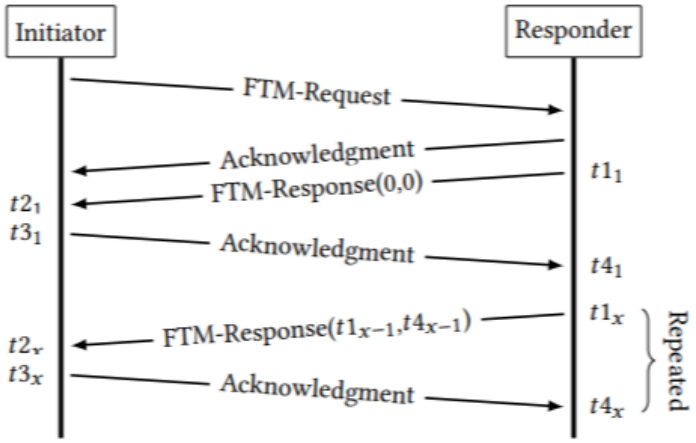
Wi-Fi QoS bridged to TSN topology

Additional modifications: **ported to ns-3.36.1**

Composition

- TSN-enabled switch and nodes connected to Wi-Fi 6 AP and stations with its own switch.
- The simulation uses PerfectClockModel for time sync
  - Different clocks but synchronized by design
  - An additional goal is to implement Fine Time Measurement (FTM) to obtain timestamps



Advantages

- QoS allows setting of priority of transmissions in a multimodal, bidirectional system, and is built into the WiFi model.

    - The specified topology has 3 STAs, 1 AP, 2 switches for the wireless and wired ends, and 3 wired nodes.
    - I implemented applications running on all nodes, transferring data back and forth to simulate traffic.
- QoS allows deterministic transmission for high-priority real-time applications, minimizing the impact of concurrent best-effort traffic.
- Additionally, I installed the time-aware shaper on to the switches, as well as a porting process to ns-3.36 to utilize the new build system and enable newer features of the ns-3 stack to be used.

Configuration

- Setup process of ns-3.34 (I used **bake**) and ns-3.36.
    - Ubuntu 20.04 containerized instance
        - `sudo apt install g++ python3 python3-dev pkg-config sqlite3 cmake python3-setuptools git qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools gir1.2-goocanvas-2.0 python3-gi python3-gi-cairo python3-pygraphviz gir1.2-gtk-3.0 ipython3 openmpi-bin openmpi-common openmpi-doc libopenmpi-dev autoconf cvs bzr unrar gsl-bin libgsl-dev libgslcblas0 wireshark tcpdump sqlite libxml2 libxml2-dev libc6-dev libc6-dev-i386 libclang-dev llvm-dev automake python3-pip libxml2 libxml2-dev libboost-all-dev`
        - `git clone https://gitlab.com/nsnam/ns-3-dev.git && cd ns-3-dev`
        - `git checkout -b ns-3.36.1-branch ns-3.36.1`
        - VS Code running on the same instance to utilize container's libraries
        - The porting process resolves the non-termination issue in ns-3.34 since 3.36 supports concurrent execution of applications and Flowmon displays statistics after the execution is over. Additionally, CMake allows easier debugging through IDEs.
            - launch.json for debugging ns-3.36

```
"version": "0.2.0",
    "configurations": [
        {
```

```
            "name": "(gdb) Launch from scratch",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/build/scratc
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "preLaunchTask": "Build",
            "environment": [
                {
                    "name": "LD_LIBRARY_PATH",
                    "value": "${workspaceFolder}/build/
                }
            ],
```

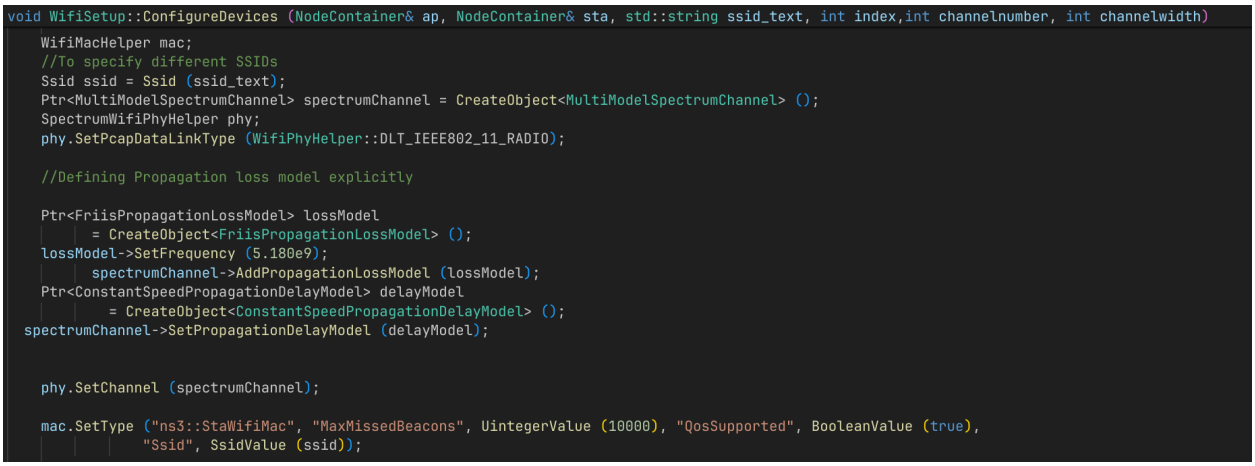
- Pipe Launching for ns-3.34

```
{
            "name": "(gdb) Pipe Launch",
            "type": "cppdbg",
            "request": "launch",
            //"program": "${workspaceFolder}/build/util
            "program": "${workspaceFolder}/build/scratc
            //"args": ["--nWifi=12"],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "pipeTransport": {
                "debuggerPath": "",  // leave blank
                "pipeProgram": "${workspaceFolder}/waf"
                // pipeArgs is essentially the entire w
                "pipeArgs": [
                    "--command-template", "\"",
                    "${debuggerCommand}",
```

```
                    "--args", "%s",
                    "\"",
                    "--run", "${fileBasenameNoExtensio
                    ],
                    "quoteArgs":false,
                    "pipeCwd": ""
                },
```

Implementation

The first step after configuration is enabling QosSupported in *src/tsnwifi/wifi-setup.cc*



```
void WifiSetup::ConfigureDevices (NodeContainer& ap, NodeContainer& sta, std::string ssid_text, int index,int channelnumber, int channelwidth)
    WifiMacHelper mac;
    //To specify different SSIDs
    Ssid ssid = Ssid (ssid_text);
    Ptr<MultiModelSpectrumChannel> spectrumChannel = CreateObject<MultiModelSpectrumChannel> ();
    SpectrumWifiPhyHelper phy;
    phy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);

    //Defining Propagation loss model explicitly

    Ptr<FriisPropagationLossModel> lossModel
         = CreateObject<FriisPropagationLossModel> ();
    lossModel->SetFrequency (5.180e9);
         spectrumChannel->AddPropagationLossModel (lossModel);
    Ptr<ConstantSpeedPropagationDelayModel> delayModel
          = CreateObject<ConstantSpeedPropagationDelayModel> ();
  spectrumChannel->SetPropagationDelayModel (delayModel);


    phy.SetChannel (spectrumChannel);

    mac.SetType ("ns3::StaWifiMac", "MaxMissedBeacons", UintegerValue (10000), "QosSupported", BooleanValue (true),
                 "Ssid", SsidValue (ssid));
```

```
mac.SetType ("ns3::ApWifiMac",
                "EnableBeaconJitter", BooleanValue (false), "Qos
                "Ssid", SsidValue (ssid));
```

Create InetSocketAddress objects for QoS tagging, based on AC Matrix followed in ns-3.34+

```
Ipv4Address apIp = APWifiInterfaces.GetAddress (0);
Ipv4Address staIP1 = staInterfaces.GetAddress (0);
Ipv4Address staIP2 = staInterfaces.GetAddress (1);
Ipv4Address staIP3 = staInterfaces.GetAddress (2);
```

```
InetSocketAddress apAddress (apIp, 10);
apAddress.SetTos (0x0);

InetSocketAddress sta1Address (staIP1, 10);
sta1Address.SetTos (0x2e);

InetSocketAddress sta2Address (staIP2, 10);
sta2Address.SetTos (0x18);

InetSocketAddress sta3Address (staIP3, 11);
sta3Address.SetTos (0x14);
```

Mapping the values of the DS field onto user priorities is performed similarly to the Linux mac80211 subsystem. Basically, the **ns3::WifiNetDevice::SelectQueue()** method sets the user priority (UP) of an MSDU to the three most significant bits of the DS field. The Access Category is then determined based on the user priority according to the following table:

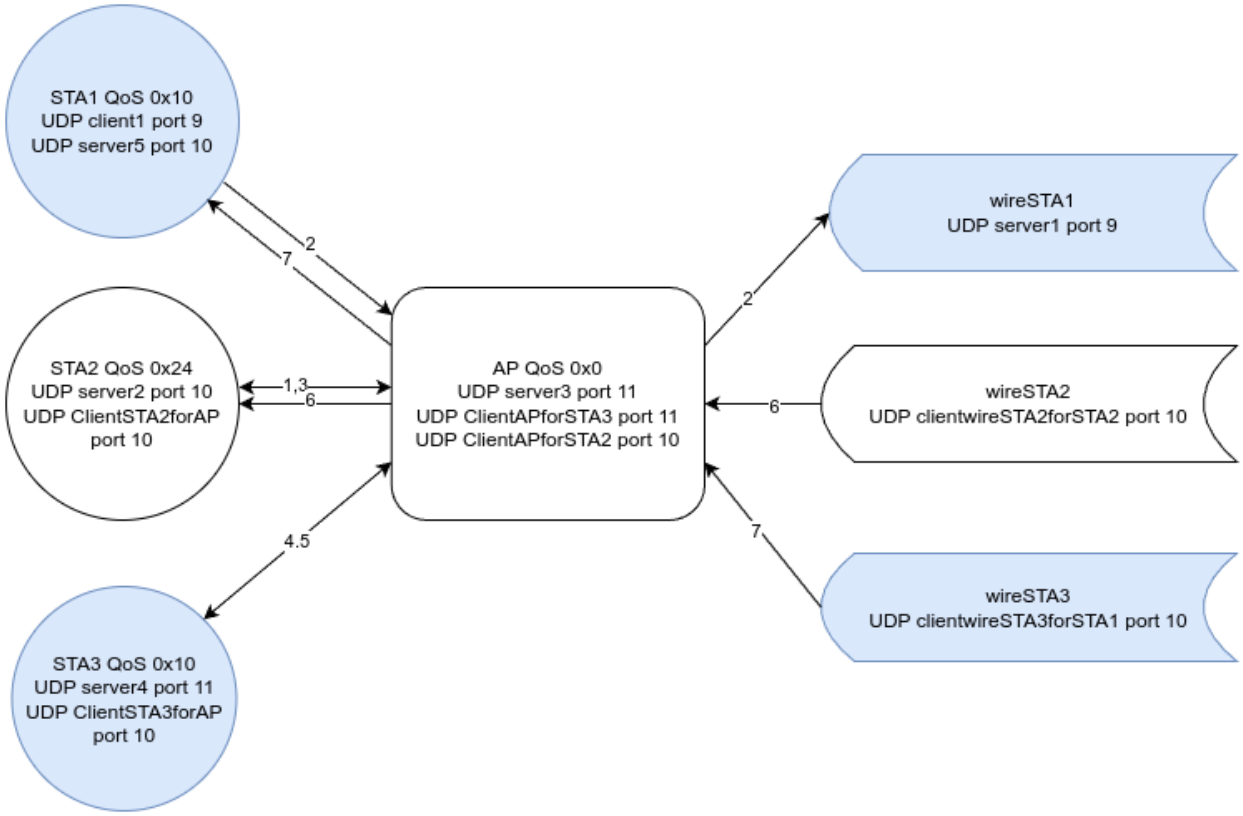
UP	Access Category
7	AC_VO
6	AC_VO
5	AC_VI
4	AC_VI
3	AC_BE
0	AC_BE
2	AC_BK
1	AC_BK

TOS and DSCP values map onto user priorities and access categories according to the following table.

DiffServ PHB	TOS (binary)	UP	Access Category
EF	101110xx	5	AC_VI
AF11	001010xx	1	AC_BK
AF21	010010xx	2	AC_BK
AF31	011010xx	3	AC_BE
AF41	100010xx	4	AC_VI
AF12	001100xx	1	AC_BK
AF22	010100xx	2	AC_BK
AF32	011100xx	3	AC_BE
AF42	100100xx	4	AC_VI
AF13	001110xx	1	AC_BK
AF23	010110xx	2	AC_BK
AF33	011110xx	3	AC_BE
AF43	100110xx	4	AC_VI
CS0	000000xx	0	AC_BE
CS1	001000xx	1	AC_BK
CS2	010000xx	2	AC_BK
CS3	011000xx	3	AC_BE
CS4	100000xx	4	AC_VI
CS5	101000xx	5	AC_VI
CS6	110000xx	6	AC_VO
CS7	111000xx	7	AC_VO

TOS Matrix, ns-3.34 docs

- A topology is implemented to enable cross-communication between nodes on the network using UDP to generate traffic.




```
ApplicationContainer serverApp1
UdpServerHelper server1 (9);
serverApp1 = server1.Install (wireStaNodeContainer.Get (0))
serverApp1.Start (Seconds (0.0));
serverApp1.Stop (Seconds (simulationTime + 1));

UdpClientHelper client1 (wireInterfaces.GetAddress (0), 9);
client1.SetAttribute ("MaxPackets", UintegerValue (42949672
client1.SetAttribute ("Interval", TimeValue (Time ("0.002"))
client1.SetAttribute ("PacketSize", UintegerValue (200)); /
ApplicationContainer clientApp1 = client1.Install (wifiStaN
clientApp1.Start (Seconds (0.2));
clientApp1.Stop (Seconds (simulationTime + 1.2));
```

```
    ApplicationContainer serverAppSTA2;
    UdpServerHelper server2 (10);
    serverAppSTA2 = server2.Install (wifiStaNodeContainer.Get (n
    serverAppSTA2.Start (Seconds (0));
    serverAppSTA2.Stop (Seconds (simulationTime + 1));

    UdpClientHelper clientAPforSTA2 (sta2Address, 10);
    clientAPforSTA2.SetAttribute ("MaxPackets", UintegerValue (4
    clientAPforSTA2.SetAttribute ("Interval", TimeValue (Time ("
    clientAPforSTA2.SetAttribute ("PacketSize", UintegerValue (1
    ApplicationContainer clientApp2 = clientAPforSTA2.Install (A
    clientApp2.Start (Seconds (0.2));
    clientApp2.Stop (Seconds (simulationTime + 1.2));

    ApplicationContainer serverAppAP;
    UdpServerHelper server3 (10);
    serverAppAP = server3.Install (APNodeContainer.Get(0));
    serverAppAP.Start (Seconds (0.0));
    serverAppAP.Stop (Seconds (simulationTime + 1));

    UdpClientHelper clientSTA3forAP (apAddress, 10);
    clientSTA3forAP.SetAttribute ("MaxPackets", UintegerValue (4
    clientSTA3forAP.SetAttribute ("Interval", TimeValue (Time ("
    clientSTA3forAP.SetAttribute ("PacketSize", UintegerValue (1
    ApplicationContainer clientApp3 = clientSTA3forAP.Install (w
    clientApp3.Start (Seconds (0.2));
    clientApp3.Stop (Seconds (simulationTime + 1.2));

    ApplicationContainer serverAppSTA3;
    UdpServerHelper server4(11);
    serverAppSTA3 = server4.Install (wifiStaNodeContainer.Get (n
    serverAppSTA3.Start (Seconds (0.0));
    serverAppSTA3.Stop (Seconds (simulationTime + 1));

    UdpClientHelper clientAPforSTA3 (sta3Address, 11);
    clientAPforSTA3.SetAttribute ("MaxPackets", UintegerValue (4
```

```
    clientAPforSTA3.SetAttribute ("Interval", TimeValue (Time ("
    clientAPforSTA3.SetAttribute ("PacketSize", UintegerValue (1
    ApplicationContainer clientApp4 = clientAPforSTA3.Install (A
    clientApp4.Start (Seconds (0.2));
    clientApp4.Stop (Seconds (simulationTime + 1.2));

    UdpClientHelper clientSTA2forAP (apAddress, 10);
    clientSTA2forAP.SetAttribute ("MaxPackets", UintegerValue (4
    clientSTA2forAP.SetAttribute ("Interval", TimeValue (Time ("
    clientSTA2forAP.SetAttribute ("PacketSize", UintegerValue (1
    ApplicationContainer clientApp5= clientSTA2forAP.Install (wi
    clientApp5.Start (Seconds (0.2));
    clientApp5.Stop (Seconds (simulationTime + 1.2));

    UdpClientHelper clientwireSTA2forSTA2 (sta2Address, 10);
    clientwireSTA2forSTA2.SetAttribute ("MaxPackets", UintegerVa
    clientwireSTA2forSTA2.SetAttribute ("Interval", TimeValue (T
    clientwireSTA2forSTA2.SetAttribute ("PacketSize", UintegerVa
    ApplicationContainer clientApp6= clientwireSTA2forSTA2.Insta
    clientApp6.Start (Seconds (0.2));
    clientApp6.Stop (Seconds (simulationTime + 1.2));

    ApplicationContainer serverAppSTA1;
    UdpServerHelper server5 (10);
    serverAppSTA1 = server5.Install (wifiStaNodeContainer.Get (n
    serverAppSTA1.Start (Seconds (0.0));
    serverAppSTA1.Stop (Seconds (simulationTime + 1));

    UdpClientHelper clientwireSTA3forSTA1 (sta1Address, 10);
    clientwireSTA3forSTA1.SetAttribute ("MaxPackets", UintegerVa
    clientwireSTA3forSTA1.SetAttribute ("Interval", TimeValue (T
    clientwireSTA3forSTA1.SetAttribute ("PacketSize", UintegerVa
    ApplicationContainer clientApp7= clientwireSTA3forSTA1.Insta
    clientApp7.Start (Seconds (0.2));
    clientApp7.Stop (Seconds (simulationTime + 1.2));
```

Testing

- Flowmon
  - ns3::FlowMonitorHelper
  - ns3::Ipv4HelperClassifier - statistics segregation per flow

```
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator
    {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlo

        NS_LOG_UNCOND("\n----Flow ID:" <<iter->first);
        NS_LOG_UNCOND("Src Addr " <<t.sourceAddress << " Dst
        NS_LOG_UNCOND("Sent Packets=" <<iter->second.txPacket
        NS_LOG_UNCOND("Received Packets =" <<iter->second.rxP
        //NS_LOG_UNCOND("Lost Packets =" <<iter->second.txPac
        NS_LOG_UNCOND("Packet delivery ratio =" <<iter->secon
        //NS_LOG_UNCOND("Packet loss ratio =" << (iter->secon
        NS_LOG_UNCOND("Packet loss percentage =" << iter->sec
        NS_LOG_UNCOND("Avg. Delay =" << NanoSeconds(((iter->s
        NS_LOG_UNCOND("Sum. Delay =" << iter->second.delaySum
        NS_LOG_UNCOND("Avg. Jitter =" << NanoSeconds(((iter->
        NS_LOG_UNCOND("Sum. Jitter =" <<iter->second.jitterSu
    }
```

Dataset QoS ns-3 measurements.xlsx

  - For a more accurate picture, a manual measurement using PCAP files is required.

```
// in TSNWifi.cc
csma.EnablePcap ("TSNWifi-wire.pcap", APDevices.Get (0), true
csma.EnablePcap ("TSNWifi-wire", wireDevices.Get (0), true);
csma.EnablePcap ("TSNWifi-Switch1-AP", switchFullLink.Get (1
```

```
csma.EnablePcap ("TSNWifi-Switch1-Node0", switch1Link.Get (0)
csma.EnablePcap ("TSNWifi-Switch1-Switch0", switch1Link.Get (
csma.EnablePcap ("TSNWifi-Switch0-Switch1", switchFullLink.Ge

// in wifi-setup.cc
phy.EnablePcap ("AccessPoint", m_netDeviceContainerAP.Get (0)
phy.EnablePcap ("StationBG", m_netDeviceContainerSTA.Get (0)
phy.EnablePcap ("StationFG", m_netDeviceContainerSTA.Get (1)
```

I am using Wireshark to analyze traffic.