The following is the P4 code implementation for the **P4-enabled Distributed Load Balancing System (P4-DLBS)**, which leverages an **Enhanced Weighted Round Robin (P4-EWRR)** algorithm for distributed load balancing across multiple P4 switches and controllers in an SD-IoT network. This implementation follows **Algorithm 2** and integrates the key steps of the module.

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

|  |
| --- |
| **/\***  **P4-DLBS Module: Distributed Load Balancing with Enhanced Weighted Round Robin (P4-EWRR)**  **\*/**  **/\* Define headers \*/**  **header ethernet\_t {**  **bit<48> dstAddr;**  **bit<48> srcAddr;**  **bit<16> etherType;**  **}**  **header ipv4\_t {**  **bit<4> version;**  **bit<4> ihl;**  **bit<8> diffserv;**  **bit<16> totalLen;**  **bit<16> identification;**  **bit<3> flags;**  **bit<13> fragOffset;**  **bit<8> ttl;**  **bit<8> protocol;**  **bit<16> hdrChecksum;**  **bit<32> srcAddr;**  **bit<32> dstAddr;**  **}**  **header tcp\_t {**  **bit<16> srcPort;**  **bit<16> dstPort;**  **bit<32> seqNo;**  **bit<32> ackNo;**  **bit<4> dataOffset;**  **bit<6> reserved;**  **bit<6> flags;**  **bit<16> window;**  **bit<16> checksum;**  **bit<16> urgentPtr;**  **}**  **/\* Metadata and global variables \*/**  **struct metadata\_t {**  **bit<32> server\_index;**  **bit<32> total\_weight;**  **bit<32> cum\_weight;**  **bit<32> rand\_value;**  **bit<32> remaining\_resources;**  **}**  **metadata\_t meta;**  **/\* Registers \*/**  **register<bit<32>>(1) Last\_Server; // Store the index of last serving server**  **register<bit<32>>(1) Selected\_Index; // Store the selected server index**  **register<bit<32>>(10) Server\_Res; // Store remaining resources for each server**  **register<bit<32>>(10) Server\_State; // Store health state of servers (0: down, 1: up)**  **/\* Parser \*/**  **parser MyParser(packet\_in pkt, out headers\_t hdr, inout metadata\_t meta) {**  **state start {**  **pkt.extract(hdr.ethernet);**  **transition select(hdr.ethernet.etherType) {**  **0x0800: parse\_ipv4;**  **default: accept;**  **}**  **}**    **state parse\_ipv4 {**  **pkt.extract(hdr.ipv4);**  **transition select(hdr.ipv4.protocol) {**  **6: parse\_tcp;**  **default: accept;**  **}**  **}**    **state parse\_tcp {**  **pkt.extract(hdr.tcp);**  **transition accept;**  **}**  **}**  **/\* Ingress Pipeline \*/**  **action round\_robin\_selection() {**  **bit<32> n = 10; // Assuming 10 servers**  **bit<32> server\_index = Last\_Server.read(0);**    **// Enhanced Weighted Round Robin (EWRR)**  **meta.total\_weight = 0;**  **for (bit<32> i = 0; i < n; i++) {**  **bit<32> weight = Server\_Res.read(i);**  **meta.total\_weight += weight;**  **}**    **// Random selection based on weight**  **meta.rand\_value = random(meta.total\_weight); // Generate a random value**  **meta.cum\_weight = 0;**  **for (bit<32> i = 0; i < n; i++) {**  **bit<32> cur\_weight = Server\_Res.read(i);**  **meta.cum\_weight += cur\_weight;**  **if (meta.rand\_value < meta.cum\_weight) {**  **server\_index = i;**  **break;**  **}**  **}**  **// Health check: ensure the selected server is up**  **if (Server\_State.read(server\_index) == 0) {**  **server\_index = (server\_index + 1) % n; // Round-robin to next server if down**  **}**  **// Update selected server index and last server**  **Selected\_Index.write(0, server\_index);**  **Last\_Server.write(0, server\_index);**  **}**  **action forward\_to\_server(bit<32> server\_index) {**  **// Modify packet headers to forward to the selected server**  **modify\_field(hdr.ethernet.dstAddr, server\_mac[server\_index]);**  **modify\_field(hdr.ipv4.dstAddr, server\_ip[server\_index]);**  **modify\_field(hdr.tcp.dstPort, server\_port[server\_index]);**    **// Send to the corresponding port**  **standard\_metadata.egress\_spec = server\_port[server\_index];**  **}**  **table load\_balancing\_table {**  **actions = {**  **round\_robin\_selection;**  **forward\_to\_server;**  **}**  **key = {**  **hdr.ipv4.srcAddr: ternary;**  **hdr.ipv4.dstAddr: ternary;**  **}**  **size = 1024;**  **default\_action = round\_robin\_selection;**  **}**  **control ingress {**  **apply(load\_balancing\_table);**  **}**  **/\* Egress Pipeline \*/**  **control egress {**  **// Optional egress processing (e.g., for server responses)**  **}**  **/\* Deparser \*/**  **control MyDeparser(packet\_out pkt, in headers\_t hdr) {**  **pkt.emit(hdr.ethernet);**  **pkt.emit(hdr.ipv4);**  **pkt.emit(hdr.tcp);**  **}**  **/\* Main Control Block \*/**  **control MyControl(**  **inout headers\_t hdr,**  **inout metadata\_t meta,**  **inout standard\_metadata\_t standard\_metadata**  **) {**  **apply {**  **ingress.apply();**  **egress.apply();**  **}**  **}**  **/\* Switch Program \*/**  **V1Switch(MyParser(), MyDeparser(), MyControl()) {**  **}** |

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

**Simulation Environment:**

1. **Set Up P4 Development Environment:**
   * Install the P4 development environment (e.g., BMv2 or Mininet) on your machine.
   * You can use the P4 Tutorial VM (available on GitHub) for a quick setup.
   * Install necessary tools: p4c (P4 compiler), Mininet, and runtime tools (simple\_switch).
2. **Write the P4 Program:**
   * Save the above P4 code in a file, e.g., *p4\_dlbs.p4*.
   * Compile the P4 code using the *p4c* compiler:
   * *p4c --target bmv2 --arch v1model --std p4-16 p4\_dlbs.p4 -o p4\_dlbs.json.*
3. **Set Up Mininet Topology:**
   * Create a Mininet topology with multiple switches and hosts to simulate the SD-IoT environment.
   * Use a script to create a topology with P4 switches and servers (IoT nodes):
   * *sudo mn --custom < >.py --topo mytopo --controller remote*
4. **Run the P4-DLBS Program on P4 Switches:**
   * Load the compiled P4-DLBS program (*p4\_dlbs.json*) onto each P4 switch in the network:
   * *sudo simple\_switch --log-console --thrift-port 9090 p4\_dlbs.json*
5. **Verify Load Balancing:**
   * Use traffic generation tools (*hping3, iperf*, or similar) to send traffic from IoT nodes to servers.
   * Monitor traffic and server selection across multiple switches to verify that the load balancing mechanism is distributing requests according to the P4-EWRR algorithm.

**Real-World Deployment:**

1. **Hardware Setup:**
   * Install P4-compatible switches (e.g., Tofino-based or Netronome) and SDN controllers in the real-world network environment.
   * Ensure you have access to the IoT devices, servers, and controllers.
2. **Configure SDN Controllers:**
   * Set up multiple SDN controllers (e.g., ONOS, ODL) to manage the P4 switches.
   * Enable communication between controllers and P4 switches for coordinated load balancing decisions.
3. **Deploy the P4-DLBS Program:**
   * Load the compiled P4 program (*p4\_dlbs.p4*) onto the P4 switches in the real network.
   * Use the management interfaces of the P4 switches to upload and activate the P4-DLBS module.
4. **Server Health Monitoring and Resource Updates:**
   * Implement server health checks via an external script or controller, updating the *Server\_State* register in the P4 switches.
   * Dynamically adjust server weights (*Server\_Res* register) based on the remaining resources in each IoT server.
5. **Monitor and Test:**
   * Use network monitoring tools to ensure that traffic is being balanced effectively across servers.
   * Test failure scenarios (e.g., a server going down) to confirm that the module correctly reroutes traffic.