P4 Code Implementation for Feature Extraction

```
// P4 program for Data-Plane-enabled Feature Extraction

// P4 Header Definitions
header ethernet {
    address srcAddr;
    address dstAddr;
    short etherType;
}

header ip {
    bit protocol;
    address srcAddr;
    address dstAddr;
}

header tcp {
    short srcPort;
    short dstPort;
    ... other TCP header fields ...
}

header icmp {
    bit type;
    ... other ICMP header fields ...
}

// P4 Table Definitions

table flow_table {
    fields {
        ingress_port: ingress_port_t;
        ipv4_srcAddr: ipv4_address;
        ipv4_dstAddr: ipv4_address;
        tcp_srcPort: tcp_port;
        tcp_dstPort: tcp_port;
        in_metadata: metadata;  // Stores packet arrival timestamp
    }
    actions {
        add_to_feature_table;
    }
}

table feature_table {
    fields {
        flow_id: handle_t;  // Foreign key referencing flow_table entry
        wpc: counter;
        bytesw: counter;
        pps: varbit;
        bps: varbit;
        df: bool;
        ... other features from Table 2 ...
    }
}

table attack_state_table {
    fields {
        flow_id: handle_t;  // Foreign key referencing flow_table entry
        state: enum { NORMAL, POTENTIAL_ATTACK, CONFIRMED_ATTACK };
    }

// P4-based Feature Extraction

register<counter> packet_counter;

table_match my_table_match {
    type: exact;
    fields {
        standard_metadata.ingress_port: ingress_port_t;
        ethernet.etherType: short;
    }
}

table_action my_table_action {
    actions {
        packet_counter.increment();
        flow_table.add_to_feature_table;
    }
}
```

```
struct feature_data {
    handle_t flow_id;
    ... other features from Table 2 ...
}

packet ingress(packet in_packet)
 // Extract header information
    var ethernet eth;
    var { ip, tcp } hdr;
    parse(in_packet, eth, hdr);

// Match on ingress port and etherType
    if (my_table_match(standard_metadata.ingress_port, eth.etherType)) {
        packet_counter.increment();

// Flow lookup (assuming source and destination IP/port for simplicity)
        table_match flow_lookup_match {
            type: exact;
            fields {
                standard_metadata.ingress_port: ingress_port_t;
                ip.srcAddr: ipv4_address;
                ip.dstAddr: ipv4_address;
                tcp.srcPort: tcp_port;
                tcp.dstPort: tcp_port;
            }
   }
        action flow_lookup_action {
            flow_table.add_to_table(
                ingress_port: standard_metadata.ingress_port,
                ipv4_srcAddr: ip.srcAddr,
                ipv4_dstAddr: ip.dstAddr,
                tcp_srcPort: tcp.srcPort,
                tcp_dstPort: tcp.dstPort,
                in_metadata: ingress_metadata  // Store timestamp
            );
        }
        apply(flow_lookup_match, flow_lookup_action);

 // Send features to cloud server for analysis by the ensemble model
        send_to_controller(features);
}
```