**DYNEX Module: P4 Source Code**

|  |
| --- |
| **// Define headers**  **header ethernet\_t {**  **macAddr\_t dstAddr;**  **macAddr\_t 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;**  **ipv4Addr\_t srcAddr;**  **ipv4Addr\_t dstAddr;**  **}**  **header tcp\_t {**  **bit<16> srcPort;**  **bit<16> dstPort;**  **bit<32> seqNo;**  **bit<32> ackNo;**  **bit<4> dataOffset;**  **bit<4> reserved;**  **bit<8> flags;**  **bit<16> window;**  **bit<16> checksum;**  **bit<16> urgentPtr;**  **}**  **// Define metadata**  **struct metadata\_t {**  **bit<32> classifier\_score[6]; // Scores from 6 classifiers: RF, DT, kNN, GNB, SVM, BLR**  **bit<8> classifier\_confidence[6]; // Confidence scores for each classifier**  **bit<32> meta\_learner\_output; // Final output from meta-learner**  **bit<1> is\_attack; // Final decision**  **}**  **// Define parser**  **parser MyParser(packet\_in packet,**  **out headers\_t hdr,**  **inout metadata\_t meta,**  **inout standard\_metadata\_t standard\_meta) {**  **state start {**  **packet.extract(hdr.ethernet);**  **transition select(hdr.ethernet.etherType) {**  **0x0800: parse\_ipv4;**  **default: accept;**  **}**  **}**  **state parse\_ipv4 {**  **packet.extract(hdr.ipv4);**  **transition select(hdr.ipv4.protocol) {**  **6: parse\_tcp;**  **default: accept;**  **}**  **}**  **state parse\_tcp {**  **packet.extract(hdr.tcp);**  **transition accept;**  **}**  **}**  **// Define tables**  **table feature\_selection\_table {**  **actions = {**  **select\_relevant\_features;**  **\_nop;**  **}**  **size = 1024;**  **default\_action = \_nop();**  **}**  **table classifier\_prediction\_table {**  **actions = {**  **rf\_predict;**  **dt\_predict;**  **knn\_predict;**  **gnb\_predict;**  **svm\_predict;**  **blr\_predict;**  **\_nop;**  **}**  **size = 6; // One entry for each classifier**  **default\_action = \_nop();**  **}**  **table meta\_learner\_table {**  **actions = {**  **combine\_predictions;**  **\_nop;**  **}**  **size = 1;**  **default\_action = \_nop();**  **}**  **// Define actions**  **action select\_relevant\_features() {**  **// Placeholder for feature selection logic**  **// Example: Extract specific features from the packet and store them in metadata**  **}**  **action rf\_predict() {**  **// Placeholder for RF prediction logic**  **meta.classifier\_score[0] = <calculated\_score>;**  **meta.classifier\_confidence[0] = <calculated\_confidence>;**  **}**  **action dt\_predict() {**  **// Placeholder for DT prediction logic**  **meta.classifier\_score[1] = <calculated\_score>;**  **meta.classifier\_confidence[1] = <calculated\_confidence>;**  **}**  **action knn\_predict() {**  **// Placeholder for kNN prediction logic**  **meta.classifier\_score[2] = <calculated\_score>;**  **meta.classifier\_confidence[2] = <calculated\_confidence>;**  **}**  **action gnb\_predict() {**  **// Placeholder for GNB prediction logic**  **meta.classifier\_score[3] = <calculated\_score>;**  **meta.classifier\_confidence[3] = <calculated\_confidence>;**  **}**  **action svm\_predict() {**  **// Placeholder for SVM prediction logic**  **meta.classifier\_score[4] = <calculated\_score>;**  **meta.classifier\_confidence[4] = <calculated\_confidence>;**  **}**  **action blr\_predict() {**  **// Placeholder for BLR prediction logic**  **meta.classifier\_score[5] = <calculated\_score>;**  **meta.classifier\_confidence[5] = <calculated\_confidence>;**  **}**  **action combine\_predictions() {**  **// Combine the predictions using the meta-learner logic**  **bit<32> weighted\_sum = 0;**  **for (int i = 0; i < 6; i++) {**  **weighted\_sum += meta.classifier\_score[i] \* meta.classifier\_confidence[i];**  **}**  **meta.meta\_learner\_output = weighted\_sum / 6; // Example weighted average**  **}**  **action make\_final\_decision() {**  **if (meta.meta\_learner\_output >= <threshold>) {**  **meta.is\_attack = 1;**  **} else {**  **meta.is\_attack = 0;**  **}**  **}**  **// Define control logic**  **control MyIngress(inout headers\_t hdr,**  **inout metadata\_t meta,**  **inout standard\_metadata\_t standard\_meta) {**  **apply(feature\_selection\_table);**  **apply(classifier\_prediction\_table);**  **apply(meta\_learner\_table);**  **make\_final\_decision();**  **}**  **// Define deparser**  **control MyDeparser(packet\_out packet,**  **in headers\_t hdr) {**  **apply {**  **packet.emit(hdr.ethernet);**  **packet.emit(hdr.ipv4);**  **packet.emit(hdr.tcp);**  **}**  **}**  **// Switch pipeline**  **control MyVerifyChecksum(inout headers\_t hdr,**  **inout metadata\_t meta) { ... }**  **control MyComputeChecksum(inout headers\_t hdr,**  **inout metadata\_t meta) { ... }**  **V1Switch(MyParser(),**  **MyVerifyChecksum(),**  **MyIngress(),**  **MyEgress(),**  **MyComputeChecksum(),**  **MyDeparser()) main;** |

**Explanation of the P4 Code:**

1. **Header Definitions**:
   * **Ethernet, IPv4, and TCP Headers**: These headers are used to extract the necessary information from each packet for further processing.
2. **Metadata Structure**:
   * **Classifier Scores and Confidence**: Metadata fields store the predictions and confidence scores from the six classifiers (RF, DT, kNN, GNB, SVM, and BLR).
   * **Meta-Learner Output**: Stores the combined output from the meta-learner.
   * **Final Decision**: Stores the final decision on whether the traffic is an attack or normal.
3. **Parser**:
   * The parser extracts the Ethernet, IPv4, and TCP headers from incoming packets.
4. **Tables**:
   * **Feature Selection Table**: This table is used to apply the feature selection process on the packet data.
   * **Classifier Prediction Table**: This table applies each classifier's prediction logic to the packet data.
   * **Meta-Learner Table**: This table combines the predictions from the classifiers based on their confidence scores to generate a final decision.
5. **Actions**:
   * **select\_relevant\_features**: Placeholder for logic that selects relevant features from the packet.
   * **rf\_predict, dt\_predict, etc.**: Placeholders for the prediction logic of each classifier.
   * **combine\_predictions**: Combines predictions using a weighted average or other meta-learner logic.
   * **make\_final\_decision**: Determines whether the traffic is an attack based on the meta-learner's output.
6. **Control Logic**:
   * **MyIngress**: Applies feature selection, classifier predictions, and meta-learner combination in sequence, followed by the final decision-making process.
7. **Deparser**:
   * The deparser reassembles the packet headers and emits the packet after processing.

**Deployment Instructions (Emulated Environment - Mininet-WiFi)**

1. **Install Mininet and P4 Software:**

* Install Mininet and configure it for P4 development (e.g., P4Runtime and BMv2).
* Run sudo apt-get update && sudo apt-get install -y mininet.

1. **Setup Mininet-WiFi Topology:**

* Define a custom topology in Mininet-WiFi that includes the P4 switch and a few host nodes**.**
* Use a command such as: *sudo mn --topo single,3 --switch ovsk --controller remote.*

1. **Compile and Deploy P4 Code:**

* Compile the P4 program (*dynex.p4*) and generate the JSON configuration using: *p4c-bm2-ss --arch v1model --std p4-16 dynex.p4 -o dynex.json*
* Start the BMv2 switch with: *simple\_switch\_grpc --device-id 0 --log-console --no-p4 <<path-to-dynex.json>>*

1. **Configure Controllers and Hosts:**

* Use a Python script or CLI commands to configure forwarding behavior and establish connections to the remote controller.

**Deployment Instructions (Real-World Deployment)**

1. **Hardware Requirements:**

* Deploy the code on P4-enabled hardware (e.g., Tofino switches or Netronome SmartNICs).
* Ensure all devices support P4 and have P4Runtime connectivity enabled.

1. **Install P4 Runtime Environment:**

* Install P4 software (ONOS or any controller with P4Runtime) and connect to the hardware.

1. **Controller Setup:**

* Configure the controller with the dynex.json file to define rules and handle feature extraction for further ensemble processing.

1. **Monitoring and Logging:**

* Implement remote logging to track confidence scores and DDoS alerts at the controller level.