

Implementation of ECN+/Wait Algorithm in ns-3

Team Members :

- | | |
|-------------------------|----------|
| 1. G.Nithin Kumar Reddy | 211CS121 |
| 2. Guruprasad Ambesange | 211CS126 |
| 3. Sailada Sowjanya | 211CS149 |
| 4. Vaibhav Agrawal | 211CS162 |

Overview :

ECN (Explicit Congestion Notification) is a signalling mechanism used to inform TCP senders about network congestion.

Difference between ECN and it's Variants:

- ECN specification enables congested routers to mark TCP data packets during congestion, this is not the case with TCP control (TCP SYN and SYN ACK) packets. This is simply because these packets are used initially to negotiate the use of ECN options between the two endpoints.
- These variants allow one of the control packets to use the ECN flag. ECN+ basically allows SYN/ACK packets to use the ECN flag. It says the second packet of 3 way handshake must not be dropped because it will cause resources wastage if dropped. The server has to wait for one RTO time and overhead of retransmitting it.
- Similar to ECN+ , In ECN+/wait SYN/ACK packet is permitted to carry ECT(0) or ECT(1) in the IP header. It means the SYN/ACK packet can get marked by the router if there is congestion. But the difference is in ECN+ the server will reduce the congestion window to one segment and send immediately 1 data packet with ECT. In ECN+/wait the Server will wait for 1 RTT time period to let

the router come out of the congestion state. Then, the server will send one data packet with ECT.

ACTIVITIES:

Activity-1:

Begin by reading relevant RFCs related to ECN and TCP . Key RFCs to focus on include RFC 3168 for ECN specification and RFC 793 for TCP. And installed the NS-3 network simulator (ns-3.41) with necessary tools on Ubuntu (22.04) by referring to NS-3 documentation and NS-3-Users google group for assistance with resolving queries.








Output after running first.cc -

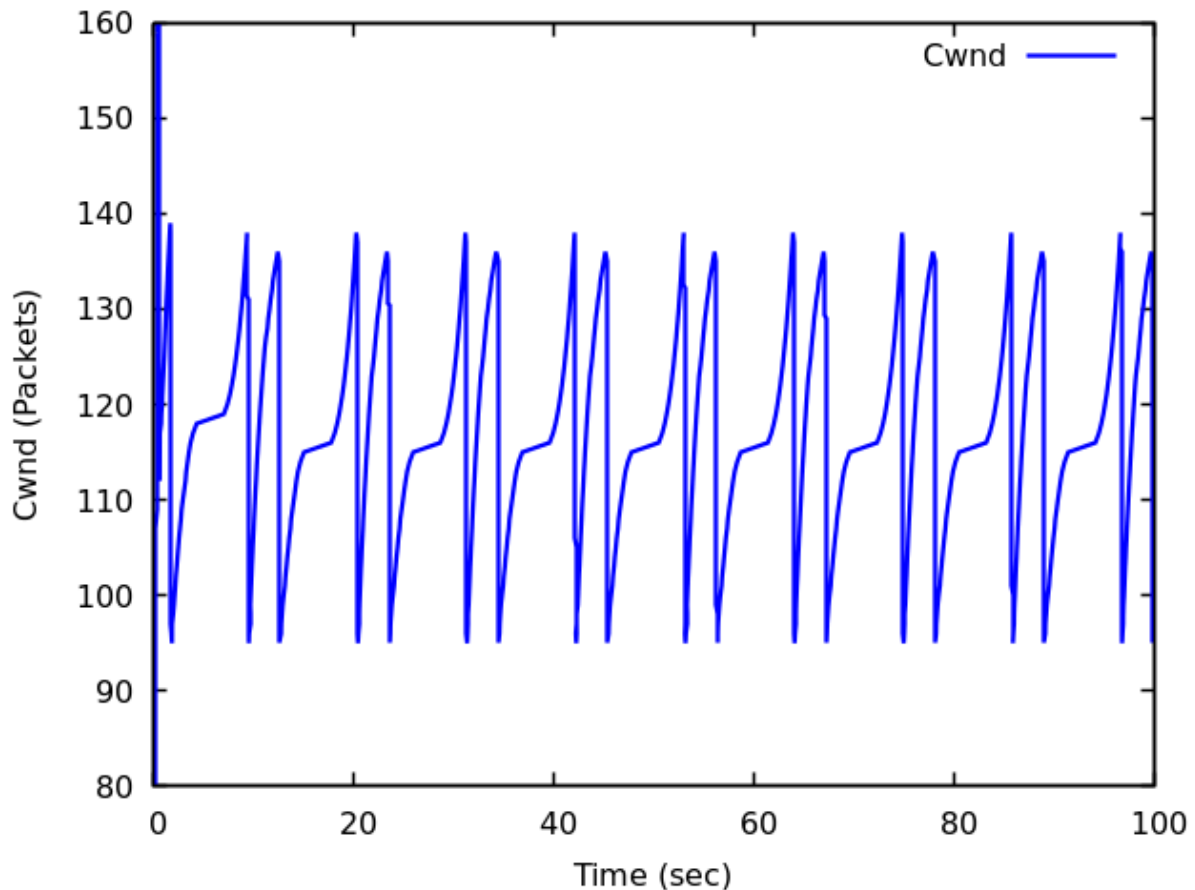
```
[0/2] Re-checking globbed directories...  
[985/985] Linking CXX executable ../build/scratch/ns3-dev-first-default  
At time +2s client sent 1024 bytes to 10.1.1.2 port 9  
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153  
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153  
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

Activity-2:

Ran Tcp-Cubic-Example

- We ran the Tcp-Cubic after necessary changes in the file tcp-bbr-example from the path examples/tcp/tcp-bbr-example, to get the metrics like cwnd, throughput, queueSize and got familiarised with gnuplot commands.

Name	Size	Modified	
 cwnd.dat	24.6 kB	19 Mar	☆
 cwnd.png	33.3 kB	19 Mar	☆
 gnuplotScriptCwnd	261 bytes	19 Mar	☆
 gnuplotScriptQueueSize	294 bytes	19 Mar	☆
 gnuplotScriptThroughput	288 bytes	19 Mar	☆
 queueSize.dat	3.8 kB	19 Mar	☆
 throughput.dat	8.7 kB	19 Mar	☆



Activity-3:

To modify the `Send()` function in NS-3 to set SYN/ACK packets eligible for ECN marking, we need to locate the relevant section of code responsible for generating SYN/ACK packets and add logic to enable ECN for these packets. That is ,

- We set the SYN and ACK flags in the TCP header to indicate a SYN-ACK packet.
- We enable the ECE (Explicit Congestion Notification Echo) flag in the TCP header to support ECN.
- We set the receiver's state to `ECN_IDLE` to indicate that it supports ECN.
- We enable the ECT (Explicit Congestion Notification-Capable Transport) bit in the IP header using the `AddSocketTags()` function with the `withECT` parameter set to true.
- We set the IP TOS (Type of Service) field to indicate ECN capability using the `Setiptos()` function with the `withECT` parameter set to true.

```

// Define a variable to store the RTT
Time m_rtt;

TcpSocketBase::Send(Ptr<Packet> p, uint32_t flags) {
    NS_LOG_FUNCTION(this << p);
    NS_ABORT_MSG_IF(flags, "use of flags is not supported in TcpSocketBase::Send()");

    if (m_state == ESTABLISHED || m_state == SYN_SENT || m_state == CLOSE_WAIT) {
        // Store the packet into Tx buffer
        if (!m_txBuffer->Add(p)) {
            m_errno = ERROR_MSGSIZE;
            return -1;
        }
        if (m_shutdownSend) {
            m_errno = ERROR_SHUTDOWN;
            return -1;
        }

        // Calculate and store the RTT value
        m_rtt = CalculateRtt();

        // Submit the data to lower layers after waiting for one RTT
        Simulator::Schedule(m_rtt, &TcpSocketBase::SendPendingData, this, m_connected);

        return p->GetSize();
    } else {
        m_errno = ERROR_NOTCONN;
        return -1;
    }
}

```

```

void TcpSocketBase::SendPendingData(bool connected) {
    // Check if the connection is still active
    if (!connected) {
        return;
    }

    // Check if one RTT has passed
    if (Simulator::Now() < m_lastSendTime + m_rtt) {
        // Wait for one RTT before sending data
        Simulator::Schedule(m_rtt, &TcpSocketBase::SendPendingData, this, connected);
        return;
    }

    //SendPendingData function code from ECN implementation part in NS-3

    // Update last send time
    m_lastSendTime = Simulator::Now();
}

```

Activity-4:

- The way to find RTT is by extracting Timestamps in relevant packets from pcap traces, and make the necessary calculation for RTT. We can add the following code in the topology file, to enable pcap tracing and finding RTT.

```
// Enable pcap tracing for a specific net device (e.g., client-side)
Ptr<NetDevice> clientDevice = ...;
clientDevice->EnablePcap("client-trace", clientDevice);

// Enable pcap tracing for another net device (e.g., server-side)
Ptr<NetDevice> serverDevice = ...;
serverDevice->EnablePcap("server-trace", serverDevice);

// Extract timestamps from pcap traces
PcapFileWrapper clientTrace("client-trace.pcap");
PcapFileWrapper serverTrace("server-trace.pcap");

Time requestTimestamp = clientTrace.GetTimestamp(0);
Time responseTimestamp = serverTrace.GetTimestamp(0);

// Calculate RTT
Time rtt = responseTimestamp - requestTimestamp;
NS_LOG_INFO("RTT: " << rtt.GetNanoSeconds() << " ns");
```

Another way to find RTT using ElapsedTimeFromTsValue

```
TcpOptionTS::ElapsedTimeFromTsValue(uint32_t echoTime)
{
    uint64_t now64 = (uint64_t)Simulator::Now().GetMilliSeconds();
    uint32_t now32 = now64 & 0xFFFFFFFF;

    Time ret = Seconds(0.0);
    if (now32 > echoTime)
    {
        ret = MilliSeconds(now32 - echoTime);
    }

    return ret;
}
```

- ElapsedTimeFromTsValue function which returns the 1 RTT using timestamp.

<https://github.com/nsnam/ns-3-dev-git/blob/master/src/interne t/model/tcp-option-ts.cc>

Challenges Faced :

- Took almost one day to successfully install ns-3 due to various dependency management issues, platform compatibility concerns, and compiler-related challenges
- Faced installation issues , while installing ns3. The waf command was not working in newer versions of ns3.
- Figuring out a suitable RTT estimate function was difficult. Referred ns-3 google groups to resolve the issues like initialising RTT estimator pointer from NULL pointer.
- Identifying the relevant sections of ns-3's TCP implementation where SYN/ACK eligibility is determined was challenging due to the size and complexity of the codebase.
- Modifying SYN/ACK eligibility in original ECN code i.e modifying the Send() function in NS-3 to set SYN/ACK packets eligible for ECN marking, we need to locate the relevant section of code responsible for generating SYN/ACK packets and add logic to enable ECN for these packets.
- Melding the changes into existing code and building a topology to get the metrics when ECN+/wait algorithm is used.