Network Monitoring System - Risk Assessment Matrix

Table of Contents

Executive Summary

This risk assessment matrix identifies, analyzes, and prioritizes risks associated with the Network Monitoring System. The assessment covers technical, security, operational, business, and compliance risks, providing mitigation strategies and contingency plans.

Risk Summary

- **Critical Risks**: 3 identified
- **High Risks**: 8 identified
- **Medium Risks**: 12 identified
- **Low Risks**: 7 identified
- **Overall Risk Score**: 6.2/10 (Medium-High)

Key Risk Areas

- **Security**: Elevated privileges and data exposure
- **Performance**: Scalability limitations under high load
- **Compliance**: Data privacy and retention requirements

Operational: System availability and maintenance

Risk Assessment Framework

Risk Scoring Methodology

Probability Scale (1-5)

1 - Very Low: <5% chance of occurrence
2 - Low: 5-25% chance of occurrence
3 - Medium: 25-50% chance of occurrence
4 - High: 50-75% chance of occurrence
5 - Very High: >75% chance of occurrence

Impact Scale (1-5)

1 - Very Low: Minimal impact, easily recoverable
2 - Low: Minor impact, short-term effects
3 - Medium: Moderate impact, some business disruption
4 - High: Significant impact, major business disruption
5 - Very High: Severe impact, critical business failure

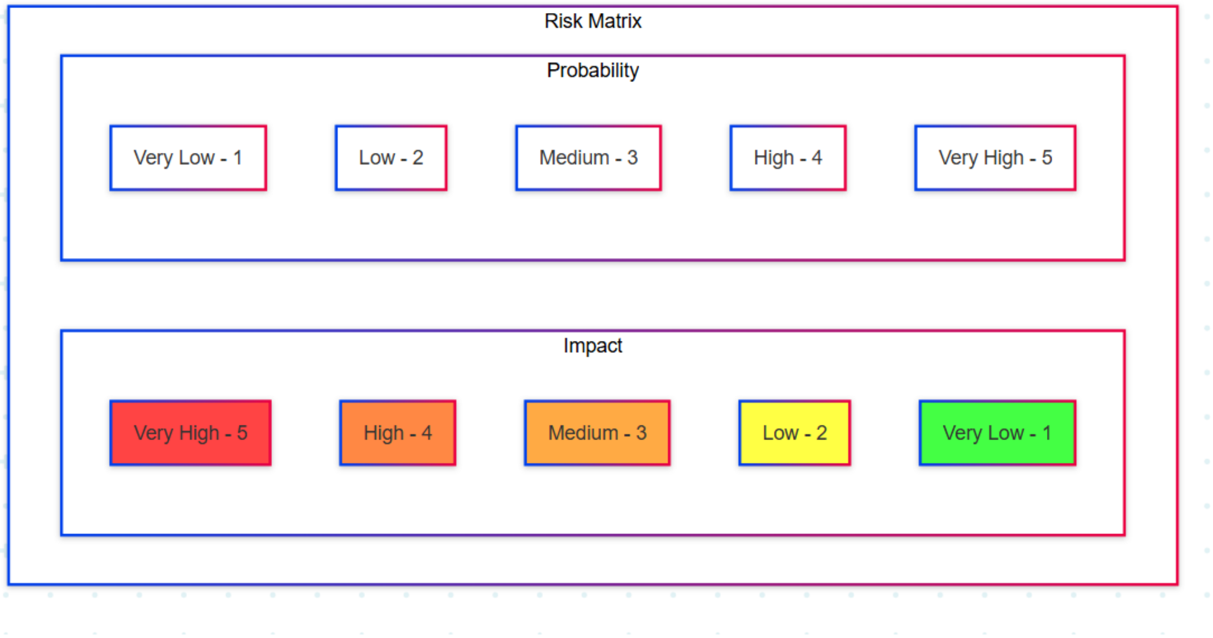
Risk Score Calculation

Risk Score = Probability × Impact

Risk Categories

1-4: Low Risk (Green)
5-9: Medium Risk (Yellow)
10-16: High Risk (Orange)
17-25: Critical Risk (Red)

Risk Matrix Visualization



```
graph TB
    subgraph "Risk Matrix"
        subgraph "Impact"
            I5[Very High - 5]
            I4[High - 4]
            I3[Medium - 3]
            I2[Low - 2]
            I1[Very Low - 1]
        end

        subgraph "Probability"
            P1[Very Low - 1]
            P2[Low - 2]
            P3[Medium - 3]
            P4[High - 4]
            P5[Very High - 5]
        end
    end

    style I5 fill:#ff4444
    style I4 fill:#ff8844
    style I3 fill:#ffaa44
    style I2 fill:#ffff44
    style I1 fill:#44ff44
```

Technical Risks

TECH-001: System Performance Degradation

Category: Technical **Probability**: 4 (High) **Impact**: 4 (High) **Risk Score**: 16 (High) **File Reference**: `src/core/NetworkMonitor.cpp`

Description: System performance may degrade under high network traffic loads, leading to packet loss and delayed processing.

Risk Factors:

Single-threaded packet capture bottleneck
Memory allocation overhead in packet processing
Database write performance limitations
GUI update frequency causing UI freezing

Potential Consequences:

Packet loss exceeding 1% threshold
Real-time monitoring delays
System unresponsiveness
Inaccurate network statistics

Mitigation Strategies:

Immediate (1-2 weeks):
Implement lock-free queues for packet processing
Optimize database batch insertion
Reduce GUI update frequency during high load
Short-term (1-3 months):
Implement multi-threaded packet processing
Add memory pooling for packet objects
Implement adaptive performance scaling
Long-term (3-6 months):
Migrate to distributed architecture

- Implement horizontal scaling capabilities

Monitoring Indicators:

- Packet loss rate > 1%
- Processing latency > 100ms
- Memory usage > 1GB
- CPU utilization > 80%

TECH-002: Database Corruption

Category: Technical **Probability**: 2 (Low) **Impact**: 5 (Very High) **Risk Score**: 10 (High) **File Reference**: `src/storage/DataStore.cpp`

Description: SQLite database corruption could result in complete loss of historical network data.

Risk Factors:

- High-frequency write operations
- Improper shutdown procedures
- Disk space exhaustion
- Concurrent access conflicts

Potential Consequences:

- Complete loss of historical data
- System startup failures
- Inability to generate reports
- Compliance violations

Mitigation Strategies:

- **Immediate**:
- Implement database backup procedures
- Add database integrity checks
- Enable SQLite WAL mode
- **Short-term**:
- Implement database replication
- Add automated backup verification
- Implement graceful shutdown procedures

Long-term:
Migrate to PostgreSQL for better reliability
Implement distributed storage

Monitoring Indicators:

Database integrity check failures
Write operation errors
Disk space < 10% free
Backup verification failures

TECH-003: Memory Leaks

Category: Technical **Probability**: 3 (Medium) **Impact**: 3 (Medium) **Risk Score**: 9 (Medium)
File Reference: `src/protocols/Packet.cpp`

Description: Memory leaks in packet processing could lead to system instability over time.

Risk Factors:

Manual memory management in C++
Exception handling gaps
Circular references in packet objects
Third-party library memory issues

Potential Consequences:

Gradual system slowdown
System crashes after extended operation
Resource exhaustion
Service interruption

Mitigation Strategies:

Immediate:
Implement comprehensive memory leak testing
Add memory usage monitoring
Use smart pointers consistently
Short-term:
Implement memory pooling
Add automated memory leak detection

Regular memory profiling

Monitoring Indicators:

Memory usage growth > 10MB/hour
Valgrind leak detection alerts
System performance degradation over time

TECH-004: Network Interface Failures

Category: Technical **Probability**: 3 (Medium) **Impact**: 4 (High) **Risk Score**: 12 (High) **File Reference**: `src/core/NetworkMonitor.cpp:45-67`

Description: Network interface failures or disconnections could interrupt packet capture.

Risk Factors:

Hardware failures
Driver issues
Network cable disconnections
Interface configuration changes

Potential Consequences:

Complete monitoring interruption
Loss of real-time visibility
Missed security events
Compliance gaps

Mitigation Strategies:

Immediate:
Implement interface health monitoring
Add automatic interface reconnection
Implement failover to backup interfaces
Short-term:
Support multiple interface monitoring
Add interface redundancy
Implement network bonding support

Monitoring Indicators:

- Interface down events
- Packet capture interruptions
- Network connectivity failures
- Driver error messages

Security Risks

SEC-001: Privilege Escalation

Category: Security **Probability**: 4 (High) **Impact**: 5 (Very High) **Risk Score**: 20 (Critical) **File Reference**: `src/main.cpp:35-45`

Description: Application running with root privileges throughout execution creates privilege escalation risks.

Risk Factors:

- Persistent root privileges
- No privilege dropping after initialization
- Potential code injection vulnerabilities
- Third-party library vulnerabilities

Potential Consequences:

- Complete system compromise
- Unauthorized access to sensitive data
- Lateral movement in network
- Compliance violations

Mitigation Strategies:

- **Immediate** (Critical Priority):
- Implement privilege dropping after interface initialization
- Add capability-based security model
- Implement process isolation
- **Short-term**:
- Add user authentication and authorization
- Implement role-based access control
- Regular security audits

Monitoring Indicators:

- Unauthorized privilege usage
- Suspicious process activities
- Security audit failures
- Unusual system calls

SEC-002: Data Exposure

Category: Security **Probability**: 3 (Medium) **Impact**: 4 (High) **Risk Score**: 12 (High) **File Reference**: `src/storage/DataStore.cpp:89-112`

Description: Unencrypted storage of sensitive network data could lead to data breaches.

Risk Factors:

- Plaintext database storage
- Unencrypted log files
- Inadequate file permissions
- Backup security gaps

Potential Consequences:

- Sensitive data exposure
- Privacy violations
- Regulatory penalties
- Reputation damage

Mitigation Strategies:

- **Immediate**:
- Implement database encryption
- Secure file permissions
- Encrypt backup files
- **Short-term**:
- Add data classification and handling
- Implement data anonymization
- Regular security assessments

Monitoring Indicators:

Unauthorized file access
Data export activities
Backup security violations
Encryption key compromises

SEC-003: Authentication Bypass

Category: Security **Probability**: 3 (Medium) **Impact**: 4 (High) **Risk Score**: 12 (High) **File Reference**: `src/gui/MainWindow.cpp`

Description: Lack of authentication mechanisms allows unauthorized access to monitoring data.

Risk Factors:

No user authentication
Direct GUI access
No session management
Missing access controls

Potential Consequences:

Unauthorized monitoring access
Data theft
System manipulation
Compliance violations

Mitigation Strategies:

Immediate:
Implement user authentication
Add session management
Implement access logging
Short-term:
Add multi-factor authentication
Implement role-based permissions
Regular access reviews

Monitoring Indicators:

Unauthorized login attempts

Suspicious user activities
Access pattern anomalies
Failed authentication events

SEC-004: Input Validation Vulnerabilities

Category: Security **Probability**: 3 (Medium) **Impact**: 3 (Medium) **Risk Score**: 9 (Medium)
File Reference: `src/config/ConfigManager.cpp`

Description: Insufficient input validation could lead to injection attacks or system compromise.

Risk Factors:

Limited BPF filter validation
Configuration file parsing vulnerabilities
Command injection possibilities
Buffer overflow risks

Potential Consequences:

Code injection attacks
System crashes
Data corruption
Denial of service

Mitigation Strategies:

Immediate:
Implement comprehensive input validation
Add sanitization for all user inputs
Use parameterized queries
Short-term:
Regular security code reviews
Automated vulnerability scanning
Penetration testing

Monitoring Indicators:

Input validation failures
Malformed request attempts

Unusual input patterns
Security scanner alerts

Operational Risks

OPS-001: System Availability

Category: Operational **Probability**: 3 (Medium) **Impact**: 4 (High) **Risk Score**: 12 (High)

Description: System downtime could result in loss of network monitoring capabilities and security blind spots.

Risk Factors:

Single point of failure
No redundancy mechanisms
Manual restart procedures
Dependency failures

Potential Consequences:

Monitoring service interruption
Security event misses
Compliance violations
Business impact

Mitigation Strategies:

Immediate:
Implement health monitoring
Add automatic restart capabilities
Create redundant deployments
Short-term:
Implement high availability architecture
Add load balancing
Automated failover procedures

Monitoring Indicators:

Service uptime < 99.9%

Health check failures
Restart frequency > 1/day
Dependency unavailability

OPS-002: Data Loss

Category: Operational **Probability**: 2 (Low) **Impact**: 4 (High) **Risk Score**: 8 (Medium) **File Reference**: `src/storage/DataStore.cpp`

Description: Data loss could occur due to hardware failures, software bugs, or operational errors.

Risk Factors:

No backup procedures
Single storage location
Hardware failures
Human errors

Potential Consequences:

Historical data loss
Compliance violations
Investigation limitations
Business continuity impact

Mitigation Strategies:

Immediate:
Implement automated backups
Add backup verification
Create disaster recovery procedures
Short-term:
Implement data replication
Add point-in-time recovery
Regular backup testing

Monitoring Indicators:

Backup failure alerts
Storage capacity warnings

Data integrity check failures
Recovery time objectives exceeded

OPS-003: Maintenance Windows

Category: Operational **Probability**: 4 (High) **Impact**: 2 (Low) **Risk Score**: 8 (Medium)

Description: Regular maintenance activities could impact monitoring availability.

Risk Factors:

Required system updates
Database maintenance
Hardware maintenance
Configuration changes

Potential Consequences:

Temporary monitoring gaps
Service interruptions
User inconvenience
Delayed incident response

Mitigation Strategies:

Immediate:
Schedule maintenance windows
Implement rolling updates
Add maintenance notifications
Short-term:
Implement zero-downtime deployments
Add blue-green deployment
Automated maintenance procedures

Monitoring Indicators:

Maintenance frequency
Downtime duration
User impact metrics
Update success rates

Business Risks

BUS-001: Scalability Limitations

Category: Business **Probability**: 4 (High) **Impact**: 3 (Medium) **Risk Score**: 12 (High)

Description: Current architecture may not scale to meet growing business requirements.

Risk Factors:

Single-node architecture
SQLite limitations
Memory constraints
Processing bottlenecks

Potential Consequences:

Inability to handle increased traffic
Performance degradation
Business growth limitations
Competitive disadvantage

Mitigation Strategies:

Immediate:
Performance optimization
Resource monitoring
Capacity planning
Long-term:
Distributed architecture migration
Cloud-native deployment
Horizontal scaling implementation

Monitoring Indicators:

Resource utilization trends
Performance degradation
Capacity thresholds
Growth rate metrics

BUS-002: Technology Obsolescence

Category: Business **Probability**: 3 (Medium) **Impact**: 3 (Medium) **Risk Score**: 9 (Medium)

Description: Technology stack may become obsolete, requiring significant modernization efforts.

Risk Factors:

- Aging dependencies
- End-of-life software
- Security vulnerabilities
- Lack of vendor support

Potential Consequences:

- Security vulnerabilities
- Maintenance difficulties
- Integration challenges
- Modernization costs

Mitigation Strategies:

- **Immediate**:
- Regular dependency updates
- Technology roadmap planning
- Vendor relationship management
- **Long-term**:
- Gradual modernization
- Technology migration planning
- Skills development

Monitoring Indicators:

- Dependency age metrics
- Security vulnerability counts
- Vendor support status
- Technology trend analysis

BUS-003: Resource Constraints

Category: Business **Probability**: 3 (Medium) **Impact**: 3 (Medium) **Risk Score**: 9 (Medium)

Description: Limited development and operational resources may impact system evolution and maintenance.

Risk Factors:

- Limited development team
- Budget constraints
- Skill gaps
- Competing priorities

Potential Consequences:

- Delayed feature development
- Maintenance backlogs
- Quality compromises
- Technical debt accumulation

Mitigation Strategies:

- **Immediate**:
- Resource planning
- Priority management
- Skills assessment
- **Long-term**:
- Team expansion
- Training programs
- Process optimization

Monitoring Indicators:

- Development velocity
- Maintenance backlog size
- Team utilization rates
- Quality metrics

Compliance Risks

COMP-001: Data Privacy Violations

Category: Compliance **Probability**: 3 (Medium) **Impact**: 4 (High) **Risk Score**: 12 (High)

Description: Inadequate data privacy controls could lead to GDPR, CCPA, or other privacy regulation violations.

Risk Factors:

- No data anonymization
- Unclear data retention policies
- Missing consent mechanisms
- Inadequate data subject rights

Potential Consequences:

- Regulatory fines
- Legal liability
- Reputation damage
- Business restrictions

Mitigation Strategies:

- **Immediate**:
- Implement data retention policies
- Add data anonymization
- Create privacy documentation
- **Short-term**:
- Privacy impact assessments
- Data subject rights implementation
- Regular compliance audits

Monitoring Indicators:

- Data retention violations
- Privacy request volumes
- Compliance audit findings
- Regulatory changes

COMP-002: Audit Trail Deficiencies

Category: Compliance **Probability**: 2 (Low) **Impact**: 3 (Medium) **Risk Score**: 6 (Low) **File Reference**: `src/utils/Logger.cpp`

Description: Insufficient audit trails could impact compliance with security and financial regulations.

Risk Factors:

- Limited audit logging
- No log integrity protection
- Missing user activity tracking
- Inadequate log retention

Potential Consequences:

- Compliance failures
- Investigation difficulties
- Regulatory penalties
- Audit findings

Mitigation Strategies:

- **Immediate**:
- Enhance audit logging
- Implement log integrity protection
- Add user activity tracking
- **Short-term**:
- Automated compliance reporting
- Log analysis tools
- Regular audit procedures

Monitoring Indicators:

- Audit log completeness
- Log integrity violations
- Compliance report accuracy
- Audit findings

COMP-003: Data Retention Violations

Category: Compliance **Probability**: 3 (Medium) **Impact**: 3 (Medium) **Risk Score**: 9 (Medium) **File Reference**: `src/storage/DataStore.cpp`

Description: Improper data retention could violate regulatory requirements or organizational policies.

Risk Factors:

- No automated retention policies
- Manual cleanup procedures
- Unclear retention requirements
- Storage capacity constraints

Potential Consequences:

- Regulatory violations
- Storage cost increases
- Performance degradation
- Compliance audit failures

Mitigation Strategies:

- **Immediate**:
- Implement automated retention policies
- Define retention requirements
- Add retention monitoring
- **Short-term**:
- Automated data archival
- Compliance reporting
- Regular policy reviews

Monitoring Indicators:

- Data age metrics
- Storage utilization
- Retention policy violations
- Cleanup operation success

Risk Mitigation Strategies

Immediate Actions (0-30 days)

Critical Risk Mitigation

```
// TECH-001: Performance optimization
class PerformanceOptimizer {
public:
    void optimizePacketProcessing() {
        // Implement lock-free queues
        packet_queue_ = std::make_unique<LockFreeQueue<Packet>>();

        // Optimize database configuration
        data_store_->enableWALMode();
        data_store_->setBatchSize(5000);

        // Reduce GUI update frequency during high load
        if (getCurrentPacketRate() > HIGH_LOAD_THRESHOLD) {
            gui_update_interval_ = std::chrono::seconds(5);
        }
    }
};

// SEC-001: Privilege dropping implementation
class PrivilegeManager {
public:
    void dropPrivileges() {
        // Drop to non-privileged user after initialization
        if (getuid() == 0) {
            if (setuid(getpwnam("network-monitor")->pw_uid) != 0) {
                throw std::runtime_error("Failed to drop privileges");
            }
            Logger::info("Successfully dropped root privileges");
        }
    }
};
```

Security Hardening

```
// Input validation framework
class InputValidator {
public:
    bool validateBPFFilter(const std::string& filter) {
        // Validate BPF filter syntax
        if (filter.length() > MAX_FILTER_LENGTH) return false;
        if (containsUnsafeCharacters(filter)) return false;

        // Test filter compilation
        struct bpf_program fp;
        if (pcap_compile_nopcap(65535, DLT_EN10MB, &fp,
                               filter.c_str(), 1,
PCAP_NETMASK_UNKNOWN) == -1) {
            return false;
        }

        pcap_freecode(&fp);
        return true;
    }

private:
    static constexpr size_t MAX_FILTER_LENGTH = 1024;

    bool containsUnsafeCharacters(const std::string& input) {
        const std::string unsafe_chars = ";&|`$(){}[]";
        return input.find_first_of(unsafe_chars) != std::string::npos;
    }
};
```

Short-term Actions (1-3 months)

High Availability Implementation

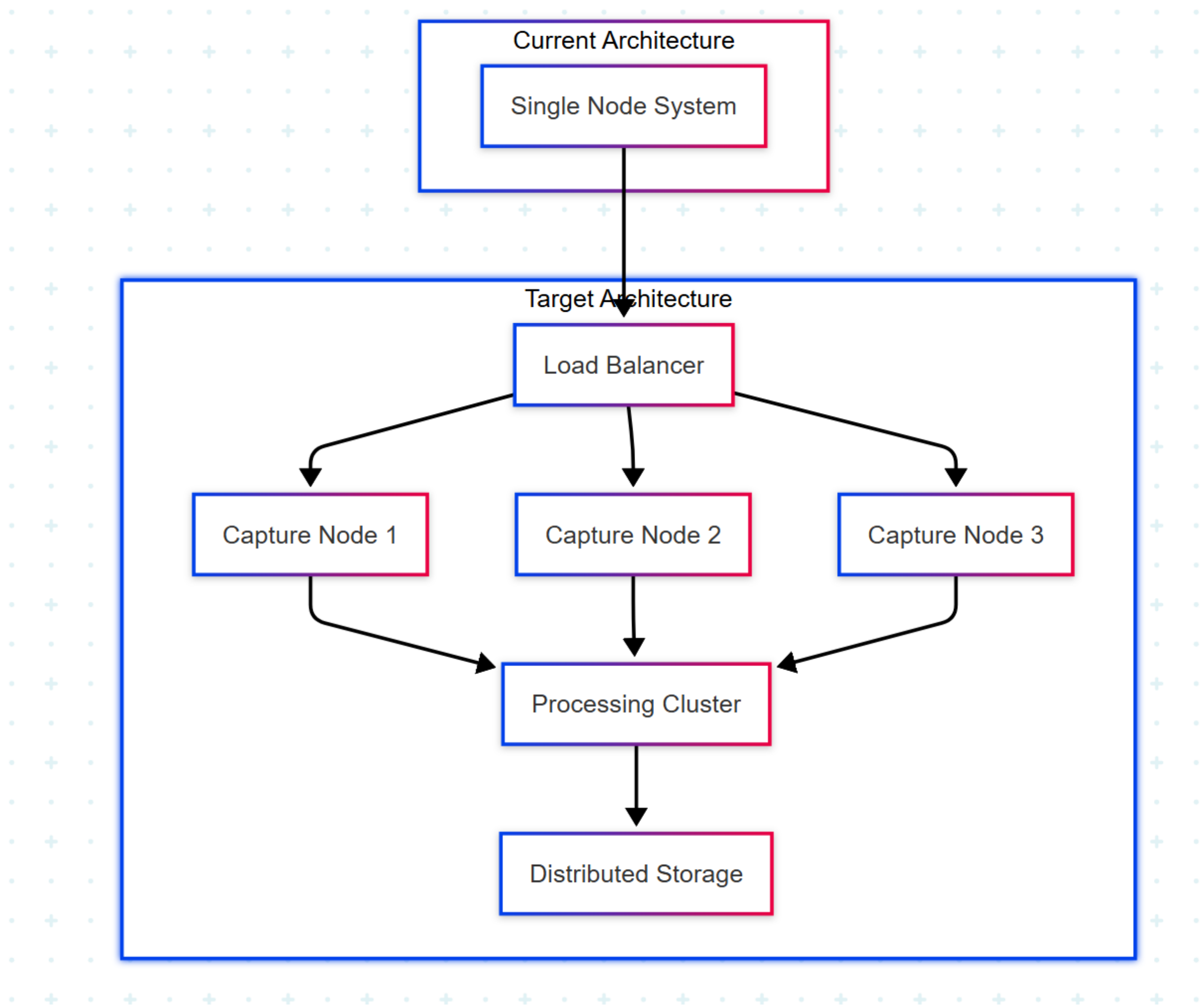
```
class HighAvailabilityManager {
private:
    std::vector<std::string> backup_interfaces_;
    std::atomic<bool> primary_interface_active_{true};

public:
    void initializeFailover() {
        // Monitor primary interface health
        health_monitor_ = std::make_unique<InterfaceHealthMonitor>();
        health_monitor_->onFailure([this](const std::string&
interface) {
            handleInterfaceFailure(interface);
        });

        // Configure backup interfaces
        for (const auto& backup : backup_interfaces_) {
            configureBackupInterface(backup);
        }
    }

private:
    void handleInterfaceFailure(const std::string& failed_interface) {
        Logger::warning("Interface failure detected: " +
failed_interface);

        // Failover to backup interface
        if (!backup_interfaces_.empty()) {
            switchToBackupInterface(backup_interfaces_[0]);
        }

        // Notify operations team
        NotificationService::sendAlert("Interface failover occurred");
    }
};
```

Data Protection Enhancement

```
class DataProtectionManager {
public:
    void implementEncryption() {
        // Database encryption
        data_store_->enableEncryption(encryption_key_);

        // Log file encryption
        logger_->enableEncryption(log_encryption_key_);

        // Backup encryption
        backup_manager_->enableEncryption(backup_encryption_key_);
    }

    void implementBackupStrategy() {
        // Automated daily backups
        backup_scheduler_->scheduleDaily([this]() {
            performIncrementalBackup();
        });

        // Weekly full backups
        backup_scheduler_->scheduleWeekly([this]() {
            performFullBackup();
        });

        // Backup verification
        backup_scheduler_->scheduleDaily([this]() {
            verifyBackupIntegrity();
        });
    }
};
```

Long-term Actions (3-12 months)

Distributed Architecture Migration



```
graph TB
    subgraph "Current Architecture"
        SINGLE[Single Node System]
    end

    subgraph "Target Architecture"
        LB[Load Balancer]
        CAP1[Capture Node 1]
        CAP2[Capture Node 2]
        CAP3[Capture Node 3]
        PROC[Processing Cluster]
        STORE[Distributed Storage]
    end

    SINGLE --> LB
    LB --> CAP1
    LB --> CAP2
    LB --> CAP3
    CAP1 --> PROC
    CAP2 --> PROC
    CAP3 --> PROC
    PROC --> STORE
```

Cloud-Native Migration

```
# Kubernetes deployment for scalability
apiVersion: apps/v1
kind: Deployment
metadata:
  name: network-monitor
spec:
  replicas: 3
  selector:
    matchLabels:
      app: network-monitor
  template:
    metadata:
      labels:
        app: network-monitor
    spec:
      containers:
      - name: monitor
        image: network-monitor:latest
        resources:
          requests:
            memory: "512Mi"
            cpu: "500m"
          limits:
            memory: "1Gi"
            cpu: "1000m"
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: db-secret
              key: url
```

Risk Monitoring and Review

Risk Monitoring Framework

Automated Risk Indicators

```
class RiskMonitor {
private:
    struct RiskMetric {
        std::string name;
        double threshold;
        std::function<double()> measurement;
        RiskLevel level;
    };

    std::vector<RiskMetric> metrics_;

public:
    void initializeMetrics() {
        metrics_ = {
            {"packet_loss_rate", 0.01, []() { return
getPacketLossRate(); }, RiskLevel::HIGH},
            {"memory_usage_gb", 1.0, []() { return
getMemoryUsageGB(); }, RiskLevel::MEDIUM},
            {"cpu_utilization", 0.8, []() { return
getCPUUtilization(); }, RiskLevel::MEDIUM},
            {"disk_usage", 0.9, []() { return getDiskUsage(); },
RiskLevel::HIGH},
            {"authentication_failures", 10, []() { return
getAuthFailures(); }, RiskLevel::HIGH}
```

```
        };
    }

    void monitorRisks() {
        for (const auto& metric : metrics_) {
            double current_value = metric.measurement();

            if (current_value > metric.threshold) {
                RiskAlert alert{
                    .metric_name = metric.name,
                    .current_value = current_value,
                    .threshold = metric.threshold,
                    .risk_level = metric.level,
                    .timestamp = std::chrono::system_clock::now()
                };

                handleRiskAlert(alert);
            }
        }
    }
};
```

Risk Dashboard

```
class RiskDashboard {
public:
    struct RiskSummary {
        size_t critical_risks;
        size_t high_risks;
        size_t medium_risks;
        size_t low_risks;
        double overall_risk_score;
        std::vector<std::string> active_alerts;
    };

    RiskSummary generateRiskSummary() {
        RiskSummary summary;
        // Calculate risk counts
        summary.critical_risks = 
countRisksByLevel(RiskLevel::CRITICAL);
        summary.high_risks = countRisksByLevel(RiskLevel::HIGH);
        summary.medium_risks = countRisksByLevel(RiskLevel::MEDIUM);
        summary.low_risks = countRisksByLevel(RiskLevel::LOW);

        // Calculate overall risk score
        summary.overall_risk_score = calculateOverallRiskScore();

        // Get active alerts
        summary.active_alerts = getActiveAlerts();

        return summary;
    }
};
```

Risk Review Process

Monthly Risk Review

Risk Assessment Update: Review and update risk probabilities and impacts

Mitigation Progress: Assess progress on risk mitigation activities

New Risk Identification: Identify new risks from system changes

Risk Trend Analysis: Analyze risk trends and patterns

Quarterly Risk Assessment

Comprehensive Risk Review: Full review of all identified risks

Risk Appetite Review: Reassess organizational risk tolerance

Mitigation Strategy Update: Update risk mitigation strategies

Stakeholder Communication: Communicate risk status to stakeholders

Annual Risk Strategy Review

Risk Framework Review: Evaluate and update risk assessment framework

Risk Management Maturity: Assess risk management process maturity

Industry Benchmark: Compare risk posture with industry standards

Strategic Risk Planning: Align risk management with business strategy

Contingency Plans

Critical System Failure Response

Incident Response Procedure

```
flowchart TD
    A[System Failure Detected] --> B{Severity Assessment}
    B -->|Critical| C[Activate Emergency Response]
    B -->|High| D[Activate Standard Response]
    B -->|Medium/Low| E[Standard Troubleshooting]

    C --> F[Notify Emergency Contacts]
    C --> G[Implement Emergency Procedures]
    C --> H[Activate Backup Systems]

    D --> I[Notify Operations Team]
    D --> J[Implement Recovery Procedures]

    E --> K[Standard Resolution Process]

    F --> L[Incident Management]
    G --> L
    H --> L
    I --> L
    J --> L
    K --> L

    L --> M[Post-Incident Review]
```

Emergency Contact List

Role	Primary Contact	Secondary Contact	Escalation Time

System Administrator	admin@company.com	+1-555-ADMIN	15 minutes
Security Team	security@company.com	+1-555-SECURITY	30 minutes
Network Operations	netops@company.com	+1-555-NETOPS	15 minutes
Management	manager@company.com	+1-555-MANAGER	60 minutes

Data Recovery Procedures

Backup Recovery Process

```
class DisasterRecoveryManager {
public:
    struct RecoveryPlan {
        std::string backup_location;
        std::chrono::hours recovery_time_objective;
        std::chrono::hours recovery_point_objective;
        std::vector<std::string> recovery_steps;
    };

    bool executeRecovery(const RecoveryPlan& plan) {
        Logger::info("Starting disaster recovery procedure");

        try {
            // Step 1: Verify backup integrity
            if (!verifyBackupIntegrity(plan.backup_location)) {
                Logger::error("Backup integrity verification failed");
                return false;
            }

            // Step 2: Stop current services
            stopAllServices();

            // Step 3: Restore database
            if (!restoreDatabase(plan.backup_location)) {
                Logger::error("Database restoration failed");
                return false;
            }

            // Step 4: Restore configuration
            if (!restoreConfiguration(plan.backup_location)) {
                Logger::error("Configuration restoration failed");
                return false;
            }

            // Step 5: Restart services
            startAllServices();

            // Step 6: Verify system functionality
            if (!verifySystemHealth()) {
                Logger::error("System health verification failed");
                return false;
            }

            Logger::info("Disaster recovery completed successfully");
            return true;

        } catch (const std::exception& e) {
            Logger::error("Disaster recovery failed: " +
std::string(e.what()));
            return false;
        }
    }
};
```

Business Continuity Plans

Alternative Monitoring Solutions

Temporary Manual Monitoring: Manual network analysis procedures

Third-party Tools: Backup monitoring tools and services

Reduced Functionality Mode: Core monitoring with limited features

Partner Solutions: Temporary use of partner monitoring systems

Communication Plans

Internal Communication: Stakeholder notification procedures
Customer Communication: Customer impact notification
Regulatory Communication: Compliance authority notification
Public Communication: Public relations and media response

Risk Escalation Matrix

Risk Level	Response Time	Escalation Path	Authority Level
Critical	15 minutes	CTO → CEO → Board	Executive
High	1 hour	Manager → Director → CTO	Senior Management
Medium	4 hours	Team Lead → Manager	Management
Low	24 hours	Individual → Team Lead	Operational

Generated on: $(date) Risk Assessment Version: 1.0 Next Review Date: $(date +3 months) Classification: Internal Use Only