Network topology of the system

The following figure shows the network topology for the test, it’ very simple: just boot the boards, wait ~ 30 sec for WDS establishment, connect 2 laptops via Wi-Fi (1 for ping/iperf and the other to look at the board’s load via GUI) the on the other end connect to the ethernet port another laptop which will be used for ping / iperf client or server. On VLC you have to transmit a multicast stream in the 224.0.0.1 group with a standard video.

What we see from the “[Environmental Settings](Environmental%20settings.docx)” topology (which is pretty much the same as this) is that there is only one big network (192.168.2.0/24).

Just do a simple ping on the other end of the network with no active clients (for now the video transmission is stopped). Then execute iperf, I did this with both TCP and UDP but collected data only for the TCP case (ease of calculation). Now transmit the video and start pinging, you should see no response at all. As outlined in the [analysis](Analisi.html) file we have experienced a complete collapse of the network during the video projection. Here are the video data:

* Format: MPEG-4
* Codec: H264
* Bit rate: lowest possible
* Location: Legacy device (ethernet connected one)
* Resolution: HD

I calculated the mean time delay, T, to send a frame onto the channel with capacity 8 Mbps. I assumed that the frames arrive randomly with an average arrival rate of λ frames/sec, and that the frames vary in length with an average length of 1/μ bits. With these parameters, the service rate of the channel is μC frames/sec. A standard queueing method used is the ‘‘M/M/1’’ queue. These are general and approximated numbers but by computing the delay vs board-load, I expected a 200 % increase of the delay at 65% of the board load, but this happened just at 20 % load when I just pressed transmit in VLC. The sending legacy device reached the onset of congestion, just 1-2 sec after transmitting. These results are in big contrast with the theory (which is based on a queue management).

The network collapse could have been caused by a backlog of non MTU compliant packets, but that was not the case, because I looked at the packet size in Wireshark and it was well below of the MTU (1472 B). In fact the packet dimension was around 1350 B.

At this point I thought that the problem must be the flow management enforced by OpenWRT, because I couldn’t find any other issues on IP level management, and the main reason to look towards flow management is that this issue appears only on video transmitting. What I found on the Internet is that OpenWRT uses no queue flow management by default on Barrier Breaker, but it uses the Linux base queue management which is thought as a general control law for every type of traffic, thus it’s not specifically made for video (bandwidth demanding) and “normal” traffic together, which is sort of “4 seasons” policy.

It seems like it waits until the incoming buffer bucket fills up and then unloads everything without control on the radio channel. What I wanted to try was enforcing the “fire and forget” policy by reducing the bucket size of the incoming traffic to force the forwarding process at the router without necessarily waiting for other traffic, but to avoid starvation of other fluxes I should manage everything using a priority-based flow control. I couldn’t find a way to implement in OpenWRT something like a pfifo or HTB (not so good in my opinion but why not try).

This is what I could see from the analysis and I hope this queue management can fix this, I saw a dedicated package SQM to solve this bufferbloat problem.