

REFERENCE GUIDE



Network Troubleshooting

ISP vs Enterprises: Diagnostic Techniques & tools

Gitesh Dhungana

<https://www.linkedin.com/in/gitesh-d-aa95a8227/>

ISP vs. Enterprise Network Issue Identification

A Complete Reference Guide for Network Engineers

Master the art of network troubleshooting with systematic diagnostic techniques, practical tools, and real-world scenarios to quickly identify whether network issues originate from your enterprise infrastructure or your ISP.

Table of Contents

- **Chapter 1:** Foundations - Understanding the Tools
 - **Chapter 2:** Troubleshooting Logic - Where is the Problem?
 - **Chapter 3:** Categorized Scenarios and Root Cause Mapping
 - **Chapter 4:** Advanced Techniques and Best Practices
 - **Chapter 5:** Quick Reference Matrix
 - **Chapter 6:** Practical Examples
 - **Chapter 7:** Best Practices for Ongoing Monitoring
 - **Chapter 8:** Common Network Design Pitfalls
 - **Appendix:** Quick Diagnostic Scripts
-

⚠ Important Note

This guide provides general troubleshooting methodologies. Always follow your organization's specific procedures and escalation policies. Network configurations may vary significantly between environments.

CHAPTER 1: Foundations - Understanding the Tools

Network troubleshooting is both an art and a science. Success depends on understanding your tools, following systematic approaches, and interpreting results correctly. This chapter introduces the fundamental tools every network engineer must master.

1.1 Key Tools and Their Roles

PING The most basic connectivity test. Sends ICMP Echo Request packets to verify reachability, measure latency, and detect packet loss. *Available on: Windows/Linux/macOS*

traceroute / tracert Maps the route packets take through the network by incrementing TTL values. Identifies each hop and measures delays. *Available on: Linux/Windows*

pathping Windows-specific tool that combines ping and traceroute functionality, providing comprehensive per-hop loss and latency statistics. *Available on: Windows*

mtr (My Traceroute) Real-time, continuous traceroute with detailed loss and latency statistics per hop. Excellent for identifying intermittent issues. *Available on: Linux/macOS*

1.2 Understanding Tool Output

Parameter	Meaning	Interpretation
Reply from...	Target responded successfully	Device is reachable and responding
Request timed out	No reply received within timeout	Device unreachable, filtered, or packet dropped
TTL (Time To Live)	Number of hops before packet expires	Indicates routing path length
Time (ms)	Round-trip latency in milliseconds	Network performance indicator
Loss (%)	Percentage of packets lost	Network reliability indicator
Hop (traceroute)	Each device traversed along the path	Network topology mapping
Host unreachable	No route to destination	Routing or firewall configuration issue
Unknown host	DNS resolution failed	DNS configuration or connectivity issue

⚠ Tool Limitations

Remember that some network devices and firewalls may block or rate-limit ICMP traffic, potentially giving false negatives. Always correlate results with other diagnostic methods.

CHAPTER 2: Troubleshooting Logic - Where is the Problem?

Effective network troubleshooting follows a systematic, step-by-step approach. This methodology helps you quickly isolate issues and determine responsibility - whether the problem lies within your enterprise network or with your ISP.

2.1 The Stepwise Diagnostic Flow

Step 1: Check Local Network

bash

```
ping <default-gateway>
```

- **Success:** Local network segment is functioning properly.
- **Failure:** Problem is within the local segment, switch, or client configuration.

Step 2: Check Internal Enterprise Reachability

bash

```
ping <internal-server>
traceroute <internal-server>
```

- **Success:** Core enterprise network infrastructure is operational.
- **Failure:** Issue exists within enterprise network (LAN, VLAN, firewall, or routing).

Step 3: Check ISP Edge Router/Firewall

bash

```
ping <ISP-edge-router-IP>
```

- **Success:** Enterprise edge devices are up; proceed to test external connectivity.
- **Failure:** Issue at enterprise WAN edge (firewall/router, cabling, or ISP handoff).

Step 4: Test Internet Reachability

bash

```
ping 8.8.8.8
ping 1.1.1.1
```

- **Success:** Internet connectivity is available; issue may be remote or application-specific.
- **Failure:** Possible ISP circuit issue or upstream provider problem.

Step 5: Test DNS Resolution

bash

```
ping google.com
nslookup google.com
```

- **Success:** DNS resolution is working correctly.
- **Failure:** DNS misconfiguration, ISP DNS servers down, or firewall blocking DNS traffic.

Step 6: Trace the Path to Internet

bash

```
traceroute 8.8.8.8 (Linux)
tracert 8.8.8.8 (Windows)
mtr 8.8.8.8 (Linux/macOS)
```

- **Loss/latency at early hops:** Enterprise network problem.

- **Loss/latency at ISP/after handoff:** ISP or upstream provider issue.

Step 7: Use Advanced Tools for Deeper Analysis

bash

pathping [8.8.8.8](#) (Windows)

mtr [8.8.8.8](#) (Linux/macOS)

Purpose: Pinpoint the exact hop where loss/latency spikes occur for precise problem isolation.

⚠ Critical Considerations

Always test multiple destinations and run tests multiple times to account for load balancing, routing changes, and intermittent issues. Document all results with timestamps for trending and escalation purposes.

CHAPTER 3: Categorized Scenarios and Root Cause Mapping

Understanding common failure patterns helps you quickly categorize issues and determine responsibility. This chapter provides detailed scenarios you'll encounter in the field.

3.1 Enterprise Network Issues

A. Local Device/Network Problems

Symptoms:

- Cannot ping default gateway or internal servers
- Traceroute fails at the first hop
- Physical connectivity indicators show problems

Likely Causes:

- Physical cabling issues (damaged, disconnected, or wrong cables)
- Switch port configuration problems
- VLAN membership or tagging issues
- Local firewall blocking traffic
- Device network configuration errors
- NIC hardware failure

B. Internal Routing/Firewall Issues

Symptoms:

- Can ping gateway but not internal servers
- Traceroute fails after the first few hops
- Selective connectivity to some internal resources

Likely Causes:

- Routing table misconfiguration
- Access Control Lists (ACLs) blocking traffic
- Firewall rules preventing communication
- VLAN segmentation issues
- Core network device overload or failure
- Spanning Tree Protocol (STP) blocking ports

C. DNS/Internal Service Issues

Symptoms:

- Can ping by IP address but not by hostname
- Web browsing fails but IP-based services work
- Intermittent name resolution failures

Likely Causes:

- DNS server misconfiguration or failure
- DNS server unreachable due to network issues
- Split-brain DNS configuration problems
- DNS cache poisoning or corruption
- Firewall blocking DNS traffic (port 53)
- Incorrect DNS forwarding configuration

D. Enterprise Edge Issues

Symptoms:

- Can ping internal resources but not ISP edge
- Traceroute/pathping/mtr fails at edge router/firewall
- VPN connections failing

Likely Causes:

- WAN link physically down or degraded
- Edge firewall misconfiguration

- ISP handoff cable or interface issues
- Router hardware failure
- BGP or routing protocol issues
- Circuit provisioning problems

3.2 ISP/Service Provider Issues

A. ISP Link Down or Unstable

Symptoms:

- Can ping edge router but not public IP addresses
- Traceroute fails at or immediately after ISP handoff
- All external services unavailable

Likely Causes:

- ISP circuit completely down
- Fiber or copper cable cuts
- ISP scheduled or emergency maintenance
- Distributed Denial of Service (DDoS) attacks
- ISP equipment failure
- Upstream provider outage

B. High Latency/Loss at ISP or Beyond

Symptoms:

- Traceroute/mtr/pathping shows spikes/loss at ISP or subsequent hops
- Intermittent connectivity issues
- Slow application performance

Likely Causes:

- ISP network congestion during peak hours
- Peering issues between ISPs
- Upstream provider network problems
- International link saturation
- ISP traffic shaping or throttling
- Routing table convergence issues

C. DNS Issues at ISP

Symptoms:

- Can ping public IP addresses but not public hostnames
- DNS resolution timeouts or failures
- Intermittent web browsing issues

Likely Causes:

- ISP DNS servers down or overloaded
- DNS server misconfiguration at ISP
- DNS servers under DDoS attack
- Upstream DNS resolution issues
- DNS cache poisoning at ISP level
- ISP DNS filtering or blocking

D. Regional/Upstream Internet Issues

Symptoms:

- Traceroute/mtr shows loss/latency after ISP at international or cloud provider hops
- Selective destination unreachability
- Geographic-specific connectivity issues

Likely Causes:

- Internet backbone infrastructure problems
- Large-scale DDoS attacks affecting major providers
- Peering disputes between major ISPs
- Cloud service provider outages
- Submarine cable cuts (for international traffic)
- BGP routing attacks or misconfigurations

CHAPTER 4: Advanced Techniques and Best Practices

Beyond basic connectivity testing, advanced diagnostic techniques provide deeper insights into network behavior and help identify complex issues that simple ping tests might miss.

4.1 Using mtr for Real-Time Diagnosis

bash

```
mtr -c 100 8.8.8.8
```

MTR (My Traceroute) combines the functionality of traceroute and ping, providing continuous monitoring with statistical analysis.

Key mtr Parameters:

- `-c [count]`: Number of pings to send (default: unlimited)
- `-r`: Report mode (non-interactive)
- `-s [size]`: Packet size in bytes
- `-i [interval]`: Seconds between pings
- `-n`: No DNS lookups (faster execution)

Interpreting mtr Results:

- Loss/latency at first hops: Enterprise network issue
- Loss/latency at ISP hops: Service provider problem
- Loss/latency at final hops: Remote site or cloud provider issue
- Consistent loss across all hops: Often indicates rate limiting, not actual packet loss

4.2 Using pathping for Windows Environments

cmd

```
pathping -n -q 10 -p 1000 8.8.8.8
```

Pathping provides comprehensive statistics by combining ping and traceroute functionality, showing cumulative loss and latency at each hop.

Key pathping Parameters:

- `-n`: No DNS lookups (faster execution)
- `-q [queries]`: Number of queries per hop
- `-p [period]`: Milliseconds between pings
- `-w [timeout]`: Timeout in milliseconds
- `-i [address]`: Use specific source address

4.3 Advanced Ping Techniques

Testing with Different Packet Sizes

bash

```
ping -s 1472 8.8.8.8 (Linux - test MTU)  
ping -l 1472 8.8.8.8 (Windows - test MTU)
```

Continuous Monitoring

bash

```
ping -i 0.5 8.8.8.8 (Linux - ping every 0.5 seconds)
ping -t 8.8.8.8 (Windows - continuous ping)
```

Testing from Specific Interface

bash

```
ping -I eth0 8.8.8.8 (Linux)
ping -S 192.168.1.100 8.8.8.8 (Windows)
```

4.4 Documentation and Escalation

Essential Information to Record:

- Timestamp: When the issue was first observed
- Affected Systems: Which hosts, subnets, or services are impacted
- Command Output: Complete results from diagnostic commands
- Hop-by-hop Analysis: Where exactly the issue manifests
- Baseline Comparison: How current performance compares to normal
- Impact Assessment: Business impact and user complaints

ISP Escalation Requirements

When escalating to your ISP, provide:

- Complete traceroute/mtr/pathping output
- Source and destination IP addresses
- Precise timestamps of testing
- Circuit ID and account information
- Description of business impact
- Previous ticket numbers for related issues

⚠ Escalation Best Practices

Always perform your own due diligence before escalating. ISPs typically require proof that the issue is not within your network. Document your internal testing thoroughly and be prepared to provide additional information upon request.

CHAPTER 5: Quick Reference Matrix

This comprehensive matrix provides quick lookup for common symptoms, appropriate diagnostic commands, likely root causes, and responsibility assignment.

Symptom/Observation	Diagnostic Command(s)	Likely Root Cause	Responsibility
Cannot ping gateway	ping <gateway>	Local/Enterprise network issue	Enterprise IT
Can ping gateway, not internal server	ping <internal-server> traceroute <internal-server>	Internal routing/firewall issue	Enterprise IT
Can ping internal, not ISP edge	ping <ISP-edge>	Edge device/link failure	Enterprise/ISP
Can ping edge, not public IP	ping 8.8.8.8 traceroute 8.8.8.8	ISP/Upstream provider issue	ISP
Traceroute fails at first/second hop	traceroute <any>	Enterprise network problem	Enterprise IT
Traceroute fails at ISP hop	traceroute <any> mtr <any>	ISP circuit issue	ISP
Traceroute/mtr loss at remote hops	traceroute <any> mtr <any>	Upstream/Internet backbone issue	ISP/Cloud Provider
Can ping by IP, not by name	ping <hostname> nslookup <hostname>	DNS resolution issue	Enterprise/ISP
High loss/latency at all hops	mtr <any> pathping <any>	Local device/congestion	Enterprise IT
High loss/latency at ISP hops	mtr <any> pathping <any>	ISP congestion/failure	ISP
Intermittent connectivity	ping -t <destination> mtr -c 100 <destination>	Varies by hop where issue occurs	Varies
Slow application performance	ping <app-server> traceroute <app-server>	Network latency or application issue	Varies

5.1 Responsibility Assignment Guidelines

Enterprise IT Responsibility:

- Issues occurring within the first 1-3 hops of traceroute
- Problems isolated to internal network segments
- Local DNS resolution failures
- Edge device configuration issues
- Internal firewall or routing problems

ISP Responsibility:

- Issues occurring at or immediately after ISP handoff
- Complete loss of internet connectivity
- ISP DNS server failures
- High latency/loss at ISP infrastructure hops
- BGP routing issues affecting multiple destinations

Shared Responsibility:

- Issues at the exact handoff point between enterprise and ISP
- Physical layer problems (cables, connectors)
- Configuration mismatches between enterprise and ISP equipment
- Capacity planning and bandwidth utilization issues

CHAPTER 6: Practical Examples

Real-world scenarios demonstrate how to apply the diagnostic methodology. These examples show complete troubleshooting sequences with interpretation and conclusions.

Example 1: Diagnosing a Local Network Outage

bash

```
$ ping 192.168.1.1
PING 192.168.1.1: 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2

$ ping 8.8.8.8
PING 8.8.8.8: 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2

$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
1 ***
2 ***
3 ***
```

Analysis: The inability to reach the default gateway (192.168.1.1) combined with complete traceroute failure indicates a local network issue.

Conclusion: Local or enterprise network issue - check physical connections, switch configuration, and local network settings.

Example 2: ISP Circuit Failure

bash

```
$ ping 192.168.1.1  
PING 192.168.1.1: 56 data bytes  
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=1.2 ms  
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.1 ms
```

```
$ ping 203.0.113.1 # ISP edge router  
PING 203.0.113.1: 56 data bytes  
64 bytes from 203.0.113.1: icmp_seq=0 ttl=64 time=2.3 ms  
64 bytes from 203.0.113.1: icmp_seq=1 ttl=64 time=2.1 ms
```

```
$ ping 8.8.8.8  
PING 8.8.8.8: 56 data bytes  
Request timeout for icmp_seq 0  
Request timeout for icmp_seq 1
```

```
$ traceroute 8.8.8.8  
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets  
1 192.168.1.1 (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms  
2 203.0.113.1 (203.0.113.1) 2.234 ms 2.123 ms 2.345 ms  
3 ***  
4 ***
```

Analysis: Local network and ISP edge are reachable, but internet connectivity fails. Traceroute stops responding after the ISP edge router.

Conclusion: ISP circuit is down - escalate to ISP with this diagnostic data.

Example 3: Upstream Internet Congestion

```
bash
```

```
$ ping 8.8.8.8
PING 8.8.8.8: 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=116 time=245.2 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=289.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=301.5 ms
```

```
$ mtr -c 20 8.8.8.8
```

```
My traceroute [v0.92]
```

	HOST:	Loss%	Snt	Last	Avg	Best	Wrst	StDev
1. --	192.168.1.1	0.0%	20	1.2	1.3	1.0	2.1	0.3
2. --	203.0.113.1	0.0%	20	2.3	2.4	2.1	3.2	0.3
3. --	198.51.100.1	0.0%	20	15.2	16.1	14.8	22.3	2.1
4. --	203.0.113.254	5.0%	20	45.6	52.3	44.1	78.9	10.2
5. --	8.8.8.8	0.0%	20	245.2	267.8	234.5	312.1	23.4

Analysis: High latency appears starting at hop 4, with some packet loss. The final destination is reachable but with very high latency.

Conclusion: Upstream internet congestion - contact ISP to report performance issues and request investigation.

Example 4: DNS Resolution Failure

```
bash

$ ping 8.8.8.8
PING 8.8.8.8: 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=116 time=23.2 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=24.1 ms
```

```
$ ping google.com
ping: cannot resolve google.com: Unknown host
```

```
$ nslookup google.com
Server: 192.168.1.1
Address: 192.168.1.1#53
```

** server can't find google.com: NXDOMAIN

```
$ nslookup google.com 8.8.8.8
Server: 8.8.8.8
Address: 8.8.8.8#53
```

Non-authoritative answer:

```
Name: google.com
Address: 142.250.191.14
```

Analysis: Internet connectivity works by IP, but DNS resolution fails through the local DNS server (192.168.1.1) while working through Google's DNS (8.8.8.8).

Conclusion: Local DNS server issue - check DNS server configuration, forwarding settings, or switch to alternative DNS servers.

⚠ Example Interpretation Tips

Remember that these examples show typical scenarios. Real-world troubleshooting may require multiple test iterations, testing from different source locations, and correlating with other monitoring data. Always document your findings and maintain a troubleshooting log.

CHAPTER 7: Best Practices for Ongoing Monitoring

Proactive monitoring and baseline establishment are crucial for effective troubleshooting. This chapter outlines best practices for maintaining network health and preparing for rapid issue resolution.

7.1 Establishing Network Baselines

Baseline Metrics to Track:

- Latency: Round-trip times to key destinations during different times of day
- Packet Loss: Loss percentages under normal operating conditions
- Bandwidth Utilization: Peak and average usage patterns
- Route Stability: Typical routing paths and hop counts
- DNS Response Times: Resolution times for common queries

Automated Baseline Collection:

bash

```
# Linux cron job example - run every 15 minutes
*/15 * * * * /usr/bin/mtr -c 10 -r 8.8.8.8 >> /var/log/network-baseline.log
```

```
# Windows Task Scheduler equivalent
schtasks /create /tn "Network Baseline" /tr "pathping -n -q 10 8.8.8.8" /sc minute /mo 15
```

7.2 Monitoring Critical Network Points

Edge Device Monitoring

Key Monitoring Points:

- Default gateway availability and response time
- ISP edge router reachability
- Primary and backup circuit status
- Interface utilization and error rates
- BGP peer status and route advertisements

Internet Connectivity Monitoring

Recommended Test Destinations:

- 8.8.8 and 1.1.1.1 (Public DNS servers)
- Major cloud providers (AWS, Azure, Google Cloud)
- Content delivery networks (CDNs)
- Business-critical external services
- Geographic diversity of test points

7.3 Automated Alerting and Escalation

Alert Thresholds:

- Latency: Alert when >150% of baseline for >5 minutes

- Packet Loss: Alert on >1% loss for >2 minutes
- Availability: Alert on >30 seconds of complete unreachability
- Route Changes: Alert on unexpected routing path changes
- DNS: Alert on >5 second resolution times

7.4 Documentation and Trending

Essential Documentation:

- Network Topology: Current network diagrams and IP assignments
- Baseline Performance: Historical performance data and trends
- Escalation Procedures: Contact information and escalation paths
- Common Issues: Known problems and their solutions
- Change Log: Recent network changes and their impact

7.5 Proactive Maintenance

Regular Health Checks

Monthly Activities:

- Review baseline performance trends
- Test backup circuits and failover procedures
- Verify monitoring system accuracy
- Update network documentation
- Review ISP SLA compliance

CHAPTER 8: Common Network Design Pitfalls

Poor network design can significantly complicate troubleshooting efforts. This chapter identifies common pitfalls and their impact on diagnostic procedures.

8.1 Documentation and Baseline Pitfalls

Absence of Network Baseline Documentation

Impact on Troubleshooting:

- Inability to distinguish between normal and abnormal performance
- Difficulty identifying gradual performance degradation
- Challenges in setting appropriate alerting thresholds
- Ineffective capacity planning

Mitigation Strategies:

- Implement automated baseline collection using tools like mtr and pathping
- Establish performance baselines during different time periods
- Document normal routing paths and hop counts
- Create performance dashboards for trend analysis

8.2 Redundancy and Failover Pitfalls

Lack of Redundancy at Edge/ISP Handoff

Impact on Troubleshooting:

- Single point of failure makes root cause analysis difficult
- Cannot distinguish between enterprise and ISP issues during outages
- Extended downtime during troubleshooting procedures
- Pressure to restore service quickly may compromise thorough diagnosis

Mitigation Strategies:

- Implement dual ISP connections with different providers
- Deploy redundant edge devices with proper failover mechanisms
- Use diverse physical paths for primary and backup circuits
- Implement intelligent routing protocols (BGP) for automatic failover

8.3 Segmentation and Security Pitfalls

Poor Segmentation Between Enterprise and ISP Networks

Impact on Troubleshooting:

- Difficulty isolating issues at the network boundary
- Security policies may interfere with diagnostic tools
- Lack of visibility into ISP handoff performance
- Complications in determining responsibility for issues

Mitigation Strategies:

- Implement proper network segmentation with clear boundaries
- Deploy monitoring points at critical network junctions
- Establish clear diagnostic procedures that work within security constraints
- Maintain separate management networks for troubleshooting access

Overly Restrictive Firewall Policies

Impact on Troubleshooting:

- Diagnostic tools may be blocked or filtered
- ICMP traffic restrictions limit ping/traceroute effectiveness
- Difficulty accessing network devices during outages
- Increased time to resolution due to security approval processes

Mitigation Strategies:

- Create specific firewall rules for diagnostic traffic
- Implement out-of-band management networks
- Establish emergency access procedures for troubleshooting
- Regular review and testing of diagnostic tool accessibility

8.4 Monitoring and Alerting Pitfalls

Inadequate Monitoring Coverage

Impact on Troubleshooting:

- Issues may go undetected until users report problems
- Lack of historical data for trend analysis
- Difficulty correlating events across network segments
- Reactive rather than proactive approach to network issues

Mitigation Strategies:

- Implement comprehensive monitoring across all network segments
- Use multiple monitoring tools for redundancy and validation
- Establish clear escalation procedures based on monitoring alerts
- Regular review and tuning of monitoring thresholds

Alert Fatigue and Poor Thresholds

Impact on Troubleshooting:

- Important alerts may be ignored due to excessive noise
- Inappropriate thresholds leading to false positives or missed issues
- Delayed response to genuine network problems
- Reduced effectiveness of automated escalation procedures

Mitigation Strategies:

- Regularly review and adjust alert thresholds based on baseline data
- Implement intelligent alerting with correlation and suppression
- Use escalation procedures that account for alert severity and duration
- Provide training on proper alert response procedures

8.5 Change Management Pitfalls

Poor Change Control and Documentation

Impact on Troubleshooting:

- Difficulty correlating network changes with performance issues
- Lack of rollback procedures during troubleshooting
- Inconsistent network configurations across similar devices
- Increased time to resolution due to unknown network state

Mitigation Strategies:

- Implement strict change control procedures with proper documentation
 - Maintain configuration backups and version control
 - Establish clear rollback procedures for network changes
 - Regular auditing of network configurations for consistency
-

APPENDIX: Quick Diagnostic Scripts

These scripts provide automated diagnostic capabilities for common troubleshooting scenarios. Customize them according to your specific network environment and requirements.

Linux Diagnostic Scripts

Basic Connectivity Test Script

```

bash

#!/bin/bash
# basic_connectivity_test.sh

echo "==== Basic Network Connectivity Test ===="
echo "Timestamp: $(date)"
echo

# Test local gateway
GATEWAY=$(ip route | grep default | awk '{print $3}' | head -1)
echo "Testing local gateway: $GATEWAY"
ping -c 3 $GATEWAY
echo

# Test DNS resolution
echo "Testing DNS resolution:"
nslookup google.com
echo

# Test internet connectivity
echo "Testing internet connectivity:"
ping -c 3 8.8.8.8
ping -c 3 1.1.1.1
echo

# Trace route to internet
echo "Tracing route to 8.8.8.8:"
traceroute 8.8.8.8
echo

echo "==== Test Complete ===="

```

Advanced MTR Monitoring Script

```
bash

#!/bin/bash
# mtr_monitoring.sh

TARGETS=("8.8.8.8" "1.1.1.1" "google.com" "cloudflare.com")
LOGFILE="/var/log/network-monitoring.log"

echo "==== MTR Network Monitoring - $(date) ====" >> $LOGFILE

for target in "${TARGETS[@]}"; do
    echo "Testing $target:" >> $LOGFILE
    mtr -c 10 -r $target >> $LOGFILE
    echo >> $LOGFILE
done

echo "==== End of Test - $(date) ====" >> $LOGFILE
echo >> $LOGFILE
```

Windows Diagnostic Scripts

Basic Connectivity Test Script (PowerShell)

```
powershell
```

```
# BasicConnectivityTest.ps1
```

```
Write-Host "==== Basic Network Connectivity Test ====" -ForegroundColor Green
```

```
Write-Host "Timestamp: $(Get-Date)" -ForegroundColor Yellow
```

```
Write-Host
```

```
# Test local gateway
```

```
$gateway = (Get-NetRoute -DestinationPrefix 0.0.0.0/0).NextHop
```

```
Write-Host "Testing local gateway: $gateway" -ForegroundColor Cyan
```

```
Test-NetConnection -ComputerName $gateway -InformationLevel Detailed
```

```
Write-Host
```

```
# Test DNS resolution
```

```
Write-Host "Testing DNS resolution:" -ForegroundColor Cyan
```

```
Resolve-DnsName google.com
```

```
Write-Host
```

```
# Test internet connectivity
```

```
Write-Host "Testing internet connectivity:" -ForegroundColor Cyan
```

```
Test-NetConnection -ComputerName 8.8.8.8 -InformationLevel Detailed
```

```
Test-NetConnection -ComputerName 1.1.1.1 -InformationLevel Detailed
```

```
Write-Host
```

```
# Trace route to internet
```

```
Write-Host "Tracing route to 8.8.8.8:" -ForegroundColor Cyan
```

```
Test-NetConnection -ComputerName 8.8.8.8 -TraceRoute
```

```
Write-Host
```

```
Write-Host "==== Test Complete ====" -ForegroundColor Green
```

Pathping Monitoring Script (Batch)

batch

@echo off

REM pathping_monitoring.bat

echo === Pathping Network Monitoring - %date% %time% === >> network-monitoring.log
echo.

echo Testing 8.8.8: >> network-monitoring.log
pathping -n -q 10 8.8.8 >> network-monitoring.log
echo. >> network-monitoring.log

echo Testing 1.1.1.1: >> network-monitoring.log
pathping -n -q 10 1.1.1.1 >> network-monitoring.log
echo. >> network-monitoring.log

echo Testing google.com: >> network-monitoring.log
pathping -n -q 10 google.com >> network-monitoring.log
echo. >> network-monitoring.log

echo === End of Test - %date% %time% === >> network-monitoring.log
echo. >> network-monitoring.log

Network Issue Classification Script

Automated Issue Classification (Python)

python

```
#!/usr/bin/env python3
# network_issue_classifier.py

import subprocess
import sys
import re
from datetime import datetime

def run_command(command):
    """Execute a command and return the output"""
    try:
        result = subprocess.run(command, shell=True, capture_output=True, text=True, timeout=30)
        return result.stdout, result.stderr, result.returncode
    except subprocess.TimeoutExpired:
        return "", "Command timed out", 1
    except Exception as e:
        return "", str(e), 1

def test_connectivity(host):
    """Test connectivity to a host"""
    command = f"ping -c 3 {host}" if sys.platform != "win32" else f"ping -n 3 {host}"
    stdout, stderr, returncode = run_command(command)

    if returncode == 0:
        return True, "Success"
    else:
        return False, stderr

def get_default_gateway():
    """Get the default gateway IP address"""
    if sys.platform == "win32":
        command = "ipconfig | findstr Gateway"
    else:
        command = "ip route | grep default | awk '{print $3}' | head -1"

    stdout, stderr, returncode = run_command(command)
    if returncode == 0 and stdout.strip():
        if sys.platform == "win32":
            # Extract IP from Windows output
            match = re.search(r'(\d+\.\d+\.\d+\.\d+)', stdout)
            return match.group(1) if match else None
        else:
            return stdout.strip()
    return None

def classify_network_issue():
```

```
"""Classify network issues based on diagnostic results"""
print("== Network Issue Classification ==")
print(f"Timestamp: {datetime.now()}")
print()

# Step 1: Test local gateway
gateway = get_default_gateway()
if not gateway:
    print("✗ Cannot determine default gateway")
    return "CONFIGURATION_ERROR"

print(f"Testing local gateway: {gateway}")
gateway_success, gateway_msg = test_connectivity(gateway)

if not gateway_success:
    print(f"✗ Local gateway unreachable: {gateway_msg}")
    return "LOCAL_NETWORK_ISSUE"

print("✓ Local gateway reachable")

# Step 2: Test DNS servers
print("\nTesting DNS servers...")
dns_servers = ["8.8.8.8", "1.1.1.1"]
dns_success = False

for dns in dns_servers:
    success, msg = test_connectivity(dns)
    if success:
        print(f"✓ DNS server {dns} reachable")
        dns_success = True
        break
    else:
        print(f"✗ DNS server {dns} unreachable: {msg}")

if not dns_success:
    print("✗ All DNS servers unreachable")
    return "ISP_CONNECTIVITY_ISSUE"

# Step 3: Test DNS resolution
print("\nTesting DNS resolution...")
if sys.platform == "win32":
    command = "nslookup google.com"
else:
    command = "host google.com"

stdout, stderr, returncode = run_command(command)
```

```

if returncode == 0:
    print("✅ DNS resolution working")
else:
    print(f"❌ DNS resolution failed: {stderr}")
    return "DNS_RESOLUTION_ISSUE"

# Step 4: Test web connectivity
print("\nTesting web connectivity...")
web_hosts = ["google.com", "cloudflare.com"]
web_success = False

for host in web_hosts:
    success, msg = test_connectivity(host)
    if success:
        print(f"✅ Web host {host} reachable")
        web_success = True
        break
    else:
        print(f"❌ Web host {host} unreachable: {msg}")

if web_success:
    print("\n✅ Network connectivity appears normal")
    return "NO_ISSUE_DETECTED"
else:
    print("\n❌ Web connectivity issues detected")
    return "UPSTREAM_CONNECTIVITY_ISSUE"

def main():
    """Main function"""
    issue_type = classify_network_issue()

    print(f"\n==== CLASSIFICATION RESULT: {issue_type} ====")

    # Provide recommendations based on classification
    recommendations = {
        "LOCAL_NETWORK_ISSUE": [
            "Check physical network connections",
            "Verify network adapter configuration",
            "Check switch port status",
            "Verify VLAN configuration"
        ],
        "ISP_CONNECTIVITY_ISSUE": [
            "Contact ISP support",
            "Check ISP service status",
            "Verify WAN connection status",
            "Check for scheduled maintenance"
        ],
    }

```

```

"DNS_RESOLUTION_ISSUE": [
    "Check DNS server configuration",
    "Try alternative DNS servers",
    "Clear DNS cache",
    "Check firewall DNS rules"
],
"UPSTREAM_CONNECTIVITY_ISSUE": [
    "Run traceroute to identify problem hops",
    "Check for regional internet issues",
    "Contact ISP about upstream connectivity",
    "Monitor for resolution"
],
"CONFIGURATION_ERROR": [
    "Verify network configuration",
    "Check routing table",
    "Verify IP address assignment",
    "Review recent configuration changes"
],
"NO_ISSUE_DETECTED": [
    "Issue may be intermittent",
    "Check application-specific connectivity",
    "Monitor for recurring issues",
    "Verify user reports"
]
}
}

if issue_type in recommendations:
    print("\nRecommended Actions:")
    for i, action in enumerate(recommendations[issue_type], 1):
        print(f"{i}. {action}")

if __name__ == "__main__":
    main()

```

Continuous Monitoring Script

Network Health Monitor (Python)

python

```
#!/usr/bin/env python3
# network_health_monitor.py

import subprocess
import time
import json
import logging
from datetime import datetime
from collections import defaultdict

class NetworkHealthMonitor:
    def __init__(self, config_file="network_config.json"):
        self.config = self.load_config(config_file)
        self.setup_logging()
        self.stats = defaultdict(list)

    def load_config(self, config_file):
        """Load monitoring configuration"""
        default_config = {
            "targets": [
                {"host": "8.8.8.8", "name": "Google DNS", "type": "dns"},
                {"host": "1.1.1.1", "name": "Cloudflare DNS", "type": "dns"},
                {"host": "google.com", "name": "Google", "type": "web"},
                {"host": "cloudflare.com", "name": "Cloudflare", "type": "web"}
            ],
            "intervals": {
                "ping_interval": 60,
                "traceroute_interval": 300,
                "report_interval": 900
            },
            "thresholds": {
                "latency_warning": 100,
                "latency_critical": 200,
                "loss_warning": 1,
                "loss_critical": 5
            }
        }

        try:
            with open(config_file, 'r') as f:
                config = json.load(f)
            return {**default_config, **config}
        except FileNotFoundError:
            return default_config

    def setup_logging(self):
```

```
"""Setup logging configuration"""
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('network_health.log'),
        logging.StreamHandler()
    ]
)
self.logger = logging.getLogger(__name__)

def ping_host(self, host, count=3):
    """Ping a host and return statistics"""
    import platform

    if platform.system().lower() == "windows":
        command = f"ping -n {count} {host}"
    else:
        command = f"ping -c {count} {host}"

    try:
        result = subprocess.run(command, shell=True, capture_output=True, text=True, timeout=30)

        if result.returncode == 0:
            # Parse ping output for statistics
            output = result.stdout
            if platform.system().lower() == "windows":
                import re
                times = re.findall(r'time[<=](\d+)ms', output)
                loss_match = re.search(r'(\d+)% loss', output)
            else:
                import re
                times = re.findall(r'time=(\d+\.\?\d*)', output)
                loss_match = re.search(r'(\d+)% packet loss', output)

            if times:
                avg_time = sum(float(t) for t in times) / len(times)
                loss = int(loss_match.group(1)) if loss_match else 0
                return {
                    "success": True,
                    "avg_latency": avg_time,
                    "packet_loss": loss,
                    "timestamp": datetime.now().isoformat()
                }
            return {
                "success": False,
```

```

        "error": result.stderr,
        "timestamp": datetime.now().isoformat()
    }

except subprocess.TimeoutExpired:
    return {
        "success": False,
        "error": "Ping timeout",
        "timestamp": datetime.now().isoformat()
    }

def traceroute_host(self, host):
    """Perform traceroute to a host"""
    import platform

    if platform.system().lower() == "windows":
        command = f"tracert -w 5000 {host}"
    else:
        command = f"traceroute -w 5 {host}"

    try:
        result = subprocess.run(command, shell=True, capture_output=True, text=True, timeout=60)

        return {
            "success": result.returncode == 0,
            "output": result.stdout,
            "timestamp": datetime.now().isoformat()
        }

    except subprocess.TimeoutExpired:
        return {
            "success": False,
            "error": "Traceroute timeout",
            "timestamp": datetime.now().isoformat()
        }

def check_thresholds(self, stats):
    """Check if statistics exceed thresholds"""
    alerts = []

    if stats["success"]:
        latency = stats["avg_latency"]
        loss = stats["packet_loss"]

        if latency > self.config["thresholds"]["latency_critical"]:
            alerts.append(f"CRITICAL: High latency ({latency}ms)")
        elif latency > self.config["thresholds"]["latency_warning"]:

```

```

        alerts.append(f"WARNING: High latency ({latency}ms)")

    if loss > self.config["thresholds"]["loss_critical"]:
        alerts.append(f"CRITICAL: High packet loss ({loss}%)")
    elif loss > self.config["thresholds"]["loss_warning"]:
        alerts.append(f"WARNING: High packet loss ({loss}%)")

return alerts

def monitor_targets(self):
    """Monitor all configured targets"""
    for target in self.config["targets"]:
        host = target["host"]
        name = target["name"]

        self.logger.info(f"Monitoring {name} ({host})")

        # Ping test
        ping_stats = self.ping_host(host)
        self.stats[host].append(ping_stats)

        # Check thresholds and generate alerts
        if ping_stats["success"]:
            alerts = self.check_thresholds(ping_stats)
            for alert in alerts:
                self.logger.warning(f"{name}: {alert}")
        else:
            self.logger.error(f"{name}: {ping_stats['error']}")

def generate_report(self):
    """Generate health report"""
    self.logger.info("Generating network health report")

    print("\n== Network Health Report ==")
    print(f"Generated: {datetime.now()}")
    print()

    for target in self.config["targets"]:
        host = target["host"]
        name = target["name"]

        if host in self.stats and self.stats[host]:
            recent_stats = self.stats[host][-10:] # Last 10 measurements
            successful = [s for s in recent_stats if s["success"]]

            if successful:
                avg_latency = sum(s["avg_latency"] for s in successful) / len(successful)

```

```

avg_loss = sum(s["packet_loss"] for s in successful) / len(successful)
availability = (len(successful) / len(recent_stats)) * 100

print(f"{name} ({host}):")
print(f" Availability: {availability:.1f}%")
print(f" Average Latency: {avg_latency:.1f}ms")
print(f" Average Loss: {avg_loss:.1f}%")
print()

else:
    print(f"{name} ({host}): No successful measurements")
    print()

def run(self):
    """Main monitoring loop"""
    self.logger.info("Starting network health monitoring")

    last_report = 0
    last_traceroute = 0

    try:
        while True:
            current_time = time.time()

            # Regular ping monitoring
            self.monitor_targets()

            # Periodic traceroute
            if current_time - last_traceroute > self.config["intervals"]["traceroute_interval"]:
                self.logger.info("Running traceroute diagnostics")
                for target in self.config["targets"]:
                    if target["type"] == "web": # Only traceroute web targets
                        traceroute_result = self.traceroute_host(target["host"])
                        if not traceroute_result["success"]:
                            self.logger.warning(f"Traceroute failed for {target['name']}")

                last_traceroute = current_time

            # Periodic reporting
            if current_time - last_report > self.config["intervals"]["report_interval"]:
                self.generate_report()
                last_report = current_time

            # Wait for next interval
            time.sleep(self.config["intervals"]["ping_interval"])

    except KeyboardInterrupt:
        self.logger.info("Monitoring stopped by user")
    except Exception as e:

```

```
self.logger.error(f"Monitoring error: {e}")

if __name__ == "__main__":
    monitor = NetworkHealthMonitor()
    monitor.run()
```

ISP Escalation Report Generator

ISP Escalation Report Script (Python)

python

```
#!/usr/bin/env python3
# isp_escalation_report.py

import subprocess
import sys
import json
from datetime import datetime, timedelta
from collections import defaultdict

class ISPEscalationReport:
    def __init__(self):
        self.report_data = {
            "timestamp": datetime.now().isoformat(),
            "tests_performed": [],
            "issue_classification": "",
            "recommended_actions": [],
            "escalation_priority": ""
        }

    def run_diagnostic_command(self, command, description):
        """Run a diagnostic command and capture output"""
        print(f"Running: {description}")

        try:
            result = subprocess.run(
                command,
                shell=True,
                capture_output=True,
                text=True,
                timeout=60
            )

            test_result = {
                "command": command,
                "description": description,
                "timestamp": datetime.now().isoformat(),
                "exit_code": result.returncode,
                "stdout": result.stdout,
                "stderr": result.stderr,
                "success": result.returncode == 0
            }

            self.report_data["tests_performed"].append(test_result)
            return test_result
        except subprocess.TimeoutExpired:
```

```

test_result = {
    "command": command,
    "description": description,
    "timestamp": datetime.now().isoformat(),
    "exit_code": -1,
    "stdout": "",
    "stderr": "Command timed out",
    "success": False
}
self.report_data["tests_performed"].append(test_result)
return test_result

def perform_comprehensive_diagnostics(self):
    """Perform comprehensive network diagnostics"""
    print("==== ISP Escalation Report Generator ===")
    print(f"Starting diagnostics at {datetime.now()}")
    print()

    # Platform-specific commands
    is_windows = sys.platform == "win32"

    # Basic connectivity tests
    self.run_diagnostic_command(
        "ping -n 4 8.8.8.8" if is_windows else "ping -c 4 8.8.8.8",
        "Test connectivity to Google DNS (8.8.8.8)"
    )

    self.run_diagnostic_command(
        "ping -n 4 1.1.1.1" if is_windows else "ping -c 4 1.1.1.1",
        "Test connectivity to Cloudflare DNS (1.1.1.1)"
    )

    # DNS resolution tests
    self.run_diagnostic_command(
        "nslookup google.com",
        "Test DNS resolution for google.com"
    )

    # Traceroute tests
    self.run_diagnostic_command(
        "tracert 8.8.8.8" if is_windows else "traceroute 8.8.8.8",
        "Traceroute to Google DNS (8.8.8.8)"
    )

    if not is_windows:
        # MTR test (Linux/macOS only)
        self.run_diagnostic_command(

```

```

        "mtr -c 10 -r 8.8.8.8",
        "MTR analysis to Google DNS (8.8.8.8)"
    )
else:
    # Pathping test (Windows only)
    self.run_diagnostic_command(
        "pathping -n -q 5 8.8.8.8",
        "Pathping analysis to Google DNS (8.8.8.8)"
    )

# Additional connectivity tests
self.run_diagnostic_command(
    "ping -n 4 google.com" if is_windows else "ping -c 4 google.com",
    "Test connectivity to google.com (by hostname)"
)

# Network interface information
if is_windows:
    self.run_diagnostic_command(
        "ipconfig /all",
        "Network interface configuration"
    )
else:
    self.run_diagnostic_command(
        "ip addr show",
        "Network interface configuration"
    )

self.run_diagnostic_command(
    "ip route show",
    "Routing table information"
)

def analyze_results(self):
    """Analyze test results and classify the issue"""
    print("\n==== Analyzing Results ===")

    # Count successful tests
    successful_tests = sum(1 for test in self.report_data["tests_performed"] if test["success"])
    total_tests = len(self.report_data["tests_performed"])

    # Analyze ping results
    ping_results = [test for test in self.report_data["tests_performed"] if "ping" in test["command"]]
    dns_results = [test for test in self.report_data["tests_performed"] if "nslookup" in test["command"]]
    traceroute_results = [test for test in self.report_data["tests_performed"] if any(cmd in test["command"] for cr

# Classification logic

```

```

if successful_tests == 0:
    self.report_data["issue_classification"] = "COMPLETE_CONNECTIVITY_FAILURE"
    self.report_data["escalation_priority"] = "CRITICAL"
elif any(not test["success"] for test in ping_results):
    self.report_data["issue_classification"] = "PARTIAL_CONNECTIVITY_FAILURE"
    self.report_data["escalation_priority"] = "HIGH"
elif any(not test["success"] for test in dns_results):
    self.report_data["issue_classification"] = "DNS_RESOLUTION_FAILURE"
    self.report_data["escalation_priority"] = "MEDIUM"
elif successful_tests < total_tests:
    self.report_data["issue_classification"] = "INTERMITTENT_CONNECTIVITY_ISSUES"
    self.report_data["escalation_priority"] = "MEDIUM"
else:
    self.report_data["issue_classification"] = "NO_ISSUES_DETECTED"
    self.report_data["escalation_priority"] = "LOW"

# Generate recommendations
self.generate_recommendations()

def generate_recommendations(self):
    """Generate recommendations based on analysis"""
    classification = self.report_data["issue_classification"]

    recommendations = {
        "COMPLETE_CONNECTIVITY_FAILURE": [
            "Verify all internal network connectivity first",
            "Check physical WAN connections",
            "Verify ISP circuit status",
            "Contact ISP immediately for circuit troubleshooting"
        ],
        "PARTIAL_CONNECTIVITY_FAILURE": [
            "Analyze traceroute results for failure points",
            "Check for ISP routing issues",
            "Verify ISP DNS server functionality",
            "Request ISP investigation of packet loss"
        ],
        "DNS_RESOLUTION_FAILURE": [
            "Check internal DNS configuration",
            "Verify ISP DNS servers are responding",
            "Test with alternative DNS servers",
            "Request ISP DNS server investigation"
        ],
        "INTERMITTENT_CONNECTIVITY_ISSUES": [
            "Monitor for pattern in connectivity issues",
            "Check for ISP network congestion",
            "Verify QoS policies with ISP",
            "Request ISP network stability analysis"
        ]
    }

```

```

        ],
    "NO_ISSUES_DETECTED": [
        "Issue may be application-specific",
        "Check internal network configuration",
        "Monitor for recurring issues",
        "Document for future reference"
    ]
}

self.report_data["recommended_actions"] = recommendations.get(classification, [])

def generate_report(self):
    """Generate the final escalation report"""
    print("\n==== ISP Escalation Report ====")
    print(f"Report Generated: {self.report_data['timestamp']}")
    print(f"Issue Classification: {self.report_data['issue_classification']}")
    print(f"Escalation Priority: {self.report_data['escalation_priority']}")
    print()

    print("EXECUTIVE SUMMARY:")
    print(f"- Total tests performed: {len(self.report_data['tests_performed'])}")
    print(f"- Successful tests: {sum(1 for test in self.report_data['tests_performed'] if test['success'])}")
    print(f"- Failed tests: {sum(1 for test in self.report_data['tests_performed'] if not test['success'])}")
    print()

    print("DETAILED TEST RESULTS:")
    for i, test in enumerate(self.report_data["tests_performed"], 1):
        status = "✓ PASS" if test["success"] else "✗ FAIL"
        print(f"{i}. {test['description']}: {status}")
        if not test["success"] and test["stderr"]:
            print(f"  Error: {test['stderr']}")
    print()

    print("RECOMMENDED ACTIONS:")
    for i, action in enumerate(self.report_data["recommended_actions"], 1):
        print(f"{i}. {action}")
    print()

    print("TECHNICAL DETAILS:")
    print("The following diagnostic commands were executed:")
    for test in self.report_data["tests_performed"]:
        print(f"- {test['command']}")
    print()

    print("Complete command output is available in the JSON report file.")

def save_json_report(self, filename="isp_escalation_report.json"):

```

```

"""Save detailed report to JSON file"""
with open(filename, 'w') as f:
    json.dump(self.report_data, f, indent=2)
    print(f"Detailed report saved to: {filename}")

def run(self):
    """Main execution method"""
    try:
        self.perform_comprehensive_diagnostics()
        self.analyze_results()
        self.generate_report()
        self.save_json_report()

    except KeyboardInterrupt:
        print("\nReport generation interrupted by user")
    except Exception as e:
        print(f"Error during report generation: {e}")

if __name__ == "__main__":
    report_generator = ISPEscalationReport()
    report_generator.run()

```

CONCLUSION

This comprehensive guide provides network engineers with the systematic methodology, tools, and scripts necessary to effectively troubleshoot network connectivity issues and accurately determine whether problems originate from enterprise infrastructure or ISP services.

The key to successful network troubleshooting lies in following a structured approach, maintaining proper documentation, and using the right tools for each situation. By implementing the practices outlined in this guide, network engineers can:

- Reduce mean time to resolution (MTTR) for network issues
- Improve accuracy in problem identification and responsibility assignment
- Enhance communication with ISP support teams
- Establish proactive monitoring and alerting systems
- Build comprehensive troubleshooting capabilities

Remember that network troubleshooting is an iterative process that requires patience, systematic thinking, and thorough documentation. Always correlate multiple data points before drawing conclusions, and maintain detailed records of all diagnostic activities for future reference and trend analysis.

Final Recommendations:

1. **Practice regularly** with these tools and techniques in controlled environments
2. **Customize scripts** and procedures to match your specific network infrastructure
3. **Maintain current documentation** of network topology and baseline performance
4. **Establish clear escalation procedures** with defined thresholds and contact information
5. **Regularly review and update** monitoring thresholds based on network changes and performance trends

This guide serves as a foundation for building robust network troubleshooting capabilities. Continue to expand your knowledge through hands-on experience, vendor-specific training, and staying current with evolving network technologies and diagnostic tools.