Java Implementation of MLDM Algorithm

```
// MLDM Algorithm Implementation (Java)

import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public class MLDM {

    private final long monWinDur;
    private final long mitigationInterval;
    private final Queue<Map<String, Object>> alertQueue = new LinkedList<>();
    private final Map<String, Object> mitigationRules = new ConcurrentHashMap<>();
    private final Map<String, Object> currentNetworkState = new ConcurrentHashMap<>();

    public MLDM(long monWinDur, long mitigationInterval) {
        this.monWinDur = monWinDur;
        this.mitigationInterval = mitigationInterval;
    }

    public void applyMitigationRules(String switchId, Map<String, Object> rules) {
        System.out.println("Applying rules " + rules + " to switch " + switchId);
    }

    public void distributeMitigationRules(String targetSegment, Map<String, Object> rules) {
        System.out.println("Distributing rules " + rules + " to segment " + targetSegment);
    }

    public String determineMitigationLevel(Map<String, Object> alert) {
        int atkSev = (int) alert.get("atkSev");
        if (atkSev == 3) {
            return "Global";
        } else if (atkSev == 2) {
            return "Regional";
        } else {
            return "Local";
        }
    }

    public Map<String, Object> generateMitigationRules(Map<String, Object> alert, String
mitigationLevel, Map<String, Object> networkState) {
        Map<String, Object> rules = new HashMap<>();
        rules.put("rule", "Mitigation rule for " + mitigationLevel);
        return rules;
    }

    public String getTargetPC(Map<String, Object> alert) {
        // Implement logic to get target PC based on alert
        return "PC1";
    }

    public String getTargetRegion(Map<String, Object> alert) {
        // Implement logic to get target region based on alert
        return "Region1";
    }

    public boolean hasAlert() {
        return !alertQueue.isEmpty();
```

```
    }

    public Map<String, Object> getAlert() {
        return alertQueue.poll();
    }

    public Map<String, Object> getCurrentNetworkState() {
        // Implement logic to get current network state
        Map<String, Object> state = new HashMap<>();
        state.put("state", "Network state");
        return state;
    }

    public Map<String, Object> adjustMitigationRules(Map<String, Object> rules, Map<String, Object>
networkState) {
        Map<String, Object> adjustedRules = new HashMap<>();
        adjustedRules.put("adjusted_rule", "Adjusted rule");
        return adjustedRules;
    }

    public void sleep(long duration) {
        try {
            Thread.sleep(duration);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    public void mitigationProcess() {
        ScheduledExecutorService scheduler = Executors.newScheduledThreadPool(1);
        scheduler.scheduleAtFixedRate(() -> {
            if (hasAlert()) {
                Map<String, Object> alert = getAlert();
                String mitigationLevel = determineMitigationLevel(alert);
                Map<String, Object> rules = generateMitigationRules(alert, mitigationLevel,
currentNetworkState);

                if (mitigationLevel.equals("Local")) {
                    String targetPC = getTargetPC(alert);
                    String[] switches = {"Switch_1", "Switch_2", "Switch_3"}; // Placeholder switches
                    for (String switchId : switches) {
                        applyMitigationRules(switchId, rules);
                    }
                } else if (mitigationLevel.equals("Regional")) {
                    String targetRegion = getTargetRegion(alert);
                    distributeMitigationRules(targetRegion, rules);
                } else if (mitigationLevel.equals("Global")) {
                    distributeMitigationRules("Net", rules);
                }
            }

            if (System.currentTimeMillis() % monWinDur == 0) {
                currentNetworkState.putAll(getCurrentNetworkState());
                if (!mitigationRules.isEmpty()) {
                    mitigationRules.putAll(adjustMitigationRules(mitigationRules, currentNetworkState));
                    distributeMitigationRules("Net", mitigationRules);
                }
                sleep(mitigationInterval);
            }
        }, 0, 100, TimeUnit.MILLISECONDS); //check every 100ms
    }
```

```
    public void start() {
        mitigationProcess();
    }

    public static void main(String[] args) {
        MLDM mldm = new MLDM(5000, 1000); // Example: 5s window, 1s interval
        mldm.start();

        // Placeholder for alert injection (Replace with your controller integration)
        ScheduledExecutorService alertScheduler = Executors.newScheduledThreadPool(1);
        alertScheduler.scheduleAtFixedRate(() -> {
            Map<String, Object> alert = new HashMap<>();
            alert.put("atkSev", new Random().nextInt(3) + 1); // Random severity 1, 2, or 3
            mldm.alertQueue.add(alert);
        }, 0, 2, TimeUnit.SECONDS); // inject an alert every 2 seconds.

        // Keep the main thread alive (or use a proper controller integration)
        try {
            Thread.currentThread().join();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

Explanation of the Code:

1. **Imports:**
   - Imports necessary Java utility classes for collections and concurrency.
2. **MLDM Class:**
   - Initializes monWinDur, mitigationInterval, alertQueue, mitigationRules, and currentNetworkState.
3. **Methods:**
   - applyMitigationRules(): Applies mitigation rules to a switch.
   - distributeMitigationRules(): Distributes mitigation rules using LDMC.
   - determineMitigationLevel(): Determines mitigation level based on alert severity.
   - generateMitigationRules(): Generates mitigation rules based on alert and network state.
   - getTargetPC(): Gets target Physical Controller.
   - getTargetRegion(): Gets target region.
   - hasAlert(): Checks for alerts.
   - getAlert(): Retrieves an alert.
   - getCurrentNetworkState(): Retrieves current network state.
   - adjustMitigationRules(): Adjusts mitigation rules based on network state.
   - sleep(): Pauses execution.
   - mitigationProcess(): Main mitigation loop, scheduled to run every 100ms.
   - start(): Starts the mitigation process.
4. **mitigationProcess():**
   - Continuously checks for alerts and applies mitigation rules.
   - Periodically adjusts mitigation rules based on network state.
5. **Example Usage (main()):**
   - Creates an MLDM instance.

- Starts the mitigation process.
- Defines an alert injection scheduler to simulate alert injection.
- Keeps the main thread alive.

Deployment Instructions:

1. **Controller Integration:**
   - Integrate the MLDM class into your Java-based controller (Beacon, Floodlight, OpenDaylight, ONOS).
   - Implement the LDMC distribution logic using the controller's APIs.
   - Implement the alert mechanism to receive alerts from DPSAD and AMCE.
2. **Alert Source Integration:**
   - Connect the MLDM module to the AMCE and DPSAD modules, so the MLDM module can receive alerts.
3. **Network State Integration:**
   - Implement the network state retrieval logic using the controller's APIs.
4. **Rule Application:**
   - Implement the rule application logic using the controller's APIs to configure P4 switches.
5. **Controller Startup:**
   - Start your controller.
   - The MLDM application will start its mitigation process.
6. **Testing:**
   - Generate SYN flood traffic in your SD-IoT network.
   - Monitor the controller for mitigation actions.
   - Verify the effectiveness of the mitigation rules.
7. **Error Handling:**
   - Add error handling to your code.
8. **Performance Tuning:**
   - Tune the MLDM algorithm parameters (e.g., mitigation interval, thresholds) to optimize performance.
9. **Controller Communication:**
   - Implement the necessary communication with the controller.
10. **LDMC Integration:**
    - Ensure proper integration with the LDMC architecture from Module 2.