Python Implementation of LDMC Algorithm for Ryu Multi-Controller

```
// LDMC Algorithm Implementation for Ryu Controller (Python)

# LDMC Algorithm Implementation for Ryu Controller (Python)

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
import threading
import time
import socket
import ssl
import json
import logging
import uuid

# Configuration (Replace with your actual values)
LC_ADDRESS = ('127.0.0.1', 8081)
PC_ADDRESSES = [('127.0.0.1', 8082), ('127.0.0.1', 8083)]
CERT_FILE = 'path/to/your/certificate.pem'
KEY_FILE = 'path/to/your/private_key.pem'
CA_CERT_FILE = 'path/to/ca_certificate.pem'
MONITORING_INTERVAL = 1  # Seconds
GLOBAL_MITIGATION_SEVERITY = 3
REGIONAL_MITIGATION_SEVERITY = 2

# Logging Setup
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# Data Structures
class Alert:
    def __init__(self, src_ctrl_id, tgt_net_seg, atk_sev, ts, ext_feats):
        self.src_ctrl_id = src_ctrl_id
        self.tgt_net_seg = tgt_net_seg
        self.atk_sev = atk_sev
        self.ts = ts
        self.ext_feats = ext_feats

    def to_json(self):
        return json.dumps(self.__dict__)

    @classmethod
    def from_json(cls, json_str):
        data = json.loads(json_str)
        return cls(**data)

# Security Context Setup
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
context.load_cert_chain(certfile=CERT_FILE, keyfile=KEY_FILE)
context.load_verify_locations(cafile=CA_CERT_FILE)
context.verify_mode = ssl.CERT_REQUIRED

# Communication Functions
def send_secure(sock, message):
    try:
        sock.sendall(message.encode())
    except Exception as e:
```

```python
        logging.error(f"Error sending message: {e}")

def receive_secure(sock):
    try:
        data = sock.recv(4096).decode()
        return data
    except Exception as e:
        logging.error(f"Error receiving message: {e}")
        return None

# Placeholder Functions (Replace with actual implementations)
def anomaly_detected(switch):
    # Simulate anomaly detection logic
    return False

def detect_severity(anomaly):
    # Simulate severity detection logic
    return 1

def extract_features(switch):
    # Simulate feature extraction logic
    return []

def generate_global_mitigation_rules(alert):
    # Simulate global mitigation rule generation logic
    return ["global_rule"]

def generate_regional_mitigation_rules(alert):
    # Simulate regional mitigation rule generation logic
    return ["regional_rule"]

def generate_local_mitigation_rules(alert):
    # Simulate local mitigation rule generation logic
    return ["local_rule"]

def find_controllers(target_segment):
    # Simulate finding controllers for a segment
    return [('127.0.0.1', 8082)]

def find_controller(controller_id):
    # Simulate finding a controller by ID
    return ('127.0.0.1', 8082)

def apply_mitigation(controller_address, rules):
    # Simulate applying mitigation rules
    logging.info(f"Applying mitigation rules {rules} to {controller_address}")

def update_global_view(data):
    # Simulate updating global network view
    logging.info(f"Updating global view with data: {data}")

# Physical Controller (PC) Thread
def pc_thread(pc_address, managed_switches):
    pc_id = str(uuid.uuid4())
    try:
        with socket.create_connection(LC_ADDRESS) as sock:
            with context.wrap_socket(sock, server_hostname=LC_ADDRESS[0]) as ssock:
                while True:
                    for switch in managed_switches:
                        if anomaly_detected(switch):
                            severity = detect_severity(None) # replace None with anomaly
                            features = extract_features(switch)
```

```python
                            alert = Alert(pc_id, "network_segment", severity, time.time(), features)
                            send_secure(ssock, alert.to_json())
                            send_secure(ssock, "local_data")  # Simulate sending local data
                            time.sleep(MONITORING_INTERVAL)
    except Exception as e:
        logging.error(f"PC thread error: {e}")

# Logical Controller (LC) Thread
def lc_thread():
    try:
        with socket.create_server(LC_ADDRESS) as server_socket:
            server_socket.listen(5)
            with context.wrap_socket(server_socket, server_side=True) as ssock:
                while True:
                    client_socket, addr = ssock.accept()
                    client_thread = threading.Thread(target=handle_client, args=(client_socket,))
                    client_thread.start()

    except Exception as e:
        logging.error(f"LC thread error: {e}")

def handle_client(client_socket):
    try:
        while True:
            data = receive_secure(client_socket)
            if data:
                try:
                    alert = Alert.from_json(data)
                    process_alert(alert)
                except json.JSONDecodeError:
                    update_global_view(data)
            else:
                break
    except Exception as e:
        logging.error(f"LC client handler error: {e}")

def process_alert(alert):
    if alert.atk_sev == GLOBAL_MITIGATION_SEVERITY:
        rules = generate_global_mitigation_rules(alert)
        controllers = find_controllers("Net")
        for controller in controllers:
            apply_mitigation(controller, rules)
    elif alert.atk_sev == REGIONAL_MITIGATION_SEVERITY:
        rules = generate_regional_mitigation_rules(alert)
        controllers = find_controllers(alert.tgt_net_seg)
        for controller in controllers:
            apply_mitigation(controller, rules)
    else:
        controller_address = find_controller(alert.src_ctrl_id)
        rules = generate_local_mitigation_rules(alert)
        apply_mitigation(controller_address, rules)

# Ryu Application
class LDMCApp(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(LDMCApp, self).__init__(*args, **kwargs)
        self.datapaths = {}
        threading.Thread(target=lc_thread).start()
        for pc_address in PC_ADDRESSES:
            managed_switches = ["switch1", "switch2"]  # Replace with actual switches
```

```
            threading.Thread(target=pc_thread, args=(pc_address, managed_switches)).start()

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        self.datapaths[datapath.id] = datapath

        # Install table-miss flow entry
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        pkt = packet.Packet(msg.data)
        # Add your packet processing logic here if needed
        # Example:
        # eth = pkt.get_protocol(ethernet.ethernet)
        # if eth:
        #     dst = eth.dst
        #     src = eth.src
        #     dpid = datapath.id
        #     self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # For now, just forward to the controller for table-miss
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER, ofproto.OFPCML_NO_BUFFER)]
        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data

        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id, in_port=in_port,
actions=actions, data=data)
       datapath.send_msg(out)
```

Explanation of the Ryu Code:

1. **Ryu Application:**
    - The LDMCApp class inherits from app_manager.RyuApp and represents the Ryu application.
2. **Initialization:**
    - The __init__() method initializes the datapaths dictionary and starts the LC and PC threads.
3. **Switch Features Handler:**

- The switch_features_handler() method is called when a switch connects.
- It stores the datapath object in the datapaths dictionary and installs a table-miss flow entry to send packets to the controller.

4. **Add Flow:**
    - The add_flow() method installs a flow entry on the switch.
5. **Packet-In Handler:**
    - The packet_in_handler() method is called when a switch sends a packet-in message.
    - It currently forwards all packets to the controller for table-miss handling.
    - You can add your packet processing logic here, such as analyzing packet headers and making forwarding decisions.
6. **Configuration, Data Structures, Security, Communication, and Placeholder Functions:**
    - These parts are identical to the Python code for other controllers.
7. **Physical Controller (PC) and Logical Controller (LC) Threads:**
    - These threads are also identical to the python code for other controllers.

Deployment Instructions (Ryu Controller):

1. **Prerequisites:**
    - Python: Ensure you have a compatible Python environment.
    - Ryu Controller: Install the Ryu controller.
    - SSL Certificates: Generate or obtain SSL certificates for secure communication.
    - Gson Library: Install the json python library.
2. **Code Integration:**
    - Create a new Python file (e.g., ldmc_ryu.py) in a directory where Ryu can find it.
    - 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. **Run Ryu with the LDMC Application:**
    - Start Ryu from your terminal, specifying the LDMC application: ryu-manager ldmc_ryu.py
4. **Mininet-WiFi Deployment (Emulation):**
    - Mininet-WiFi Setup: Set up a Mininet-WiFi topology with OpenFlow 1.3-enabled switches.
    - Ryu Integration: Configure the Mininet-WiFi switches to connect to the Ryu controller.
    - LDMC Deployment: Run Ryu with the ldmc_ryu.py application.
    - 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 OpenFlow 1.3-enabled switches and servers in your physical network.
    - Ryu Installation: Run the Ryu controller on a server.
    - LDMC Deployment: Run Ryu with the ldmc_ryu.py application.
    - Network Configuration: Configure the OpenFlow switches to connect to the Ryu controller.
    - Monitoring and Testing: Monitor the network for SYN flood attacks and test the LDMC framework's effectiveness.