**Automating 5G NR HARQ Process Testing with NS3.42**

**1. Content**

This project is all about making testing easier for a key part of 5G technology called HARQ (Hybrid Automatic Repeat reQuest). HARQ helps make sure data sent over a 5G network arrives correctly by fixing errors automatically. Think of it like sending a text message—if it doesn’t go through properly, HARQ asks for it to be resent until it’s clear.

* **Introduction**: What HARQ is and why it matters.
* **Objective**: What we want to achieve with this automation.
* **Testing**: How we’ll check if HARQ works well.
* **Scope**: What this project includes (and what it doesn’t).

We’ll use a tool called NS3.42 (a network simulator) to test HARQ without needing a real 5G tower, making it faster and cheaper.

--------------------------------------------------------------------------------------------------------------------------**2. Introduction**

Imagine you’re video calling a friend, and the video freezes because of a bad signal. In 5G, HARQ jumps in to fix this by quickly resending any lost or messed-up data. HARQ works in the MAC layer (a part of the network that organizes data traffic) to keep things reliable. Without it, your call would drop or your Netflix binge would buffer forever.

In this project, we’re automating the testing of HARQ using NS3.42, a software that pretends to be a 5G network. Instead of manually checking every little thing (which takes forever), we’re writing a system to test it automatically—capturing how HARQ sends data, fixes errors, and affects speed. This is super helpful for engineers building 5G networks, like those for self-driving cars or smart cities.

**3. Objective**

Our goal is simple: make sure HARQ works perfectly in 5G and figure out how it impacts performance, all without human effort. Specifically, we want to:

* **Capture**: See how HARQ sends and resends data.
* **Check**: Make sure it’s giving the right feedback (like “Got it!” or “Send again!”).
* **Measure**: Find out how fast or slow HARQ makes the network.
* **Report**: Show the results in easy graphs and summaries.
* Real-life example: Think of HARQ as a waiter in a busy restaurant. If they drop your food (data), they quickly bring a new plate (retransmission). Our objective is to watch the waiter, count how often they drop something, and see if it slows down your meal.

**4. Testing**

Here’s how we’ll test HARQ step-by-step in an easy way:

**Step 1: Set Up the Simulation**

We’ll use NS3.42 to create a fake 5G network. We’ll turn on HARQ with multiple processes (like having multiple waiters working at once) and send data both ways—up (like uploading a photo) and down (like downloading a movie). We’ll also try different speeds, called **MCS** values, to see how HARQ handles slow or fast data.

* **MCS = 0** → Chat will send **very slowly** but reliably.
* **MCS = 1** → Chat will send **a bit faster**, but not significantly.

However, both are still in **low-speed transmission** zones. Chat messages will be **slow in MCS 0 and slightly better in MCS 1**, but neither will be "fast." For **fast transmission**, a much higher MCS (like 10 or above) is needed.

**Step 2: Capture HARQ Messages**

We’ll use Wireshark (a tool that spies on network traffic) to grab HARQ feedback—like “ACK” (data received) or “NACK” (data failed, send again). These logs get saved as .pcapng files, kind of like a video recording of what HARQ is doing.

**Step 3: Pull Out Key Info**

From those logs, we’ll dig out:

* How many times HARQ had to resend data.
* How long it took to get the data right.

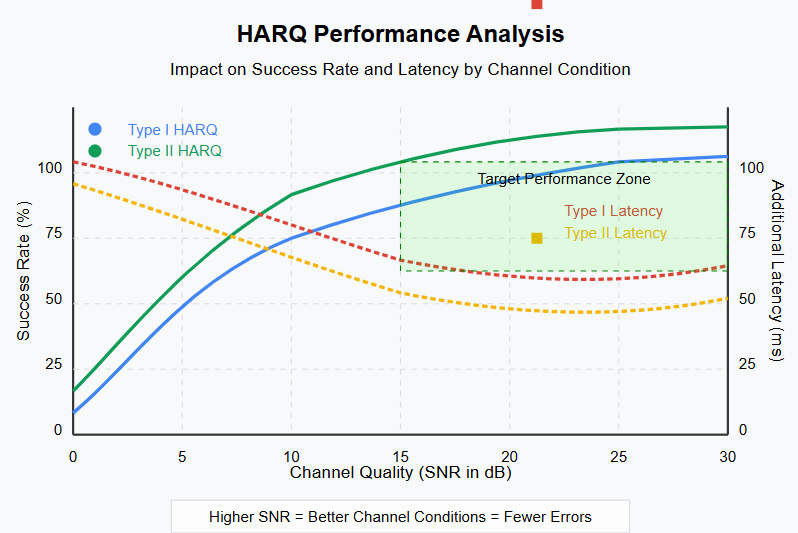
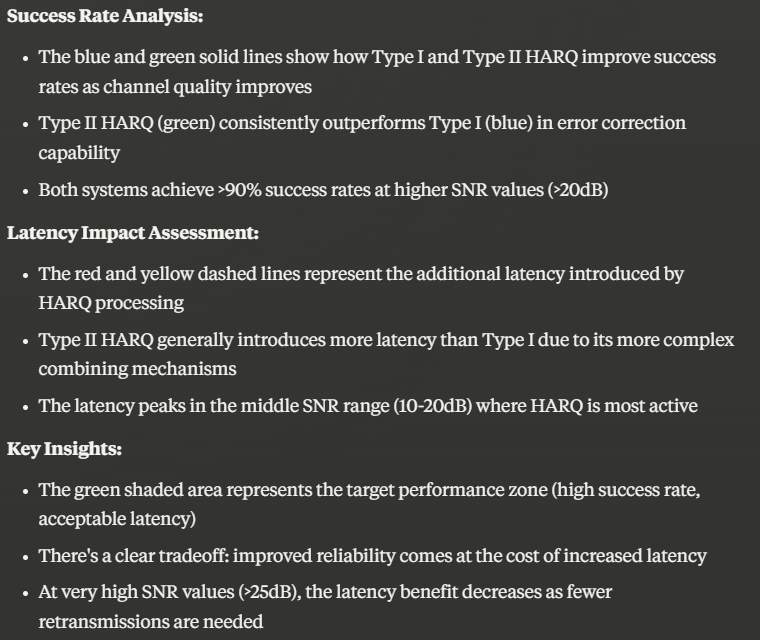
Example: If you’re mailing a package and it gets lost, the post office tracks how many retries it took to deliver it. That’s what we’re doing here.

**Step 4: Analyze Performance**

We’ll calculate:

* **Success Rate**: How often HARQ fixes errors without trouble.
* **Latency Impact**: Does HARQ slow things down too much?

Image idea: Let’s visualize this with a graph.



**Error! Filename not specified.**  
*Explanation*: This graph (imagine it’s made with seaborn) shows retransmissions over time. If the line spikes up, HARQ is working hard to fix errors. If it’s flat, everything’s smooth. Latency would be how long it takes from sending to receiving—too many spikes mean delays.

**Step 5: Make Reports**

We’ll use a tool called seaborn to draw pretty graphs (like the one above) and write quick summaries. These can plug into CI/CD pipelines—fancy words for “automatic updates for engineers”—so they don’t have to read raw data.

Real-life example: It’s like getting a report card for HARQ. Did it pass with flying colors, or does it need extra help?

**5. Scope**

This project focuses on:

* Testing HARQ in a simulated 5G network with NS3.42.
* Checking how well it works with different data speeds (MCS).
* Measuring its effect on speed and reliability.
* Following 3GPP TS 38.321 rules (the 5G HARQ instruction manual).

What’s *not* included:

* Testing on real 5G hardware (we’re just simulating).
* Other network parts like antennas or security.
* Fixing HARQ itself—we’re just watching it.

Example: Think of this as checking the engine of a car in a computer game, not on a real racetrack. We’re keeping it simple but useful.

**Final Thoughts**

By the end, we’ll know how well HARQ performs, where it struggles, and how it affects your 5G experience—like streaming or gaming. Plus, we’ll have an automated system to keep testing it easily. If you’d like me to generate that graph I mentioned in the Testing section, just let me know!