-fg-net-0007/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.