Java Implementation of LDMC Algorithm for ONOS Multi-Controller

```
// LDMC Algorithm Implementation for ONOS Controller (Java)

import org.onosproject.app.ApplicationException;
import org.onosproject.app.ApplicationService;
import org.onosproject.cfg.ComponentConfigService;
import org.onosproject.core.ApplicationId;
import org.onosproject.core.CoreService;
import org.onosproject.net.config.NetworkConfigRegistry;
import org.onosproject.net.config.basics.SubjectFactories;
import org.osgi.service.component.ComponentContext;
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Deactivate;
import org.osgi.service.component.annotations.Reference;
import org.osgi.service.component.annotations.ReferenceCardinality;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.*;
import java.net.*;
import java.security.KeyStore;
import javax.net.ssl.*;
import java.security.SecureRandom;
import java.util.*;
import java.util.concurrent.*;
import com.google.gson.Gson;

/**
 * LDMC Application for ONOS Controller.
 */
@Component(immediate = true)
public class LDMC {

    private final Logger log = LoggerFactory.getLogger(getClass());

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected CoreService coreService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected ApplicationService applicationService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected ComponentConfigService cfgService;

    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    protected NetworkConfigRegistry cfgRegistry;

    // Configuration (Replace with your actual values)
    private static final String LC_ADDRESS = "127.0.0.1";
    private static final int LC_PORT = 8081;
    private static final String[] PC_ADDRESSES = {"127.0.0.1:8082", "127.0.0.1:8083"};
    private static final String CERT_FILE = "path/to/your/certificate.jks";
    private static final String KEYSTORE_PASSWORD = "your_keystore_password";
    private static final String TRUSTSTORE_FILE = "path/to/your/truststore.jks";
    private static final String TRUSTSTORE_PASSWORD = "your_truststore_password";
    private static final int MONITORING_INTERVAL = 1000; // Milliseconds
    private static final int GLOBAL_MITIGATION_SEVERITY = 3;
    private static final int REGIONAL_MITIGATION_SEVERITY = 2;
```

```
    private static final Gson gson = new Gson();

    // Data Structures
    static class Alert {
        String srcCtrlId;
        String tgtNetSeg;
        int atkSev;
        long ts;
        List<String> extFeats;

        public String toJson() {
            return gson.toJson(this);
        }

        public static Alert fromJson(String json) {
            return gson.fromJson(json, Alert.class);
        }
    }

    // Security Context Setup
    private static SSLContext createSSLContext() throws Exception {
        KeyStore keyStore = KeyStore.getInstance("JKS");
        try (FileInputStream fis = new FileInputStream(CERT_FILE)) {
            keyStore.load(fis, KEYSTORE_PASSWORD.toCharArray());
        }

        KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
        keyManagerFactory.init(keyStore, KEYSTORE_PASSWORD.toCharArray());

        KeyStore trustStore = KeyStore.getInstance("JKS");
        try (FileInputStream fis = new FileInputStream(TRUSTSTORE_FILE)) {
            trustStore.load(fis, TRUSTSTORE_PASSWORD.toCharArray());
        }

        TrustManagerFactory trustManagerFactory =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
        trustManagerFactory.init(trustStore);

        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(keyManagerFactory.getKeyManagers(), trustManagerFactory.getTrustManagers(), new
SecureRandom());
        return sslContext;
    }

    // Communication Functions
    private static void sendSecure(SSLSocket socket, String message) throws IOException {
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        out.println(message);
    }

    private static String receiveSecure(SSLSocket socket) throws IOException {
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        return in.readLine();
    }

    private static boolean anomalyDetected(String switchId) {
        return false;
    }

    private static int detectSeverity(Object anomaly) {
        return 1;
```

```
    }

    private static List<String> extractFeatures(String switchId) {
        return new ArrayList<>();
    }

    private static List<String> generateGlobalMitigationRules(Alert alert) {
        return Arrays.asList("global_rule");
    }

    private static List<String> generateRegionalMitigationRules(Alert alert) {
        return Arrays.asList("regional_rule");
    }

    private static List<String> generateLocalMitigationRules(Alert alert) {
        return Arrays.asList("local_rule");
    }

    private static List<String> findControllers(String targetSegment) {
        return Arrays.asList("127.0.0.1:8082");
    }

    private static String findController(String controllerId) {
        return "127.0.0.1:8082";
    }

    private static void applyMitigation(String controllerAddress, List<String> rules) {
        log.info("Applying mitigation rules " + rules + " to " + controllerAddress);
    }

    private static void updateGlobalView(String data) {
        log.info("Updating global view with data: " + data);
    }

    // Physical Controller (PC) Thread
    private static class PCThread implements Runnable {
        private final String pcAddress;
        private final List<String> managedSwitches;
        private final SSLContext sslContext;

        public PCThread(String pcAddress, List<String> managedSwitches, SSLContext sslContext) {
            this.pcAddress = pcAddress;
            this.managedSwitches = managedSwitches;
            this.sslContext = sslContext;
        }

        @Override
        public void run() {
            try {
                String pcId = UUID.randomUUID().toString();
                String[] parts = pcAddress.split(":");
                String host = parts[0];
                int port = Integer.parseInt(parts[1]);

                SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();
                try (SSLSocket socket = (SSLSocket) sslSocketFactory.createSocket(LC_ADDRESS, LC_PORT))
{
                    for (String switchId : managedSwitches) {
                        if (anomalyDetected(switchId)) {
                            int severity = detectSeverity(null); //replace null with anomaly
                            List<String> features = extractFeatures(switchId);
                            Alert alert = new Alert();
```

```
                            alert.srcCtrlId = pcId;
                            alert.tgtNetSeg = "network_segment";
                            alert.atkSev = severity;
                            alert.ts = System.currentTimeMillis();
                            alert.extFeats = features;
                            sendSecure(socket, alert.toJson());
                            sendSecure(socket, "local_data"); // Simulate sending local data
                            Thread.sleep(MONITORING_INTERVAL);
                        }
                    }
                }
            } catch (Exception e) {
                log.error("PC thread error: {}", e.getMessage());
                e.printStackTrace();
            }
        }
    }

    // Logical Controller (LC) Thread
    private static class LCThread implements Runnable {
        private final SSLContext sslContext;
        private final BlockingQueue<Alert> alertQueue = new LinkedBlockingQueue<>();
        private final BlockingQueue<String> dataQueue = new LinkedBlockingQueue<>();

        public LCThread(SSLContext sslContext) {
           this.sslContext = sslContext;
    }

    @Override
    public void run() {
        try {
            SSLServerSocketFactory sslServerSocketFactory = sslContext.getServerSocketFactory();
            try (SSLServerSocket serverSocket = (SSLServerSocket)
sslServerSocketFactory.createServerSocket(LC_PORT)) {
                while (true) {
                    SSLSocket clientSocket = (SSLSocket) serverSocket.accept();
                    new Thread(() -> handleClient(clientSocket)).start();
                }
            }
        } catch (Exception e) {
            log.error("LC thread error: {}", e.getMessage());
            e.printStackTrace();
        }
    }

    private void handleClient(SSLSocket clientSocket) {
        try {
            while (true) {
                String data = receiveSecure(clientSocket);
                if (data != null) {
                    try {
                        Alert alert = Alert.fromJson(data);
                        alertQueue.put(alert);
                    } catch (Exception e) {
                        dataQueue.put(data);
                    }
                } else {
                    break;
                }
            }
        } catch (IOException e) {
            log.error("LC client handler error: {}", e.getMessage());
```

```
            e.printStackTrace();
        }
    }

    private void processAlerts() {
        while (true) {
            try {
                Alert alert = alertQueue.take();
                if (alert.atkSev == GLOBAL_MITIGATION_SEVERITY) {
                    List<String> rules = generateGlobalMitigationRules(alert);
                    List<String> controllers = findControllers("Net");
                    for (String controller : controllers) {
                        applyMitigation(controller, rules);
                    }
                } else if (alert.atkSev == REGIONAL_MITIGATION_SEVERITY) {
                    List<String> rules = generateRegionalMitigationRules(alert);
                    List<String> controllers = findControllers(alert.tgtNetSeg);
                    for (String controller : controllers) {
                        applyMitigation(controller, rules);
                    }
                } else {
                    String controllerAddress = findController(alert.srcCtrlId);
                    List<String> rules = generateLocalMitigationRules(alert);
                    applyMitigation(controllerAddress, rules);
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }

    private void processData() {
        while (true) {
            try {
                String data = dataQueue.take();
                updateGlobalView(data);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }

    public void startProcessing() {
        new Thread(this::processAlerts).start();
        new Thread(this::processData).start();
    }
}

// ONOS Component Lifecycle Methods
@Activate
protected void activate(ComponentContext context) {
    ApplicationId appId = coreService.registerApplication("org.example.ldmc");
    try {
        SSLContext sslContext = createSSLContext();
        LCThread lcThread = new LCThread(sslContext);
        new Thread(lcThread).start();
        lcThread.startProcessing();

        for (String pcAddress : PC_ADDRESSES) {
            List<String> managedSwitches = Arrays.asList("switch1", "switch2"); // Replace with actual
switches
            PCThread pcThread = new PCThread(pcAddress, managedSwitches, sslContext);
```

```
            new Thread(pcThread).start();
        }

        log.info("Started LDMC Application");
    } catch (Exception e) {
        log.error("Error activating LDMC application", e);
        applicationService.unregisterApplication(appId);
        throw new ApplicationException("Error activating LDMC application", e);
    }

    // Register network configuration
    cfgRegistry.registerConfigFactory(SubjectFactories.APP_SUBJECT_FACTORY,
            LDMCConfig.class);
}

@Deactivate
protected void deactivate() {
    // Unregister network configuration
    cfgRegistry.unregisterConfigFactory(SubjectFactories.APP_SUBJECT_FACTORY,
            LDMCConfig.class);
    log.info("Stopped LDMC Application");
}

// Example Network Configuration Class (Optional)
public static class LDMCConfig extends org.onosproject.net.config.Config<ApplicationId> {
    public static final String LC_ADDRESS = "lcAddress";
    public static final String LC_PORT = "lcPort";

    public LDMCConfig() {
        super();
    }

    public LDMCConfig(ApplicationId subject) {
        super(subject);
    }

    public String lcAddress() {
        return get(LC_ADDRESS, "127.0.0.1");
    }

    public int lcPort() {
        return get(LC_PORT, 8081);
    }
}
}
```

Key Adaptations for ONOS:

1. **ONOS Component:**
   - The LDMC class is annotated with @Component(immediate = true) to indicate it's an ONOS component.
2. **References:**
   - The coreService, applicationService, cfgService, and cfgRegistry are injected using @Reference annotations.
3. **ONOS Component Lifecycle Methods:**
   - @Activate is used for initialization when the component is activated.
   - @Deactivate is used for cleanup when the component is deactivated.
4. **Application Registration:**

- In the activate() method, the application is registered using coreService.registerApplication().

5. **Network Configuration (Optional):**
   - The code includes an example LDMCConfig class that demonstrates how to use ONOS network configuration.
   - This allows you to configure the LDMC application through ONOS configuration mechanisms.
6. **Logging:**
   - The slf4j logger is used for logging messages.
7. **ONOS Integration:**
   - You'll need to integrate the LDMC logic with ONOS's event handling and flow rule management mechanisms.
   - This involves using ONOS's APIs to listen for network events, retrieve device information, and install flow rules.

Deployment Instructions (ONOS Controller):

1. **Prerequisites:**
   - Java Development Kit (JDK): Ensure you have a compatible JDK installed.
   - ONOS Controller: Download and install the ONOS controller.
   - SSL Certificates: Generate or obtain SSL certificates for secure communication.
   - Gson Library: Add the Gson library to your ONOS project.
2. **Code Integration:**
   - Create a new Java class (e.g., LDMC.java) in your ONOS application's src/main/java directory.
   - Copy the provided code into it.
   - Update the configuration parameters with your environment's settings.
   - Place your certificate and truststore files in the specified paths.
3. **Build and Deploy:**
   - Build the ONOS application using Maven (e.g., mvn clean install).
   - Deploy the application to ONOS using the ONOS CLI (e.g., onos-app install target/ldmc-app.oar).
4. **Mininet-WiFi Deployment (Emulation):**
   - Mininet-WiFi Setup: Set up a Mininet-WiFi topology with P4-enabled switches.
   - ONOS Integration: Configure the Mininet-WiFi switches to connect to the ONOS controller.
   - LDMC Deployment: The LDMC application will start automatically when deployed to ONOS.
   - Testing: Simulate SYN flood attacks in Mininet-WiFi to test the LDMC framework's detection and mitigation capabilities.
5. **Real-World Deployment:**
   - Hardware Setup: Deploy P4-enabled switches and servers in your physical network.
   - ONOS Installation: Install the ONOS controller on a server.
   - LDMC Deployment: Deploy the LDMC application to ONOS.
   - Network Configuration: Configure the P4 switches to connect to the ONOS controller.
   - Monitoring and Testing: Monitor the network for SYN flood attacks and test the LDMC framework's effectiveness.