The following is the P4 code implementation for the **P4-enabled Distributed Packet Aggregation and Disaggregation System (P4-DPADS)** in SD-IoT network. This implementation follows **Algorithm 3** and integrates the key steps of the module.

**Full P4 Code Implementation for P4-DPADS Module**

|  |
| --- |
| **// Define the headers for Ethernet, IPv4, and UDP (assuming UDP traffic for IoT devices)**  **header ethernet\_t {**  **bit<48> dst\_addr;**  **bit<48> src\_addr;**  **bit<16> eth\_type;**  **}**  **header ipv4\_t {**  **bit<4> version;**  **bit<4> ihl;**  **bit<8> diffserv;**  **bit<16> total\_len;**  **bit<16> identification;**  **bit<3> flags;**  **bit<13> frag\_offset;**  **bit<8> ttl;**  **bit<8> protocol;**  **bit<16> hdr\_checksum;**  **bit<32> src\_addr;**  **bit<32> dst\_addr;**  **}**  **header udp\_t {**  **bit<16> src\_port;**  **bit<16> dst\_port;**  **bit<16> length;**  **bit<16> checksum;**  **}**  **struct headers {**  **ethernet\_t eth;**  **ipv4\_t ipv4;**  **udp\_t udp;**  **}**  **struct metadata {**  **// Metadata variables for packet processing**  **bit<32> pkt\_count;**  **bit<32> agg\_pkt; // Aggregated packet payload**  **bit<32> disagg\_pkt; // Disaggregated packet payload**  **bit<32> disagg\_enabled; // Disaggregation status**  **bit<32> pkt\_threshold; // Threshold for aggregation**  **}**  **// Registers for storing states**  **register<bit<32>>(pkt\_count\_reg, 1);**  **register<bit<32>>(agg\_pkt\_reg, 1);**  **register<bit<32>>(disagg\_pkt\_reg, 1);**  **register<bit<1>>(disagg\_enabled\_reg, 1);**  **register<bit<32>>(pkt\_threshold\_reg, 1); // Set this based on the aggregation threshold**  **// Standard control blocks for ingress and egress pipelines**  **control Ingress(headers hdr, metadata meta, inout standard\_metadata\_t std\_meta) {**  **apply {**  **// Packet Ingress: Parse Ethernet and IPv4 headers, then proceed to check for disaggregation or aggregation**  **if (hdr.ipv4.isValid()) {**  **// Check if disaggregation is enabled**  **if (disagg\_enabled\_reg.read(0) == 1) {**  **// Disaggregation phase**  **meta.pkt\_count = pkt\_count\_reg.read(0);**  **meta.disagg\_pkt = hdr.udp.length; // Assuming UDP payload as IoT packet content**  **// Forward disaggregated packets**  **disagg\_pkt\_reg.write(0, meta.disagg\_pkt);**  **pkt\_count\_reg.write(0, 0); // Reset packet count**  **disagg\_enabled\_reg.write(0, 0); // Disable disaggregation for next cycle**  **} else {**  **// Aggregation phase**  **meta.pkt\_count = pkt\_count\_reg.read(0) + 1;**  **agg\_pkt\_reg.write(0, hdr.udp.length); // Accumulate IoT payloads into agg\_pkt register**  **// Check if packet count reached threshold**  **if (meta.pkt\_count >= pkt\_threshold\_reg.read(0)) {**  **// Perform aggregation operations**  **disagg\_enabled\_reg.write(0, 1); // Enable disaggregation after aggregation**  **pkt\_count\_reg.write(0, 0); // Reset packet count after aggregation**  **} else {**  **pkt\_count\_reg.write(0, meta.pkt\_count); // Update packet count**  **}**  **}**  **} else {**  **// Drop packet if headers are invalid**  **std\_meta.drop = 1;**  **}**  **}**  **}**  **control Egress(headers hdr, metadata meta, inout standard\_metadata\_t std\_meta) {**  **apply {**  **// Packet Egress: Process and forward the aggregated or disaggregated packet**  **// Handle any response packets from server and forward to the next DPADS node**  **}**  **}**  **control VerifyChecksum(headers hdr, inout metadata meta) {**  **apply { }**  **}**  **control ComputeChecksum(headers hdr, inout metadata meta) {**  **apply { }**  **}**  **// Pipeline processing block**  **control MyIngressPipeline {**  **apply {**  **VerifyChecksum();**  **Ingress();**  **ComputeChecksum();**  **}**  **}**  **control MyEgressPipeline {**  **apply {**  **Egress();**  **}**  **}**  **// Main switch pipeline**  **pipeline main {**  **MyIngressPipeline();**  **MyEgressPipeline();**  **}** |

**Step-by-Step Deployment Instructions for Simulation and Real-World Scenarios**

**Simulation Environment:**

**Tools Required: Mininet (for creating SDN-enabled network topologies), BMv2 (P4 software switch), P4C (P4 compiler), and Python for control plane logic.**

1. **Step 1: Network Setup in Mininet**
   * Create a simple topology in Mininet with multiple switches representing DPADS nodes.
   * Use the following Mininet command to create a topology: *sudo mn --topo=tree,depth=3,fanout=2 --controller=remote --switch ovsk -*
2. **Step 2: Compile the P4 Program**
   * Use the P4C compiler to compile the P4-DPADS program into a JSON configuration file:
   * *p4c-bm2-ss p4\_dpads.p4 -o dpads.json*
3. **Step 3: Load BMv2 Switch**
   * Load the BMv2 switch with the compiled *dpads.json* configuration.
   * *simple\_switch --log-console --thrift-port 9090 dpads.json*
4. **Step 4: Connect Control Plane**
   * Write a Python script to communicate with the switch via Thrift API to push aggregation/disaggregation rules.
   * Example script:

|  |
| --- |
| **from p4runtime\_lib.switch import ShutdownAllSwitchConnections**  **from p4runtime\_lib.helper import P4InfoHelper**  **p4info\_helper = P4InfoHelper("dpads.p4info")**  **sw1 = p4info\_helper.get\_switch\_connection('dpads-sw1')**  **sw1.WriteTableEntry(<aggregation\_rules>)**  **sw1.WriteTableEntry(<disaggregation\_rules>)**  **ShutdownAllSwitchConnections()** |

1. **Step 5: Simulate Traffic**
   * Use *hping3 or iperf* to generate IoT traffic:
   * Example script: *hping3 -S -p 80 --flood --rand-source <target\_ip>*
2. **Step 6: Monitor Traffic**
   * Observe the packet aggregation and disaggregation using the *simple\_switch\_CLI* to query registers:
   * register\_read pkt\_count\_reg 0
   * register\_read agg\_pkt\_reg 0
   * register\_read disagg\_pkt\_reg 0

**Real-World Deployment:**

**Tools Required: P4-compatible hardware switch (e.g., Netronome, Barefoot Tofino), ONOS or ODL as an SDN controller, and physical IoT devices.**

1. **Step 1: Switch Configuration**
   * Deploy a P4-enabled hardware switch. Configure it with the P4-DPADS program compiled to target your hardware.
   * Example for Barefoot Tofino switch: *tofino-cli -p dpads\_program.p4*
2. **Step 2: Network Setup**
   * Set up the SDN topology, connecting multiple DPADS nodes (switches) in a hierarchical manner. Each switch should represent a layer in the aggregation-disaggregation hierarchy.
3. **Step 3: Controller Configuration**
   * Configure the SDN controller (ONOS or OpenDaylight) to interact with the DPADS nodes for routing traffic, controlling flows, and sending aggregation/disaggregation rules dynamically.
4. **Step 4: IoT Traffic Generation**
   * Connect IoT devices to the network. Send traffic through the DPADS nodes and monitor packet aggregation and disaggregation through controller logs and network analytics.
5. **Step 5: Monitoring and Traffic Analysis**
   * Use the integrated P4-STM module to gather traffic statistics on aggregated and disaggregated packets, and apply dynamic adjustments as needed.
6. **Step 6: Field Trial Validation**
   * Evaluate real-world performance, monitor thresholds, and verify proper packet aggregation and disaggregation under actual network load conditions.

This full implementation and deployment guide will allow both simulation testing and real-world trials of the P4-DPADS module for efficient IoT traffic management in SD-IoT networks.