

515 latency.pcapng

File

Can you review this PCAP and look at the client f4:ab:5c:fb:52:60 and see why occasionally it's ping to the gateway (10.1.140.1) and to 10.0.82.4 spike to 100ms or much higher and other times they are at 22ms (which is what we expect roughly)? Is the client picking too high of a modulation rate and then having to modulate back down?

Thought for 14m 10s

I dug into /mnt/data/515 latency.pcapng and focused on f4:ab:5c:fb:52:60.

What I can / can't see in *this* capture

Most of the actual client data frames are **802.11 “Protected” (encrypted)**, so I **can't directly read IP/ICMP** inside the frames to positively match “ping to 10.1.140.1 / 10.0.82.4” and compute RTT from the capture alone.

What I *can* see very clearly is the **Wi-Fi MAC behavior** (power-save signaling, multicast/broadcast behavior, beacon/DTIM timing, RSSI, etc.)—and that's enough to explain the “~22ms normally, sometimes ~100ms+” symptom.

Key findings (and why they match 100ms+ spikes)

1) Beacon interval + DTIM strongly line up with ~100ms / ~300ms latency jumps

Your BSSID for this client is **a2:2a:6f:d8:7f:6a** (SSID shown in beacons as “**Network8**”).

The beacons advertise:

- **Beacon interval: 100 TU \approx 102.4 ms**
- **DTIM period: 3 \rightarrow DTIM every ~307.2 ms**

That matters because **sleepy clients + DTIM buffering** is one of the most common reasons you see latency “snap” between a low baseline and **~100ms / ~200ms / ~300ms** style spikes.

2) The client is *actively using power save (PM bit)* in a DTIM-ish cadence

The client sends a lot of **QoS Null** frames (subtype 12) to the AP—this is classic WMM power-save/U-APSD behavior.

- Many of those QoS Null frames have **PM=1** (client indicating it's dozing).
- The **time between PM=1 indications** has a **median ~0.323s**, which is very close to the **DTIM=3** cadence (~0.307s).

Translation: this station looks like it's spending time in a doze/wake rhythm that aligns with DTIM. When a ping packet (request *or* reply) arrives while the client is dozing, it can sit buffered until the next wake opportunity → you get those **“random” 100ms+ RTT spikes**.

This explanation fits your symptom pattern *much better* than “picked too high an MCS then backed off,” because rate-scaling problems usually show up as **retries** and **lots of retransmitted unicast**.

3) I don't see retries from this client in the capture

For frames involving **f4:ab:5c:fb:52:60**, the **Retry bit isn't set** on its transmissions in this capture. Also, RSSI on the client's control frames is **extremely strong** (often around **-25 dBm**), which further argues against “too aggressive a modulation rate causing lots of retries.”

So: **nothing here screams “rate too high → falls back → ping spikes.”**

4) This client is generating a lot of multicast/broadcast that gets sent at low basic rates

This client shows up as the **source** of a bunch of group traffic that the AP transmits over the air as **multicast/broadcast** (encrypted with the group key):

- Destinations include **ff:ff:ff:ff:ff:ff**, **33:33:00:00:00:01** (IPv6 all-nodes), and **01:00:5e:...** (IPv4 multicast like SSDP/mDNS MACs)
- These are sent at **6 Mbps** (basic rate), and they tend to appear in **bursts** whose spacing aligns with **~307ms** DTIM timing.

Even if this isn't the *primary* reason for ping spikes, it **adds airtime/queue pressure** and makes DTIM-related “bursty delivery” more noticeable.

Bottom line

Based on this PCAP, the most likely reason for the ~100ms+ ping spikes is power-save/DTIM buffering (sleep/wake cadence), not “too-high modulation rate and then backing down.”

What I'd change / test next (most effective first)

A) If you want *consistently low latency* on that SSID

- **Set DTIM to 1** on that SSID (UniFi Wi-Fi settings).
Tradeoff: more client battery use, but noticeably less “DTIM-jitter.”
- If available in your UniFi version: **disable U-APSD / WMM Power Save** on that SSID (or test with it off).
- On the client (if Windows): NIC advanced settings + power plan
 - Disable “**Allow the computer to turn off this device to save power**”
 - Look for **U-APSD / WMM Power Save / Power Save Mode** and set to off / CAM / maximum performance (names vary by chipset).

B) Reduce multicast/broadcast airtime impact

- Turn on **Multicast to Unicast / Multicast Enhancement** (UniFi option names vary).
- Use **IGMP snooping** (switch + UniFi if applicable).
- Consider a higher **minimum data rate** on 5 GHz (common: **12 Mbps** or **18 Mbps**) if coverage allows—this reduces the airtime cost of unavoidable broadcast/multicast.

C) If you want me to prove it against the ping packets specifically

You'd need either:

- a capture taken **after decryption** (some APs/support tools can export decrypted captures), or
- a **wired capture** on the gateway / switchport where ICMP is visible, so I can directly line up ICMP echo request/reply timing with the Wi-Fi doze/DTIM timing.

If you tell me what this client actually is (Windows laptop? phone? IoT?), I can give you the most likely exact knob to change first (because phones/tablets doing DTIM sleep is very common, whereas laptops on “max performance” usually aren't this DTIM-synchronous).

What does DTIM 1, 3, etc mean?

DTIM is a Wi-Fi timing setting that mostly affects **how long “sleepy” clients can stay asleep before they have to wake up to receive broadcast/multicast (and sometimes buffered) traffic.**

The pieces

- **Beacon interval:** how often the AP sends a beacon (usually **100 ms** by default).
- **DTIM period (DTIM 1, 2, 3...):** **how many beacons occur between DTIM beacons.**

A **DTIM beacon** is a special beacon that tells power-saving clients:

“Wake up now—there may be buffered broadcast/multicast traffic for you.”

What DTIM numbers mean (with a common 100 ms beacon interval)

If your beacon interval is ~100 ms:

- **DTIM 1:** DTIM every 1 beacon → about **every 100 ms**
- **DTIM 2:** DTIM every 2 beacons → about **every 200 ms**
- **DTIM 3:** DTIM every 3 beacons → about **every 300 ms**

So **higher DTIM = clients can sleep longer** (better battery), but **worst-case latency can be higher** for anything that gets buffered until the next DTIM.

Why it affects ping/latency

If a client is in power-save mode and the AP buffers a frame while it’s asleep, delivery may wait until:

- the client wakes, or
- the next DTIM opportunity (depending on the traffic type and the client/AP behavior)

That can show up exactly like you described: **~20–30 ms normally**, but sometimes **~100 ms / ~200 ms / ~300 ms** spikes.

Quick rule of thumb

- **DTIM 1**: best for **latency** (VoIP, gaming, real-time apps), worse for battery
- **DTIM 3** (or higher): better for **battery**, more “jittery” latency for some clients

If you tell me your beacon interval (UniFi typically shows it) and what kind of device that client is, I can tell you whether DTIM 1 is likely to help and what tradeoffs you’ll notice.

The issue actually largely went away when we turned U-APSD on as it was off before - could you explain why that may be?

Thought for 8s

Yep — that outcome actually makes a lot of sense.

U-APSD in plain English

U-APSD (Unscheduled Automatic Power Save Delivery) is part of **WMM power save**. It changes *how a sleeping client gets its buffered downlink packets* from the AP.

Instead of the AP waiting for a fixed beacon/DTIM cadence (or the client doing old-school PS-Poll), **the client can “trigger” the AP to immediately dump buffered unicast frames** when it wakes up briefly and sends *any* uplink frame (often a QoS Null or a normal data frame). The AP then delivers packets in a short “service period.”

Why turning it ON can reduce ping spikes

Your ping RTT spikes (~22 ms normally, occasionally 100 ms+) are exactly what you see when **the ping reply (or other downlink packet) gets stuck in the AP’s buffer while the client is dozing**.

When U-APSD was **OFF**, many clients still doze (especially phones/tablets and some laptop chipsets), but the delivery path can be worse:

- The client dozes → AP buffers downlink unicast
- Delivery may wait until the client fully wakes (which might align with beacon/DTIM timing), or until legacy mechanisms kick in
- Result: **RTT jumps by ~1 beacon interval (≈100 ms) or multiples**

When you turned U-APSD **ON**, the client’s behavior becomes:

- Client sends the ping request (uplink) or a QoS Null → **this acts as a trigger**
- AP immediately releases buffered downlink frames during the service period
- Result: **much fewer “waiting for the next wake/DTIM” moments**, so RTT stabilizes closer to your baseline

The “counterintuitive” part

People often think “power save features = more latency,” but **U-APSD is actually designed to make power-save *more efficient and often lower-latency*** for unicast, because it replaces “wait for timing windows” with “deliver on trigger.”

Why it can be especially noticeable on ping

Ping is **tiny and periodic**. If the client is dozing, it’s easy for one ping reply to land in a buffer at the wrong moment and wait ~100 ms. With U-APSD on, each ping request can “pull” the reply through quickly.

A quick mental model

- **DTIM** mostly affects **broadcast/multicast** delivery timing.
- **U-APSD** mostly helps **unicast** delivery timing while clients are power-saving.

So if your spikes were mostly unicast (ping replies), **U-APSD is the more direct fix** than changing DTIM.

If you tell me the client type (iPhone/Android/Windows laptop + Wi-Fi chipset), I can tell you how likely it is that it was doing “WMM PS triggers but AP wasn’t honoring them” when U-APSD was off — that’s a common mismatch that produces exactly the kind of jitter you saw.

Some of the spikes we saw were actually up to 900ms. The device is an RF scanner with a Quectel chip in it and it would have low latency for 9 seconds and then the latency would rise for 1 second and then go back down to ~25ms for 9 seconds and then back and forth exactly.

Thought for 7m 9s

That 9-seconds-good / 1-second-bad “metronome” pattern is a huge clue: it almost always points to a **periodic client-side behavior** (power-save or background scanning), not random RF rate adaptation.

First: why 900 ms specifically is a “Wi-Fi timing number”

In your capture, the AP on that BSSID is advertising:

- **Beacon interval = 100 TU \approx 102.4 ms**
- **DTIM period = 3 \rightarrow DTIM every \sim 307.2 ms**

A **\sim 900 ms** spike is *very* close to either:

- **9 beacon intervals:** $9 \times 102.4 \text{ ms} \approx$ **921.6 ms**, or
- **3 DTIM intervals:** $3 \times 307.2 \text{ ms} \approx$ **921.6 ms**

So the “ \sim 900 ms” spikes are consistent with “packet waited until the next (or next-next) wake opportunity” timing, not with a modulation/rate issue.

What I see in the PCAP that supports this

This Quectel client is sending a lot of **QoS Null frames** and it **toggles the Power Management (PM) bit**—classic “I’m going to sleep / I’m awake” signaling.

- The cadence of the “PM=1” frames is around **\sim 0.32s**, which is very close to the **DTIM=3 interval (\sim 0.307s)**.
- There are also a couple instances where it stays in PM=1 for about **\sim 1.1–1.25 seconds**, which is right in the range that can create \sim 900ms-class RTT outliers if a ping happens to get queued during that doze/off-channel window.

Why enabling U-APSD made it better

When **U-APSD was OFF**, a power-saving client often ends up with worse “buffer-release” behavior:

- Downlink frames (like ping replies) can sit **buffered at the AP** until the next wake/DTIM moment.
- Uplink frames (like the ping request) can also be **queued inside the client’s Wi-Fi firmware/driver** if it’s dozing or off-channel.

When **U-APSD is ON**, the client can **trigger delivery immediately** (often via QoS Null / uplink frames), so it doesn't have to "wait for the timing window" as much. That typically **reduces those quantized 100/300/900ms spikes** dramatically.

Why the exact 9s / 1s pattern happens

That kind of clockwork pattern usually comes from one of these:

1. **Periodic background scan / roam scan**

Many handheld/industrial Wi-Fi stacks do a scan on a fixed timer (often ~10s). While scanning **off-channel**, they can't TX/RX on the serving channel, so packets queue and RTT spikes.

2. **A periodic deeper power-save cycle**

Some embedded chipsets briefly enter a deeper doze state on a fixed schedule. One ping per cycle (or a short window) gets delayed ~1s.

Either one matches "good for 9 seconds, then one bad second, repeat" far better than "MCS too high."

Quick ways to confirm which one it is

- If you run ping **at 10 pings/sec** (0.1s interval):
 - **Scan/off-channel** often shows a *cluster* of bad RTTs during the bad second (sometimes even a timeout).
 - **One delayed packet** (power-save queueing) often shows **one or two** outliers, not the whole second.
- On the UniFi side, check whether the client is showing **roam events** or frequent "connectivity check" behavior during that "bad" beat.

What to try (in order)

1. **Keep U-APSD ON** (you already saw improvement — good sign you're on the right track).
2. If that SSID is latency-sensitive: set **DTIM = 1** (reduces worst-case buffering delays).
3. On the Quectel/handheld side (best fix if available in its config/SDK):
 - Disable or reduce **Wi-Fi power save** (force CAM / "performance" mode)
 - Reduce/disable **background scan / roaming aggressiveness**

4. Consider enabling **802.11k/v** (neighbor reports / BSS transition) *if the scanner supports it* — it can reduce the need for blind scanning.

If you want, tell me:

- ping interval you were using (1s default?), and
- whether the “bad second” had **timeouts** or just high RTT, and I’ll tell you which of the two (scan vs doze) is more likely and the most targeted knob to change first for that Quectel scanner.

Outside of the ICMP test - do you think this change also improved performance on an HTTPS application running on the RF device?

Thought for 5s

Yes — **very plausibly**, and for the same underlying reason.

Even though ICMP is just a diagnostic, the *thing it was revealing* (periodic ~100–900ms stalls from doze/queueing/off-channel behavior) also hurts HTTPS, because HTTPS is built on **TCP**, and TCP is much more sensitive to **jitter + pauses** than ping is.

How the old behavior would hurt HTTPS

If the RF device periodically stops effectively receiving/transmitting on the channel (power-save buffering or off-channel scanning), TCP sees that as:

- **ACKs stop arriving for a bit** → sender can’t advance the window
- **Packets bunch up / head-of-line blocking** (one missing segment can stall everything behind it)
- **Retransmission timers** can fire (common minimums are ~200ms–1s depending on stack), which is exactly how you end up with “random” 1-second-ish stalls in app traffic
- For short transactions, **TLS handshakes** and HTTP request/response flows need multiple round trips; a single 300–900ms pause can blow up total request time

So the same “9 seconds fine / 1 second bad” rhythm that shows up in ping typically shows up in HTTPS as **periodic slow requests**, occasional retries, or perceived “hangs.”

Why U-APSD ON can help HTTPS

With U-APSD enabled, the client can more reliably **trigger immediate delivery of buffered unicast frames** (TCP data/ACKs are unicast), instead of waiting for the next timing opportunity. That tends to reduce:

- buffered-downlink delays,
- missed/misaligned wakeups,
- and the chance of TCP hitting a retransmission timeout.

So it's completely reasonable that HTTPS “felt better” after the change.

One important caveat

If the 1-second bad period is caused mostly by **active scanning/off-channel roam scans**, U-APSD won't eliminate it completely — but it can still reduce how often the scan translates into TCP RTOs by making the “normal” delivery tighter.

Best way to confirm (quick + practical)

Look for any of these before vs after:

- **TCP retransmissions / dup ACK bursts** during the “bad second” (on a wired capture at the gateway is easiest)
- **TLS handshake time** / “time to first byte” (even just running curl with timing stats if you can on the device)
- **Application response time histogram**: you'll often see the long tail collapse when these stalls go away

If you can grab a short **wired capture** on the gateway while the RF device runs the HTTPS workflow, I can tell you definitively whether the improvement is from fewer retransmissions/RTOs vs just better queuing/latency.

It seems like there are a lot less QoS Null packets in the captures now - does that make sense?

Yes, that **does** make sense — and it’s actually a pretty strong hint that the client’s power-save behavior changed.

What QoS Null frames are usually doing

QoS Null frames are commonly used as **WMM power-save / U-APSD signaling**:

- “I’m going to sleep / I’m awake” (PM bit toggles)
- “Trigger” frames to pull buffered downlink traffic from the AP (U-APSD trigger)
- Keepalive-ish “I’m still associated” behavior on some chipsets

So you tend to see **lots of QoS Null** when the client is:

- frequently **entering/leaving doze**, or
- doing short “wake, trigger, sleep” service periods, especially when there isn’t much real uplink data.

Why you might see fewer QoS Null after turning U-APSD ON

A few common reasons:

1. Real data frames replace the triggers

If the RF scanner now has a steady trickle of uplink traffic (TCP ACKs, app chatter), it may not need separate QoS Null triggers — normal QoS data can act as the trigger / keep the device awake long enough.

2. Client may be staying awake more

Some clients change their power-save strategy when U-APSD is available (or when the AP advertises it). They may:

- spend less time dozing, or
- use longer service periods, which reduces the “toggle chatter” (fewer QoS Null).

3. The chipset/firmware may have switched from legacy PS behavior

With U-APSD off, some devices “fake it” with more frequent QoS Null/PS-style probing. With U-APSD on, they behave more cleanly (fewer management-ish frames).

The one “gotcha”

In some setups, enabling U-APSD can also reduce the need for the station to *announce* doze transitions as frequently because buffered delivery is more reliable. That also shows up as fewer QoS Null frames.

What I'd check to be sure

In the “before” capture vs “after” capture, look at:

- PM bit usage (are there fewer PM=1 frames?)
- Any **gaps** where the client is silent/off-channel
- TCP retransmissions / dup ACK bursts (ideally from a wired capture)

If you want, upload one of the “after” PCAPs and I'll quantify it (QoS Null rate per minute, PM-bit toggles, and whether the remaining latency spikes line up with DTIM/beacon timing).

Can you tell by the packet capture if the client was trying to change modulations or if it's truly just about the U-APSD feature and a compatibility issue essentially.

Thought for 5m 9s

Yes — **you can usually tell from a PCAP**, but in *this* capture there's an important limitation:

What the capture can show about modulation/rate changes

To prove “client picked too high of a modulation rate then backed off,” you typically look for one (or more) of:

- **Radiotap PHY info** showing **rate/MCS/VHT/HE MCS** changing over time
- **802.11 retransmissions** (Retry bit set) and/or **duplicate sequence numbers**
- Bursts of **BlockAck / reordering / retransmission patterns** coincident with the latency spikes

What I actually see for f4:ab:5c:fb:52:60 in this PCAP

1) No usable MCS/VHT/HE info is present

The radiotap headers in this file **do not include HT/VHT/HE MCS fields** for this station's frames — only the legacy “rate” field. So I **cannot** directly chart MCS up/down for that client from this specific capture.

2) For the frames we *do* see from the client, there's no sign of “rate too high → retries”

For the client's own transmissions that are captured (mostly **QoS Null** frames):

- The over-the-air “rate” shown is **constant** (no swings in the radiotap rate field for these frames).
- **Retry bit is never set** on the captured data frames for this client.
- **Sequence numbers never repeat** (i.e., no retransmission pattern hiding behind missing Retry bits).

If modulation/rate selection were the problem, you'd normally see **retries and repeated sequence numbers** (especially with big latency spikes).

Why this still strongly points to U-APSD / power-save compatibility, not modulation

- Your “spikes up to ~900ms” line up extremely well with **DTIM/beacon timing math**:
 - Beacon interval in the capture is **100 TU (~102.4ms)**
 - DTIM period is **3 → ~307.2ms**
 - $\sim 900\text{ms} \approx 3 \times \text{DTIM}$ (or $9 \times \text{beacons}$) $\approx 921.6\text{ms}$
- The client is heavily doing **QoS Null / PM toggling** behavior (power-save signaling), which is exactly the kind of thing that produces “quantized” latency spikes at ~100/300/900ms when buffering/wake timing isn't lining up.
- The fact that turning **U-APSD ON** largely fixed it also matches: U-APSD makes buffered **unicast** delivery (TCP/ICMP replies) happen on trigger/service-periods rather than waiting for the next timing window.

Bottom line

From this PCAP:

- I **can't directly confirm MCS up/down** because the capture doesn't include the necessary radiotap MCS/VHT/HE fields.
- But I **can** say there's **no MAC-layer evidence** supporting “modulation too high → fallback” (no retries, no repeated sequence numbers), and there *is* strong timing +

behavior evidence consistent with **power-save / DTIM buffering / U-APSD interaction.**

If you want a capture that *does* definitively answer the modulation question, the key is a **monitor-mode sniffer capture** that includes radiotap **MCS/VHT/HE** fields (or a capture method that exports those PHY fields). Then we can directly correlate “MCS drops + retries” with the 1-second bad windows.