Timing Measurement Report for GR-WiFi

Natong Lin, Shengli Zhou, and Song Han
May 19, 2025

This report aims to evaluate the real-time performance of the GR-WiFi receiver by measuring the latency and processing behavior of key PHY-layer components. We focus on identifying computational bottlenecks and profiling block-level latency under various payload sizes, modulation schemes, packet formats, and decoder implementations. The evaluation includes both measurements with and without USRP hardware.

In measurements without USRP, we measure the latency of individual GNU Radio blocks, particularly the demod and decode blocks, as they are the most computationally intensive on the receive path. We also perform round-trip time (RTT) measurements by looping the transmitter and receiver flowgraphs within GNU Radio. In measurements with USRP, we measure end-to-end RTT using a single USRP device, where the transmit and receive ports are connected via a coaxial cable. These measurements provide a timing reference for the duration required for a packet to traverse the complete transmit and receive chains.

Latency profiling is conducted by recording timestamps when each packet enters and exits a block. Since GNU Radio's threaded scheduler may invoke general_work() multiple times per packet within each block, we accumulate the total elapsed time across all relevant invocations to capture the complete processing latency for that packet. The timing measurement thus reflects the total duration a packet spends within a block, from initial arrival to final departure. VHT-format packets are used for the evaluation, reflecting the system's MU-MIMO operating mode.

1. Timing measurements without USRP

We conducted experiments for two payload sizes: 30 bytes and 300 bytes in 3 independent runs. For each MCS index, 10,000 packets were loaded as file input. The results report the minimum, maximum, and average processing time per packet in each block. All measurements were performed offline, entirely within GNU Radio, without involving any hardware.

Fig. 1 and 2 show the latency profiling results for SISO VHT reception with 30-byte and 300-byte payloads, respectively. In the 30-byte case, both the demod and decode blocks exhibit low latency in the millisecond range with minimal variance, indicating efficient and stable packet processing.

In contrast, the 300-byte case shows significantly higher latency and jitter, particularly in the demod block. This result is caused by our custom Viterbi decoder used in the decode block, which is

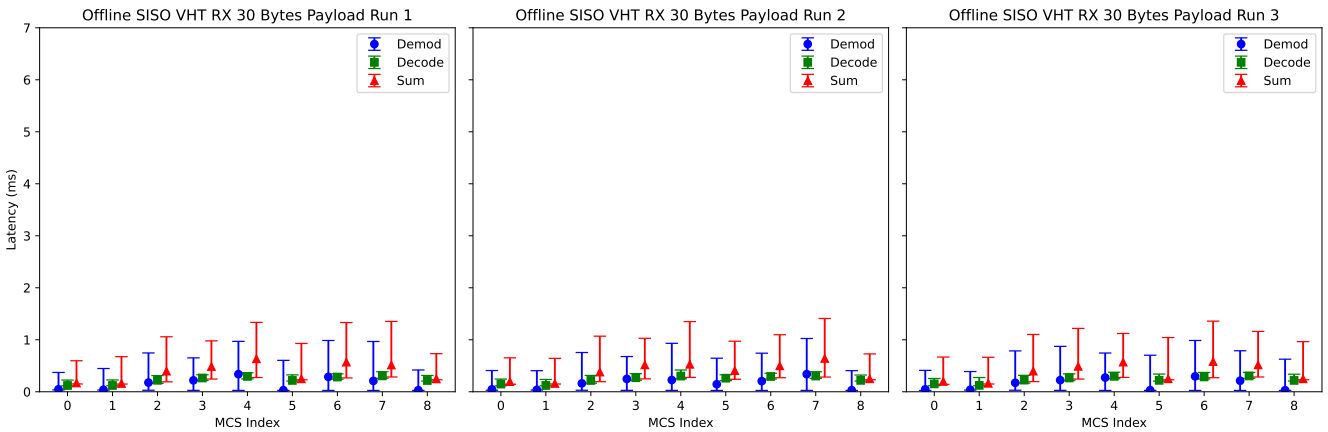

Fig. 1. Timing measurement on demod and decode block.

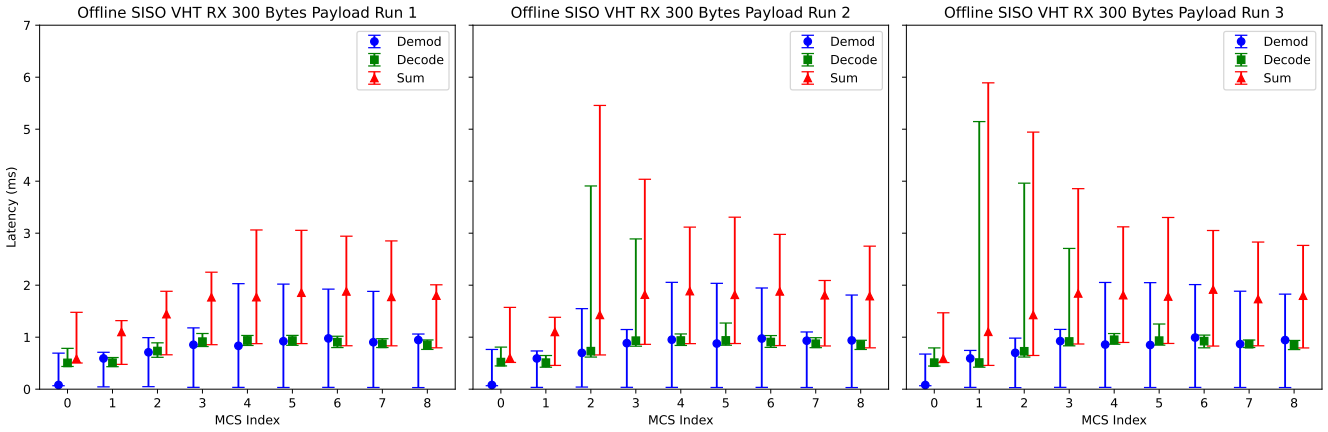

Fig. 2. Timing measurement on demod and decode block.

not optimized for streaming.

We observed that the high latency in the decoder causes its input buffer to remain occupied, which in turn blocks new samples from entering the decode block. This creates backpressure on the preceding demod block, preventing it from outputting samples, which causes the stalling of processing pipeline. The root cause is that the decoder performs traceback only after receiving the full set of soft bits, resulting in output delay. This delay prolongs buffer occupancy, further hindering the flow of new samples into the decoder on the receive chain.

To further examine the decoder bottleneck and its impact on real-time processing, we conducted a comparative analysis between our custom Viterbi decoder and a standard optimized implementation. The standard Viterbi decoder performs windowed traceback, allowing decoded bits to be output in smaller steps instead of waiting for the entire packet. Due to time constraints, we conducted this evaluation using the legacy packet format on MCS 0-4, as the VHT format introduces additional configuration parameters that require further development and validation to implement correctly. We profiled both the demod and decode blocks under the same experimental conditions and 300 bytes payload size, collecting latency statistics across three independent runs.

In Fig. 3, the average decoding latency is higher than demodulation, and demodulation has high jitter across all MCS indices. This behavior reflects the previous decoder implementation, which performed traceback only after receiving the full packet, introducing a delay.

In contrast, Fig. 4 shows a reduction in both decode and total latency. The decoder has been optimized to reduce the traceback delay, allowing earlier output and minimizing buffer occupancy time. As a result, the overall pipeline latency is improved, and backpressure on the demod block is alleviated.

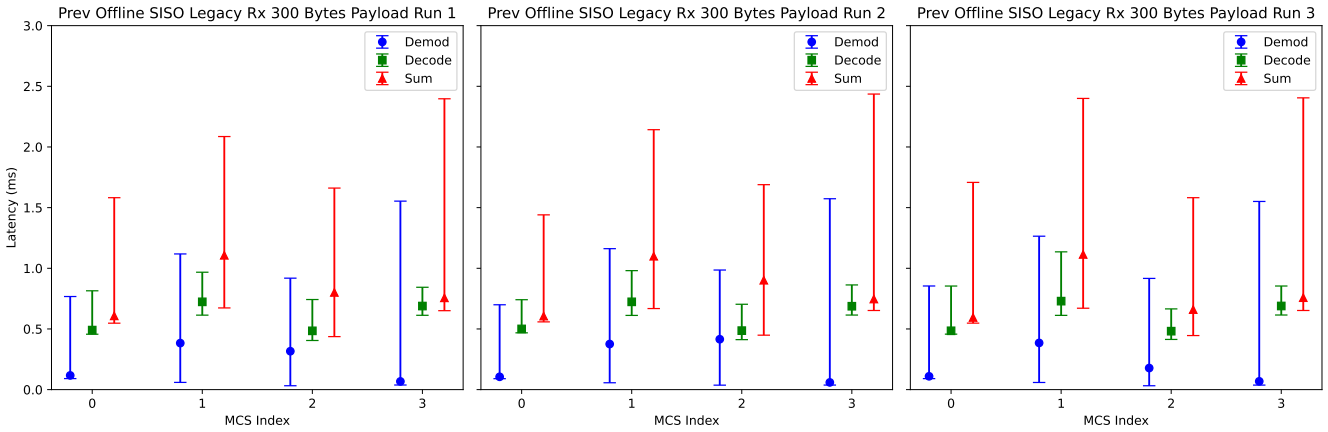

Fig. 3. Timing measurement on our customer Viterbi.

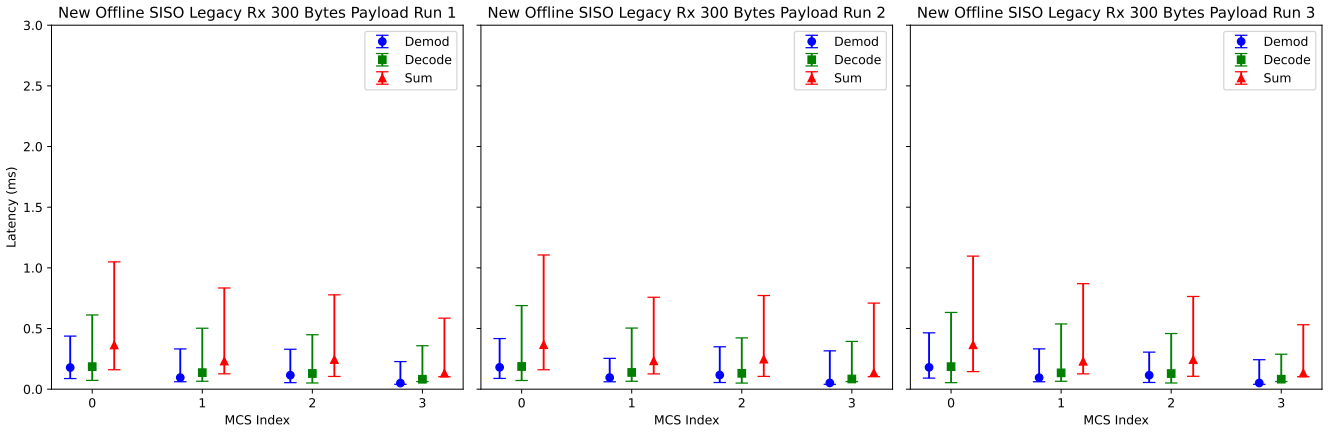

Fig. 4. Timing measurement on standard Viterbi.

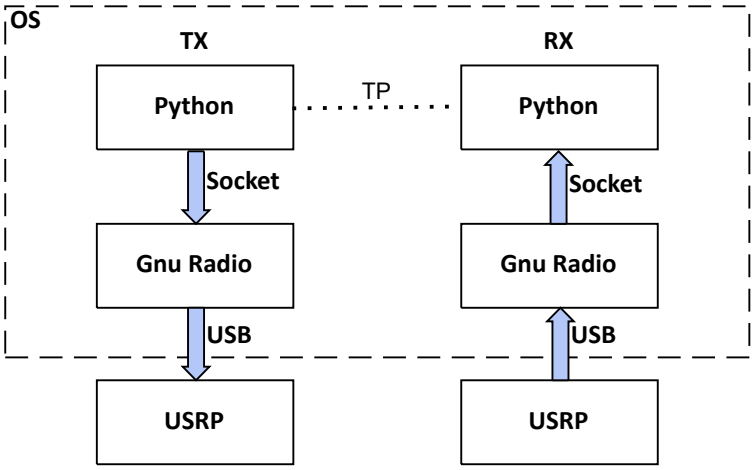

Fig. 5. The system stack of GR-WiFi.

2. RTT Timing Measurement without USRP

In RTT measurements, the focus is on the total end-to-end round-trip time, represented by the Time Pair (TP) illustrated in Fig. 5. A timestamp is taken and packed into the payload in Python before the packet is sent to GNU Radio via a socket interface. Within GNU Radio, the transmitter output passes through a channel model and is routed to the receiver. After the receiver decodes the packet, the decoded payload is forwarded back to Python through another socket. A second timestamp is recorded in Python immediately after the decoded packet is received. The RTT is computed as the difference between the reception and transmission timestamps, representing the total processing delay across the transmit and receive chains.

In addition to RTT, we also recorded block-level latency for our customer block. in transmitter, we have pktgen, encode, mod and pad blocks. In receiver, there are demod and decode blocks. Two payload sizes, 30 bytes and 300 bytes, are tested in 2 independent runs. For each MCS index, 10,000 packets were sent with 1 ms gap. The results in Fig. 6, Fig. 7, Fig. 8, and Fig. 9 report the minimum, maximum, and average processing time per packet in each block during the RTT measurement.

Comparing result on 30-byte and 300-byte payloads, the round-trip time (RTT) increases substantially for the larger payload. This is expected as more bits need to be processed by each block, especially in the decode stage. In all cases, the decode block exhibits the highest latency among individual blocks, particularly for 300-byte packets. This reinforces earlier findings that the Viterbi decoder implementation contributes significantly to processing delays.

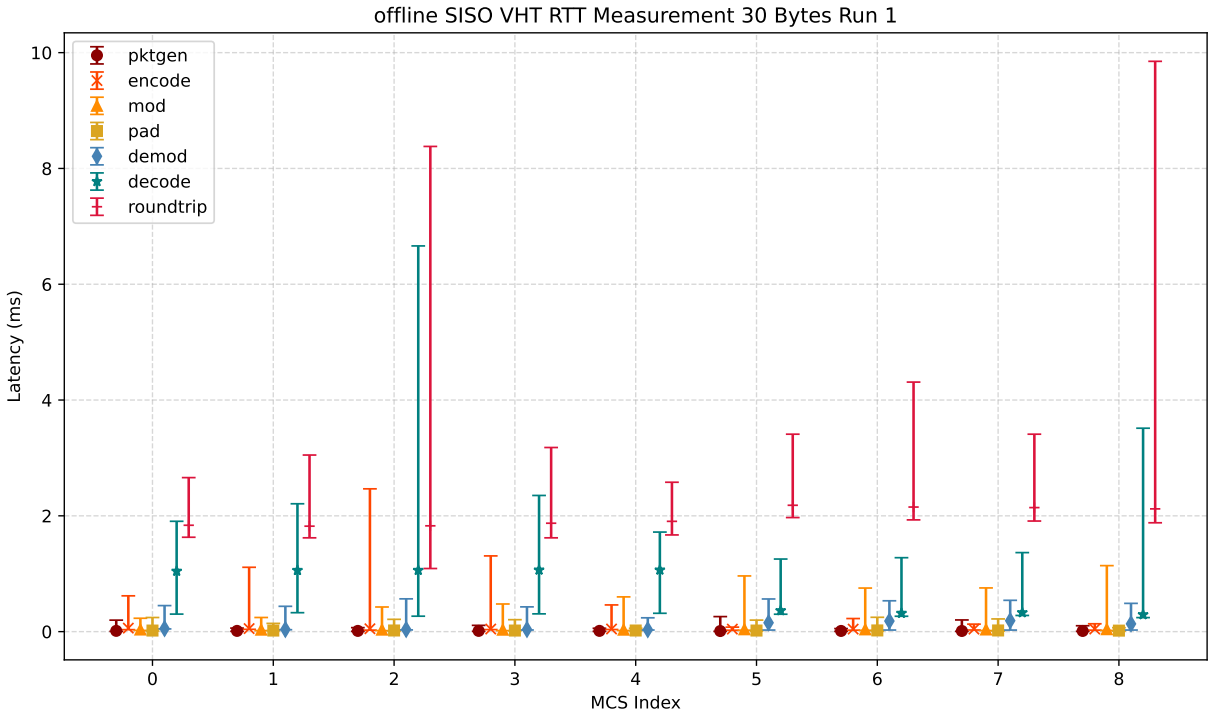

Fig. 6. RTT measurement on 30 bytes payload run 1 with no USRP.

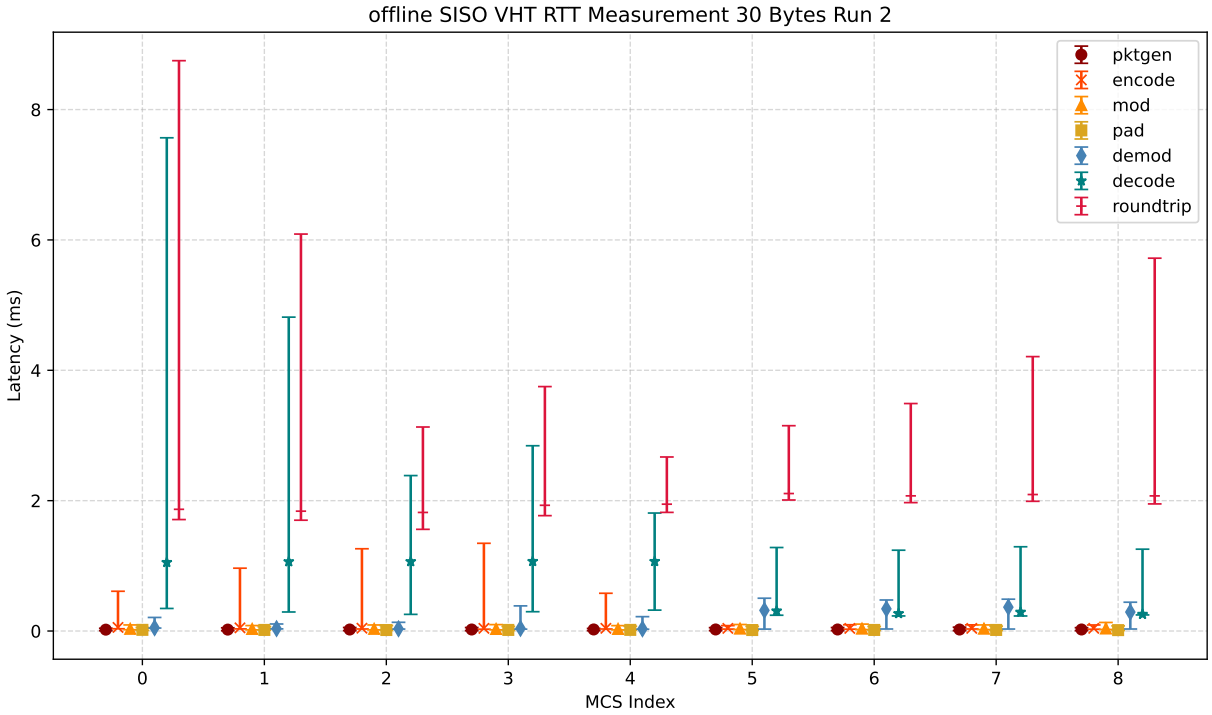

Fig. 7. RTT measurement on 30 bytes payload run 2 with no USRP

3. RRT Timing Measurement with USRP

This RTT measurement follows the same methodology as in the previous measurement, except that a USRP device is involved in the transmit and receive chain. Specifically, the transmitter and receiver are implemented in GNU Radio and connected via a coaxial loopback through a physical USRP. As before, a timestamp is inserted into the packet payload in Python before transmission, and another timestamp is recorded upon reception after decoding. The RTT is computed as the time difference between these two events.

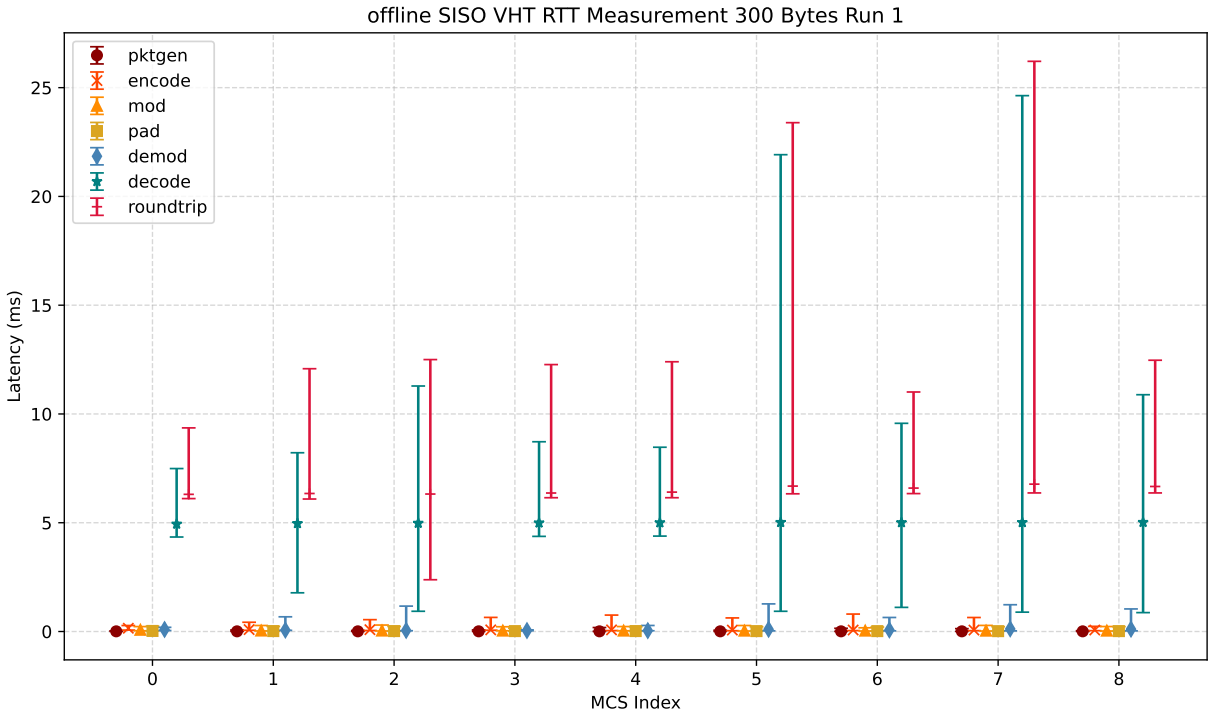

Fig. 8. RTT measurement on 300 bytes payload run 1 with no USRP

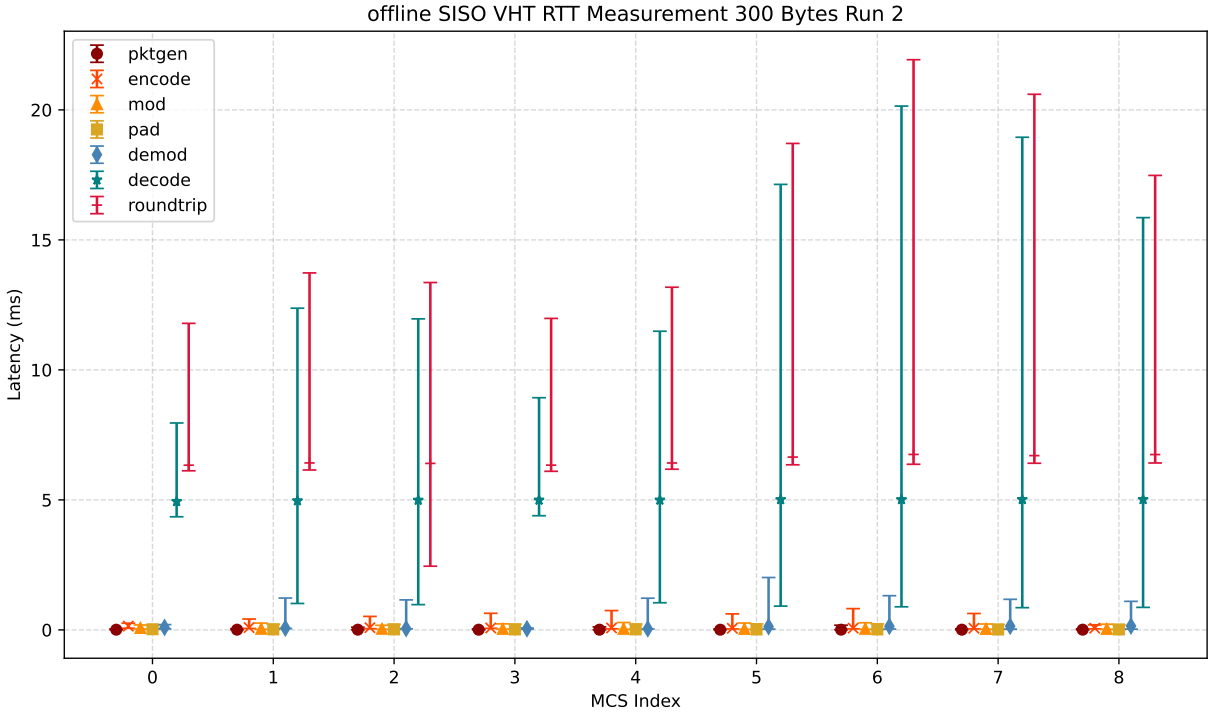

Fig. 9. RTT measurement on 300 bytes payload run 2 with no USRP

The measurement results are shown in Fig. 10, which presents the RTT across three independent runs using 30-byte VHT-format packets and MCS 0 through 8. A total of 10,000 packets are transmitted in each run, with a 1 ms packet gap. From the result, we observe the high latency and jitter when USRP is involved. While this may be influenced by factors such as hardware buffering, USB transfer latency, and asynchronous interaction between the UHD driver and GNU Radio scheduler, the exact cause remains unclear and needs further investigation. We believe that blocks in the GNU Radio receive chain may be unable to process incoming samples quickly enough, leading to buffer

congestion.

To further investigate this issue, we conducted another round of testing, in which we reduced the number of transmitted packets and increased the packet gap. This allowed more time for the receiver to process and consume samples without blocking buffers between blocks. In this round of testing, 1000 packets were transmitted with packet gaps of 1 ms, 10 ms, and 100 ms across all MCS indices, to evaluate the effect of packet pacing on jitter in our testbed.

Figure 11 shows that reducing the number of transmitted packets and increasing the packet gap influences jitter behavior, especially in lower MCS. Here are some key observations derived from the test:

- Increasing the inter-packet gap to 10 ms or 100 ms and reducing the number of transmitted packets gives the receiver more time to process incoming samples, effectively reducing RTT jitter, particularly at lower MCS values (e.g., MCS 0 to 4). Among the tested values, a 10 ms gap offers the best trade-off, minimizing RTT variation across most MCS indices.

- High jitter persists at high MCS values: At higher MCS rates (MCS 5 to MCS 8), jitter remains large regardless of packet spacing. This is particularly noticeable when the packet occupies fewer than five OFDM symbols. This observation points out that our implementation has room to improve.

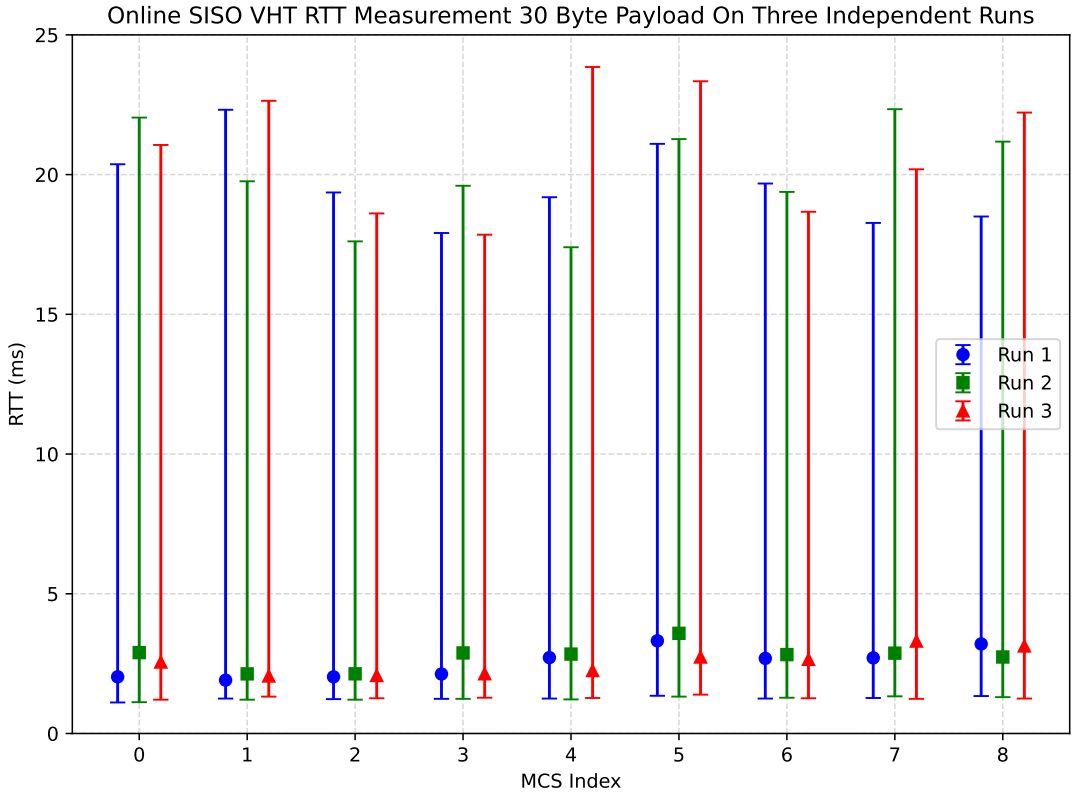

Fig. 10. Three run RTT measurement on 30 bytes payload with 10,000 packets sent.

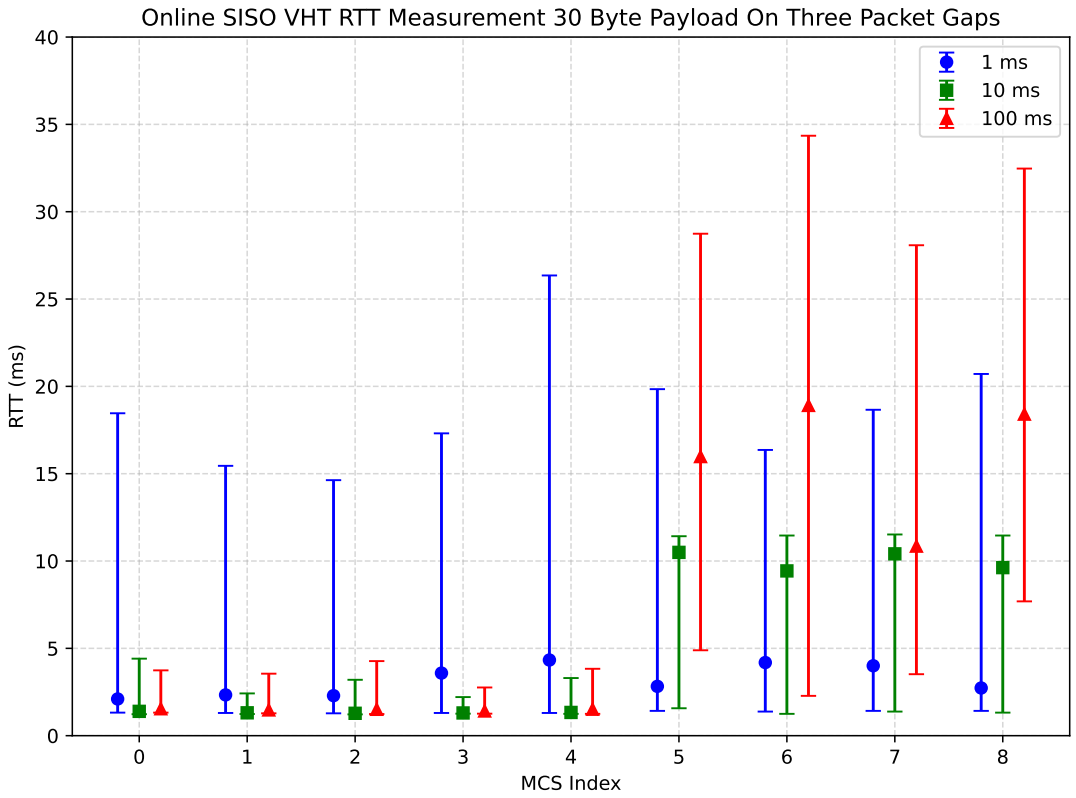

Fig. 11. Three packet gaps RTT measurement on 30 bytes payload with 1000 packets sent.