P4 Implementation of P4-ESTP Algorithm

```
// P4-ESTP Algorithm Implementation (P4)

#include <core.p4>
#include <v1model.p4>

/**
 * P4-ESTP Algorithm Implementation
 */
control IngressPipe(inout headers hdr, inout metadata meta, inout standard_metadata stdmeta) {
    // State Tables
    table InConnTable {
        key = {
            hdr.ipv4.srcAddr: exact;
            hdr.ipv4.dstAddr: exact;
            hdr.tcp.srcPort: exact;
            hdr.tcp.dstPort: exact;
        }
        actions = {
            NoAction;
        }
        size = 1024;
    }

    table SYNTable {
        key = {
            hdr.ipv4.dstAddr: exact;
        }
        actions = {
            count;
            NoAction;
        }
        size = 256;
        default_action = NoAction;
    }

    table SYNACKTable {
        key = {
            hdr.ipv4.srcAddr: exact;
            hdr.ipv4.dstAddr: exact;
            hdr.tcp.srcPort: exact;
            hdr.tcp.dstPort: exact;
        }
        actions = {
            count;
            NoAction;
        }
        size = 1024;
        default_action = NoAction;
    }

    table RSTTable {
        key = {
            hdr.ipv4.srcAddr: exact;
            hdr.ipv4.dstAddr: exact;
            hdr.tcp.srcPort: exact;
            hdr.tcp.dstPort: exact;
        }
        actions = {
            count;
```

```
            NoAction;
        }
        size = 1024;
        default_action = NoAction;
    }

    table ByteTable {
        key = {
            hdr.ipv4.srcAddr: exact;
            hdr.ipv4.dstAddr: exact;
            hdr.tcp.srcPort: exact;
            hdr.tcp.dstPort: exact;
        }
        actions = {
            count_bytes;
            NoAction;
        }
        size = 1024;
        default_action = NoAction;
    }

    // Constants
    const bit SYN_TH = 1000; // Example: SYN Rate Threshold
    const bit INC_TH = 500;  // Example: Incomplete Connections Threshold
    const bit SAR_TH = 50;   // Example: SYN-ACK Ratio Threshold (as a percentage)
    const bit monWinDur = 5; // Example: Monitoring Window Duration (seconds)

    // Counters
    counter SYNCount {
        instance_count = 256;
    }

    counter SYNACKCount {
        instance_count = 1024;
    }

    counter RSTCount {
        instance_count = 1024;
    }

    counter ByteCount {
        instance_count = 1024;
    }

    // Helper Functions
    action count() {
        count.increment();
    }

    action count_bytes(bit packet_size) {
        count.add(packet_size);
    }

    action send_alert(bit severity, bit dest_ip) {
        // Implement logic to send alert to controller
        // This could involve setting metadata or using a custom header
        meta.alert_severity = severity;
        meta.alert_dest_ip = dest_ip;
        standard_metadata.egress_port = 100; // Example: Send to controller port
    }

    action clear_state_tables() {
```

```
        InConnTable.apply();
        SYNTable.apply();
        SYNACKTable.apply();
        RSTTable.apply();
        ByteTable.apply();
    }

    // Main Processing
    apply {
        if (hdr.tcp.isValid()) {
            bit syn = hdr.tcp.flags[0];
            bit ack = hdr.tcp.flags[4];
            bit rst = hdr.tcp.flags[2];

            // State Table Updates
            if (syn == 1) {
                SYNTable.apply();
                SYNCount.count();
                InConnTable.apply();
            }

            if (syn == 1 && ack == 1) {
                SYNACKTable.apply();
                SYNACKCount.count();
                InConnTable.apply();
            }

            if (rst == 1) {
                RSTTable.apply();
                RSTCount.count();
                InConnTable.apply();
            }

            ByteTable.apply();
            ByteCount.count_bytes(hdr.ipv4.totalLen);

            // DPSAD (Simplified Periodic Check)
            if (stdmeta.ingress_port == 1) { // Example: Trigger DPSAD on packets from ingress port 1
                bit syn_count = SYNCount.get();
                bit syn_rate = syn_count / monWinDur; // Simplified rate calculation

                if (syn_rate > SYN_TH) {
                    send_alert(3, hdr.ipv4.dstAddr); // High Severity
                }

                bit inc_conn_count = InConnTable.apply().hit ? 1 : 0; // Simplified InConnTable size
check

                if (inc_conn_count > INC_TH) {
                    send_alert(2, hdr.ipv4.dstAddr); // Medium Severity
                }

                bit synack_count = SYNACKCount.get();
                bit saratio = (synack_count * 100) / syn_count; // Simplified ratio calculation (as
percentage)

                if (saratio < SAR_TH) {
                    send_alert(2, hdr.ipv4.dstAddr); // Medium Severity
                }

                // Simplified Clear State Tables (Triggered periodically)
                if (stdmeta.packet_path == 1) { // Example: Clear on a special packet
```

```
                    clear_state_tables();
                }
            }
        }
    }
}

control EgressPipe(inout headers hdr, inout metadata meta, inout standard_metadata stdmeta) {
    apply {
        // Forwarding logic (simplified)
        standard_metadata.egress_port = 1; // Example: Forward to port 1
    }
}

control VerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {
        // Checksum verification logic (simplified)
    }
}

control ComputeChecksum(inout headers hdr, inout metadata meta) {
    apply {
        // Checksum computation logic (simplified)
    }
}

control DeparserImpl(inout headers hdr, out bit<variable_width> payload) {
    apply {
        emit(hdr.ethernet);
        emit(hdr.ipv4);
        emit(hdr.tcp);
    }
}

V1Switch(IngressPipe(), EgressPipe(), VerifyChecksum(), ComputeChecksum(), DeparserImpl()) main;
```

Explanation:

1. **Headers and Metadata:**
    - The code assumes standard Ethernet, IPv4, and TCP headers.
    - Metadata (meta) is used to pass alert information to the controller.
2. **State Tables:**
    - InConnTable, SYNTable, SYNACKTable, RSTTable, and ByteTable state tables are defined to store connection and traffic statistics.
    - Keys are defined based on source/destination IP addresses and ports.
    - Actions are defined to increment counters or perform no actions.
3. **Counters:**
    - Counters are used to track SYN, SYN-ACK, RST packets, and byte counts.
4. **Constants:**
    - Thresholds (SYN_TH, INC_TH, SAR_TH) and the monitoring window duration (monWinDur) are defined as constants.
5. **Helper Actions:**
    - count(): Increments a counter.
    - count_bytes(): Adds packet size to a counter.
    - send_alert(): Sets metadata to send an alert to the controller.
    - clear_state_tables(): Clears all state tables.
6. **Ingress Processing:**

- The IngressPipe control block handles packet processing.
- It checks for TCP packets and extracts SYN, ACK, and RST flags.
- State tables and counters are updated based on packet information.
- DPSAD logic is implemented with simplified periodic checks.

7. DPSAD Logic:

❖ **Alert Generation:**

- If the syn_rate exceeds SYN_TH, a high-severity alert is generated using send_alert(3, hdr.ipv4.dstAddr).
- If the inc_conn_count exceeds INC_TH, a medium-severity alert is generated using send_alert(2, hdr.ipv4.dstAddr).
- If the saratio falls below SAR_TH, a medium-severity alert is generated using send_alert(2, hdr.ipv4.dstAddr).

❖ **Simplified Periodic Checks:**

- The DPSAD checks are triggered by packets arriving on a specific ingress port (port 1 in the example). In a real-world implementation, these checks would be triggered periodically based on the monWinDur using a timer or a similar mechanism.

❖ **Clear State Tables:**

- The clear_state_tables() action is triggered by packets with a specific packet path. This is a very simplified way to simulate the monitoring windows. In a real world scenario, the P4 runtime would need to interact with external time sources, or control plane messages, to correctly time the clearing of the state tables.

8. Egress Processing:

❖ **Forwarding:**

- The EgressPipe control block handles packet forwarding.
- In this simplified example, all packets are forwarded to port 1.
- In a real world scenario, the forwarding logic would be more complex.

9. Checksum Verification and Computation:

❖ **Simplified Logic:**

- The VerifyChecksum and ComputeChecksum control blocks contain placeholder logic for checksum verification and computation.
- In a real-world implementation, these blocks would implement the necessary checksum calculations.

10. Deparser:

❖ **Header Emission:**

- The DeparserImpl control block emits the Ethernet, IPv4, and TCP headers.
- This is the stage where the packet is reconstructed.

11. Main Control Block:

❖ **V1Switch:**

- The V1Switch control block combines the ingress, egress, checksum verification, checksum computation, and deparser control blocks.
- This defines the overall P4 pipeline.

Deployment Instructions:

Emulated Deployment (Using p4c and behavioral-model):

1. **P4 Compiler (p4c):**
    - Install the p4c compiler.
    - Compile the P4 code: p4c --target bmv2 --arch v1model ldmc.p4
2. **Behavioral Model (bmv2):**
    - Install the bmv2 software switch.
    - Run the compiled P4 program: simple_switch --thrift-port 9090 ldmc.json
3. **Mininet:**
    - Create a Mininet-Wifi topology with P4-enabled switches.
    - Configure the switches to use the bmv2 instance.
    - Generate traffic to simulate SYN flood attacks.
4. **Controller (Optional):**
    - If you have a controller, configure it to connect to the switches.
    - Implement the controller-side logic to receive and process alerts.
5. **Testing:**
    - Use tools like hping3 or scapy to generate SYN flood traffic.
    - Monitor the switch's state tables and alerts to verify the detection logic.

Real-World Deployment (Using a P4-Capable Switch):

1. **P4-Capable Switch:**
    - Ensure you have a P4-capable switch that supports the V1Model architecture.
2. **P4 Compiler:**
    - Use the switch vendor's P4 compiler to compile the P4 code for the target switch.
3. **Controller:**
    - Configure the switch to connect to a controller that can receive and process alerts.
    - Implement the controller-side logic to handle alerts and apply mitigation rules.
4. **Network Configuration:**
    - Configure the network to route traffic through the P4 switch.
5. **Traffic Generation:**
    - Use traffic generators to simulate real-world traffic and SYN flood attacks.
6. **Monitoring:**
    - Monitor the switch's performance and alerts.
    - Use the controller to apply mitigation rules if necessary.