FractiNet Firmware Prototype for Cisco UADP 3.0

```
# Prototype Firmware for Cisco UADP 3.0 with FractiNet Integration

# ============================================================

# Dynamic Fractal Layer (DFL)

# ============================================================

def fractal_route_optimize(source_ip, dest_ip, traffic_matrix):

    """

    Optimizes traffic routes dynamically using fractalized logic.

    :param source_ip: Source IP address

    :param dest_ip: Destination IP address

    :param traffic_matrix: Traffic load matrix

    :return: Optimal path

    """

    fractal_map = generate_fractal_map(traffic_matrix)

    optimal_path = find_optimal_path(fractal_map, source_ip, dest_ip)

    return optimal_path

def generate_fractal_map(traffic_matrix):

    """

    Generate a fractalized traffic map based on the current load.

    """

    harmonized_matrix = []

    for row in traffic_matrix:

        harmonized_row = [x / max(row) if max(row) > 0 else 0 for x in row]

        harmonized_matrix.append(harmonized_row)
    return harmonized_matrix

def find_optimal_path(fractal_map, source_ip, dest_ip):

    """

    Find the best path using fractal harmonization.

    """

    return fractal_map[source_ip][dest_ip]  # Simplified logic for illustration

def integrate_dfl_into_cisco():

    """

    Integrate DFL into Cisco UADP 3.0 Layer 3 Forwarding Pipeline.

    """

    cisco_pipeline.add_module("DFL", fractal_route_optimize)

    print("DFL integrated into Cisco UADP 3.0 Layer 3 Forwarding Pipeline.")

# ==================================================================

# Protocol Translation Layer (PTL)

# ==================================================================

def protocol_translate(packet, target_protocol):

    """

    Translate fractalized packets into legacy protocols.

    :param packet: Fractalized packet

    :param target_protocol: Target protocol (e.g., TCP/UDP)

    :return: Translated packet

    """

    if target_protocol == "TCP":

        return fractal_to_tcp(packet)
    elif target_protocol == "UDP":
        return fractal_to_udp(packet)
    else:
        raise ValueError(f"Unsupported protocol: {target_protocol}")
def fractal_to_tcp(packet):
    """
    Convert fractalized packet to TCP format.
    """
    return {
        "header": packet["fractal_header"],
        "payload": packet["data"]
    }
def fractal_to_udp(packet):
    """
    Convert fractalized packet to UDP format.
    """
    return {
        "header": packet["fractal_header"],
        "payload": packet["data"]
    }
def integrate_ptl_into_cisco():
    """
    Integrate PTL into Cisco's Control Plane for Layer 2/3 traffic.
    """
    cisco_pipeline.add_module("PTL", protocol_translate)
    print("PTL integrated into Cisco UADP 3.0 Control Plane.")
# ==================================================================
# Recursive Error Correction Engine (RECE)
# ==================================================================
def recursive_error_correction(packet, redundancy_level=3):
    """
    Correct packet errors using recursive redundancy.
    :param packet: Packet data
    :param redundancy_level: Number of correction iterations
    :return: Corrected packet
    """
    for _ in range(redundancy_level):
        if detect_errors(packet):
            packet = apply_correction(packet)
    return packet
def detect_errors(packet):
    """
    Detect errors in the packet using checksum.
    """
    return sum(packet["data"]) % 256 != packet["checksum"]
def apply_correction(packet):
    """
    Correct packet errors using fractalized redundancy.
    """
    packet["data"] = [x - 1 if x > 0 else x for x in packet["data"]]
    return packet
def integrate_rece_into_cisco():
    """
    Integrate RECE into Cisco UADP 3.0 Control Plane Microcode.
    """
    cisco_control_plane.add_module("RECE", recursive_error_correction)
    print("RECE integrated into Cisco UADP 3.0 Control Plane Microcode.")
# ============================================================
# Fractalized Power Management Module (FPMM)
# ============================================================
def adjust_power_mode(module_id, mode):
    """
    Adjust the power mode for specific modules.
    :param module_id: Module identifier
    :param mode: Desired power mode ("low-power" or "normal")
    """
    if mode == "low-power":
        set_low_power_mode(module_id)
    elif mode == "normal":
        set_normal_power_mode(module_id)
def set_low_power_mode(module_id):
    """
    Set module to low-power mode.
    """
    print(f"Module {module_id} switched to low-power mode.")
def set_normal_power_mode(module_id):
    """
    Restore module to normal mode.
    """
    print(f"Module {module_id} restored to normal mode.")
def integrate_fpmm_into_cisco():
    """
    Integrate FPMM into Cisco's Power Management System.
    """
    cisco_power_interface.add_module("FPMM", adjust_power_mode)
    print("FPMM integrated into Cisco UADP 3.0 Power Management System.")
# ============================================================
# Prototype Testing
# ============================================================
def test_fractinet_firmware():
    """
    Test the integrated FractiNet Firmware for Cisco UADP 3.0.
    """
    # Test Dynamic Fractal Layer
    traffic_matrix = [[0, 5, 10], [5, 0, 15], [10, 15, 0]]
    source_ip = 0
    dest_ip = 2

    print(f"Optimal Path: {fractal_route_optimize(source_ip, dest_ip, traffic_matrix)}")

    # Test Protocol Translation Layer

    packet = {"fractal_header": "FRACTAL", "data": [1, 2, 3]}

    print(f"Translated TCP Packet: {protocol_translate(packet, 'TCP')}")

    # Test Recursive Error Correction Engine

    packet_with_error = {"data": [10, 20, 30], "checksum": 5}

    corrected_packet = recursive_error_correction(packet_with_error)

    print(f"Corrected Packet: {corrected_packet}")

    # Test Fractalized Power Management Module

    adjust_power_mode("module_1", "low-power")

    adjust_power_mode("module_1", "normal")

# Run all tests

if __name__ == "__main__":

    test_fractinet_firmware()
```

Summary of Prototype Components

1. Dynamic Fractal Layer (DFL):

• Optimizes traffic routing and bandwidth allocation using fractalized harmonization.

• Integrated into Cisco's Layer 3 Forwarding Pipeline.

2. Protocol Translation Layer (PTL):

• Translates fractalized packet formats into legacy protocols (e.g., TCP/UDP).

• Embedded into Cisco's Control Plane.

3. Recursive Error Correction Engine (RECE):

• Detects and corrects errors in packets using recursive redundancy.

- Integrated into Cisco’s Control Plane Microcode.

4. Fractalized Power Management Module (FPMM):

- Dynamically manages power modes to reduce energy consumption.
- Controlled via Cisco’s Power Management System.

This prototype firmware demonstrates how to integrate fractalized networking capabilities into the Cisco UADP 3.0 platform, enhancing performance, scalability, and energy efficiency.