Fault tolerance and security testing Deepseek

I have this network which is in csv format exported from neo4j , a network created with the cypher language. I want to make this network fault tolerant, give me the commands in cypher to change the network so that it is fault tolerant.

n,r,m

"(:Switch {grpc_address: 127.0.0.1:50051,device_id: 0,json_path: build/basic.json,model: bmv2,id: s1})","[:INTERFACE {port: 1,name: s1-eth1,mac: 00:11:22:33:44:55}]","(:Host {ip: 10.0.0.1/24,id: h1,default_gw: 10.0.0.254})"

"(:Switch {grpc_address: 127.0.0.1:50051,device_id: 0,json_path: build/basic.json,model: bmv2,id: s1})",[:INTERFACE {name: s1-eth2}],"(:Host {ip: 10.0.0.2/24,id: h2,default_gw: 10.0.0.254})"

"(:Host {ip: 10.0.0.1/24,id: h1,default_gw: 10.0.0.254})",[:LINKS_TO],"(:Switch {grpc_address: 127.0.0.1:50051,device_id: 0,json_path: build/basic.json,model: bmv2,id: s1})"

"(:Host {ip: 10.0.0.2/24,id: h2,default_gw: 10.0.0.254})",[:LINKS_TO],"(:Switch {grpc_address: 127.0.0.1:50051,device_id: 0,json_path: build/basic.json,model: bmv2,id: s1})"

"(:Switch {grpc_address: 127.0.0.1:50051,device_id: 0,json_path: build/basic.json,model: bmv2,id: s1})",[:LINKS_TO {via: uplink-s1-s2}],"(:Switch {grpc_address: 127.0.0.2:50051,device_id: 1,json_path: build/basic.json,model: bmv2,id: s2})"

To make your network fault-tolerant, you need to add redundancy through multiple paths and switches. Here are the Cypher commands to modify your network:

1. Add a second switch for redundancy

```
CREATE (:Switch {
  grpc_address: "127.0.0.3:50051",
  device_id: 2,
  json_path: "build/basic.json",
  model: "bmv2",
  id: "s3",
  role: "backup"
})
```

2. Connecting hosts to multiple switches

```
// Connect h1 to the secondary switch
MATCH (h:Host {id: "h1"}), (s:Switch {id: "s3"})
CREATE (h)-[:LINKS_TO {interface: "h1-eth2", backup: true}]->(s),
       (s)-[:INTERFACE {
         port: 1,
         name: "s3-eth1",
         mac: "00:11:22:33:44:56",
         backup: true
       }]->(h)

// Connect h2 to the secondary switch
MATCH (h:Host {id: "h2"}), (s:Switch {id: "s3"})
CREATE (h)-[:LINKS_TO {interface: "h2-eth2", backup: true}]->(s),
       (s)-[:INTERFACE {
         port: 2,
```

```
    name: "s3-eth2",
    mac: "00:11:22:33:44:57",
    backup: true
  }]->(h)
```

3. Create redundant links between switches

```
// Primary link s1-s2

MATCH (s1:Switch {id: "s1"}), (s2:Switch {id: "s2"})
CREATE (s1)-[:LINKS_TO {
  via: "uplink-s1-s2-primary",
  bandwidth: "1Gbps",
  status: "active"
}]->(s2),
(s2)-[:LINKS_TO {
  via: "uplink-s2-s1-primary",
  bandwidth: "1Gbps",
  status: "active"
}]->(s1)

// Secondary link s1-s3
MATCH (s1:Switch {id: "s1"}), (s3:Switch {id: "s3"})
CREATE (s1)-[:LINKS_TO {
  via: "uplink-s1-s3-backup",
  bandwidth: "1Gbps",
  status: "standby"
}]->(s3),
(s3)-[:LINKS_TO {
  via: "uplink-s3-s1-backup",
```

```
  bandwidth: "1Gbps",
  status: "standby"
}]->(s1)

// s2-s3 link for complete redundancy
MATCH (s2:Switch {id: "s2"}), (s3:Switch {id: "s3"})
CREATE (s2)-[:LINKS_TO {
  via: "uplink-s2-s3-redundant",
  bandwidth: "1Gbps",
  status: "standby"
}]->(s3),
(s3)-[:LINKS_TO {
  via: "uplink-s3-s2-redundant",
  bandwidth: "1Gbps",
  status: "standby"
}]->(s2)
```

4. Add monitoring properties

```
// Add health check properties to switches
MATCH (s:Switch)
SET s.health_check = true,
    s.health_check_interval = "30s",
    s.failover_time = "500ms",
    s.redundancy_mode = "active-standby"
```

5. Create automatic failover policies

```
// Create failover relationships between switches
MATCH (s1:Switch {id: "s1"}), (s3:Switch {id: "s3"})
CREATE (s1)-[:FAILOVER_TO {priority: 1, auto_failback: true}]->(s3),
```

```
       (s3)-[:FAILOVER_TO {priority: 2, auto_failback: true}]->(s1)
```

6. Check the redundant topology

// Query to check redundancy

MATCH (h:Host)-[l:LINKS_TO]->(s:Switch)
RETURN h.id as Host, s.id as Switch, l.backup as BackupLink
ORDER BY h.id

7. Fault monitoring query

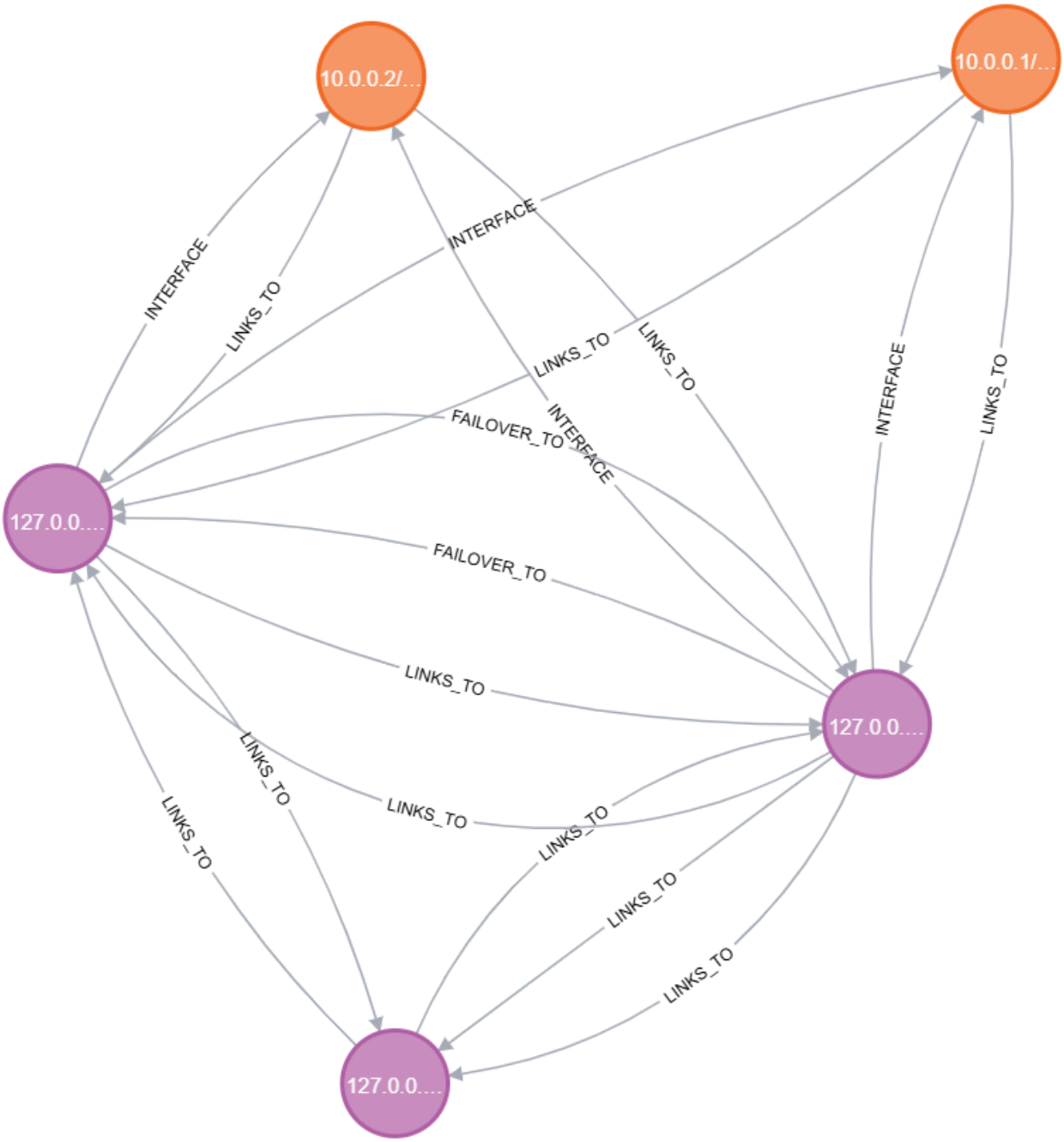
```
// Query to identify single points of failure
MATCH (n)
WHERE NOT (n)-[:FAILOVER_TO]-()
AND n:Switch OR n:Host
RETURN n.id as Node, labels(n)[0] as Type,
       "Potential single point of failure" as Status
```

Node	Type	Status
"h1"	"Host"	"Potential single point of failure"
"h2"	"Host"	"Potential single point of failure"
"s2"	"Switch"	"Potential single point of failure"

Host	Switch	BackupLink
"h1"	"s3"	true
"h1"	"s1"	null
"h2"	"s3"	true
"h2"	"s1"	null

These commands will create a redundant topology where:

- Each host has a connection to multiple switches
- Switches are interconnected with redundant links
- There is automatic failover capability
- The system can survive the failure of any single component


10.0.0.2/...
10.0.0.1/...
127.0.0....
127.0.0....
127.0.0....
INTERFACE
LINKS_TO
INTERFACE
LINKS_TO
LINKS_TO
INTERFACE
LINKS_TO
FAILOVER_TO
INTERFACE
FAILOVER_TO
LINKS_TO
LINKS_TO
LINKS_TO
LINKS_TO
LINKS_TO
LINKS_TO
LINKS_TO
LINKS_TO

Security

1. Adding security properties to switches

```
MATCH (s:Switch)
SET s.secure_mode = true,
    s.access_control = true,
    s.encryption = "AES-256",
    s.audit_logging = true,
    s.auth_required = true,
    s.max_connections = 1000,
    s.session_timeout = "300s"
```

2. Implement authentication and authorization

```
// Create user nodes and roles
CREATE (:User {
  id: "admin",
  username: "network_admin",
  password_hash: "$2b$10$EXAMPLEHASH",
  role: "administrator",
  permissions: ["read", "write", "configure", "monitor"]
})

CREATE (:User {
  id: "operator",
  username: "network_operator",
  password_hash: "$2b$10$EXAMPLEHASH2",
  role: "operator",
  permissions: ["read", "monitor"]
})
```

```
CREATE (:Role {
  name: "administrator",
  privileges: ["full_access"]
})

CREATE (:Role {
  name: "operator",
  privileges: ["read_only", "monitor"]
})

// Connect users to switches with access policies

MATCH (u:User {id: "admin"}), (s:Switch)
CREATE (u)-[:HAS_ACCESS_TO {
  permission_level: "admin",
  ssh_key: "ssh-rsa AAAAB3NzaC1yc2E...",
  last_access: datetime()
}]->(s)

MATCH (u:User {id: "operator"}), (s:Switch)
CREATE (u)-[:HAS_ACCESS_TO {
  permission_level: "operator",
  ssh_key: "ssh-rsa AAAAB3NzaC1yc2E...",
  last_access: datetime()
}]->(s)
```

3. Implement network segmentation (VLANs)

```
// Create VLANs for segmentation
CREATE (:VLAN {
  id: "vlan10",
  name: "Management",
```

```
  subnet: "10.0.10.0/24",
  purpose: "Network management traffic",
  isolation_level: "high"
})

CREATE (:VLAN {
  id: "vlan20",
  name: "User_Data",
  subnet: "10.0.20.0/24",
  purpose: "Regular user traffic",
  isolation_level: "medium"
})

CREATE (:VLAN {
  id: "vlan30",
  name: "DMZ",
  subnet: "10.0.30.0/24",
  purpose: "Public services",
  isolation_level: "low"
})

// Associate interfaces with VLANs

MATCH (s:Switch)-[i:INTERFACE]->(h:Host)
SET i.vlan = "vlan20",
    i.acl_policy = "default_restrictive"

// Management interface

MATCH (s:Switch {id: "s1"})
CREATE (s)-[:INTERFACE {
  port: 99,
  name: "s1-mgmt0",
```

```
  mac: "00:AA:BB:CC:DD:EE",
  vlan: "vlan10",
  ip: "10.0.10.1/24",
  purpose: "management"
}]->(:ManagementServer {id: "mgmt01"})
```

4. Configuring ACL policies (Access Control Lists)

```
// Create ACL policies
CREATE (:ACL {
  name: "restrictive_policy",
  description: "Default restrictive policy",
  rules: [
    "deny all except established",
    "allow tcp 22 from mgmt_network",
    "allow icmp from internal",
    "deny any any"
  ]
})

CREATE (:ACL {
  name: "dmz_policy",
  description: "DMZ policy with specific rules",
  rules: [
    "allow tcp 80,443 from any",
    "allow tcp 25 from internal",
    "deny any any"
  ]
})

// Apply ACL policies to switches
```

```
MATCH (acl:ACL {name: "restrictive_policy"}), (s:Switch)
CREATE (s)-[:HAS_ACL {applied: true}]->(acl)

// Apply specific policy for DMZ

MATCH (acl:ACL {name: "dmz_policy"}), (v:VLAN {id: "vlan30"})
CREATE (v)-[:USES_ACL]->(acl)
```

5. Implement security monitoring

```
// Create a monitoring and logging system

CREATE (:SecurityMonitor {
  id: "secmon01",
  type: "SIEM",
  ip: "10.0.10.100/24",
  log_retention: "90 days"
})

CREATE (:IDS {
  id: "ids01",
  type: "suricata",
  ip: "10.0.10.101/24",
  ruleset: "emerging-threats"
})

// Connect monitoring to switches

MATCH (mon:SecurityMonitor), (ids:IDS), (s:Switch)
CREATE (s)-[:SENDS_LOGS_TO]->(mon),
       (s)-[:MONITORED_BY]->(ids),
       (ids)-[:REPORTS_TO]->(mon)
```

6. Configure encryption for links

```
MATCH ()-[l:LINKS_TO]-()
SET l.encryption = "IPsec",
    l.encryption_level = "AES-256-GCM",
    l.integrity_check = true,
    l.psk_hash = "$2b$10$ENCRYPTEDPSKHASH"
```

7. Implement an intrusion detection system

```
// Configure anomaly detection
MATCH (s:Switch)
SET s.anomaly_detection = true,
    s.behavioral_baseline = true,
    s.threat_intelligence = true

// Create detection rules
CREATE (:DetectionRule {
  id: "rule001",
  name: "Port Scan Detection",
  severity: "high",
  condition: "multiple_connection_attempts > 10 within 1m",
  action: "alert_and_block"
})

CREATE (:DetectionRule {
  id: "rule002",
  name: "Unauthorized Access Attempt",
  severity: "critical",
  condition: "auth_failures > 3 within 5m",
  action: "block_and_alert"
})
```

```
// Connect rules to switches
MATCH (r:DetectionRule), (s:Switch)
CREATE (s)-[:HAS_DETECTION_RULE]->(r
```

8. Check security settings

```
// Query to check security status
MATCH (s:Switch)
RETURN s.id as Switch,
       s.secure_mode as SecureMode,
       s.encryption as Encryption,
       s.auth_required as AuthRequired,
       [(s)-[:HAS_ACL]->(a) | a.name] as ACLs
```

Switch	SecureMode	Encryption	AuthRequired	ACLs
"s1"	true	"AES-256"	true	["restrictive_policy"]
"s2"	true	"AES-256"	true	["restrictive_policy"]
"s3"	true	"AES-256"	true	["restrictive_policy"]

```
// Query to check access policies
MATCH (u:User)-[a:HAS_ACCESS_TO]->(s:Switch)
RETURN u.username as User, s.id as Switch, a.permission_level as AccessLevel
```

User	Switch	AccessLevel
"network_operator"	"s1"	"operator"
"network_admin"	"s1"	"admin"
"network_operator"	"s2"	"operator"
"network_admin"	"s2"	"admin"
"network_operator"	"s3"	"operator"
"network_admin"	"s3"	"admin"

9. Security audit consultation

```
MATCH (n)
WHERE n:Switch OR n:VLAN OR n:ACL
RETURN labels(n)[0] as NodeType,
       n.id as NodeID,
       keys(n) as Properties
ORDER BY NodeType, NodeID
```

NodeType	NodeID	Properties
"ACL"	null	["description", "name", "rules"]
"ACL"	null	["description", "name", "rules"]
"Switch"	"s1"	["threat_intelligence", "encryption", "anomaly_detection", "behavioral_baseline", "audit_logging", "max_connections", "session_timeout", "auth_required", "redundancy_mode", "secure_mode", "access_control", "health_check", "health_check_interval", "failover_time", "json_path", "model", "id", "grpc_address", "device_id"]
"Switch"	"s2"	["threat_intelligence", "encryption", "anomaly_detection", "behavioral_baseline", "audit_logging", "max_connections", "session_timeout", "auth_required", "redundancy_mode", "secure_mode", "access_control", "health_check", "health_check_interval", "failover_time", "json_path", "model", "id", "grpc_address", "device_id"]
"Switch"	"s3"	["anomaly_detection", "behavioral_baseline", "threat_intelligence", "session_timeout", "auth_required", "encryption", "secure_mode", "access_control", "audit_logging", "max_connections", "failover_time", "redundancy_mode", "json_path", "health_check", "health_check_interval", "grpc_address", "id", "role", "device_id", "model"]
"VLAN"	"vlan10"	["id", "subnet", "purpose", "isolation_level", "name"]
"VLAN"	"vlan20"	["id", "subnet", "purpose", "isolation_level", "name"]
"VLAN"	"vlan30"	["subnet", "purpose", "isolation_level", "name", "id"]

Conclusion

The implementation of the following commands aims to meet the proposed objective, a comprehensive security infrastructure for your network, including:

- Robust authentication and authorization
- Network segmentation via VLANs
- Access control policies (ACLs)
- Encryption of data in transit
- Threat monitoring and detection
- Logging and auditing system

Run these commands in Neo4j to significantly strengthen the security of your network.