



Tech Bulletin: Legacy Vertex on a RingCentral (TCP) Site

What “OPUS” really is, where the caller ID actually lives, and why “it works sometimes” is not a working solution

Internal Tech Bulletin — 24JUN26 · Source: customer Wireshark capture
(6,916 packets)

21-240409-12-24_63.cap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 5061 || tcp.port == 5060 || udp.port == 5061 || udp.port == 5060 || rtp || sip

No.	Time	Source	Destination	Protocol	Length	Info
46	6.006314	10.1.10.147	104.245.57.103	SIP/SDP	1130	Request: INVITE sip:16783610036@sip.ringcentral.com:5060
49	6.006359	104.245.57.103	10.1.10.147	SIP	369	Status: 100 Trying
50	6.006431	104.245.57.103	10.1.10.147	SIP	613	Status: 407 Proxy Authentication Required
52	6.006447	10.1.10.147	104.245.57.103	SIP	424	Request: ACK sip:16783610036@sip.ringcentral.com:5060
54	6.006474	10.1.10.147	104.245.57.103	SIP/SDP	1349	Request: INVITE sip:16783610036@sip.ringcentral.com:5060
57	6.006508	104.245.57.103	10.1.10.147	SIP	358	Status: 100 Trying
60	7.006693	104.245.57.103	10.1.10.147	SIP/SDP	964	Status: 183 Session Progress
63	7.006770	10.1.10.147	208.87.40.213	OPUS	102	PT=OPUS, SSRC=0x669C0073, Seq=12797, Time=1920
64	7.006794	10.1.10.147	208.87.40.213	OPUS	81	PT=OPUS, SSRC=0x669C0073, Seq=12798, Time=2880
65	7.006810	10.1.10.147	208.87.40.213	OPUS	93	PT=OPUS, SSRC=0x669C0073, Seq=12799, Time=3840
67	7.006834	10.1.10.147	208.87.40.213	OPUS	88	PT=OPUS, SSRC=0x669C0073, Seq=12800, Time=4800
69	7.006850	10.1.10.147	208.87.40.213	OPUS	100	PT=OPUS, SSRC=0x669C0073, Seq=12801, Time=5760
71	7.006874	10.1.10.147	208.87.40.213	OPUS	99	PT=OPUS, SSRC=0x669C0073, Seq=12802, Time=6720
73	7.006890	10.1.10.147	208.87.40.213	OPUS	103	PT=OPUS, SSRC=0x669C0073, Seq=12803, Time=7680
75	7.006915	10.1.10.147	208.87.40.213	OPUS	102	PT=OPUS, SSRC=0x669C0073, Seq=12804, Time=8640
77	7.006930	10.1.10.147	208.87.40.213	OPUS	95	PT=OPUS, SSRC=0x669C0073, Seq=12805, Time=9600
79	7.006954	10.1.10.147	208.87.40.213	OPUS	91	PT=OPUS, SSRC=0x669C0073, Seq=12806, Time=10560
81	7.006970	10.1.10.147	208.87.40.213	OPUS	103	PT=OPUS, SSRC=0x669C0073, Seq=12807, Time=11520
83	7.006994	10.1.10.147	208.87.40.213	OPUS	96	PT=OPUS, SSRC=0x669C0073, Seq=12808, Time=12480
85	7.007010	10.1.10.147	208.87.40.213	OPUS	101	PT=OPUS, SSRC=0x669C0073, Seq=12809, Time=13440
87	7.007034	10.1.10.147	208.87.40.213	OPUS	103	PT=OPUS, SSRC=0x669C0073, Seq=12810, Time=14400
89	7.007050	10.1.10.147	208.87.40.213	OPUS	101	PT=OPUS, SSRC=0x669C0073, Seq=12811, Time=15360
91	7.007078	10.1.10.147	208.87.40.213	OPUS	97	PT=OPUS, SSRC=0x669C0073, Seq=12812, Time=16320

> Frame 46: Packet, 1130 bytes on wire (9040 bits), 1130 bytes captured (9040 bits)
> Ethernet II, Src: YealinkNetwo_9a:ce:23 (80:5e:0c:9a:ce:23), Dst: VantivaUSA_81:54:19 (3c:82:c0:81:54:19)
> Internet Protocol Version 4, Src: 10.1.10.147, Dst: 104.245.57.103
> Transmission Control Protocol, Src Port: 12466, Dst Port: 5090, Seq: 1, Ack: 1, Len: 1076
> Session Initiation Protocol (INVITE)

The customer's own capture, with our SIP filter applied. The green bar is that SIP filter — `tcp.port == 5061 || tcp.port == 5060 || udp.port == 5061 || udp.port == 5060 || rtp || sip`. Look at the highlighted call setup (frame 46): it is an INVITE to `sip:...@sip.ringcentral.com`, and the bottom detail pane shows it riding **TCP, Dst Port 5090** — not UDP. Every row labeled **OPUS** beneath it is the voice audio, which carries no caller ID. This one screen is why the rest of this bulletin is about UDP vs. TCP, and about “OPUS.”

Team — I went through this customer's raw Wireshark file packet-by-packet, not just the screenshot above. Three things are now nailed down. **(1)** The “OPUS” everyone is asking about is **not a protocol we read** — it is the voice *audio codec*, the actual sound of the call. There is never any caller ID inside it, so there is nothing there for us to capture. **(2)** The caller ID lives in the call *signaling* (SIP), it is in **plain text**, and it is **not encrypted** — the data is right there. **(3)** On this site, **100% of that signaling is delivered over TCP** — there is **zero UDP**. That last point is the whole

ballgame: our legacy Vertex is built to read **UDP** SIP. On a TCP-only site like this one, **you cannot rely on it for caller ID**. If it pops a name now and then, that is not the box working reliably — TCP-delivered signaling is something the legacy hardware cannot follow dependably, so it reads some setups and misses others with no way to predict which. We do not sell or support “sometimes.”

1. The question that started this: “What is OPUS, and can we read it?”

Short answer: **OPUS is an audio codec, not a phone or network protocol — and it has nothing to do with caller ID**. It is the format the actual voice is compressed into so it can travel across the internet. We never read the audio, so there is nothing for us to do with OPUS. The reason it shows up so loudly in Wireshark is simply that the voice is the bulk of any call: **5,313** of the packets in this file are that audio stream.

2. What this capture proves (the hard data)

Finding 1 — The carrier is RingCentral, and every call setup is on TCP.

The phones register to `sip.ringcentral.com` and every SIP message in the file rides **TCP, port 5090**, between the phone on the LAN (`10.1.10.147`) and RingCentral’s cloud (`104.245.57.103`). I counted **27 SIP messages — all TCP, none on UDP, none on the old standard port**. This particular file is mostly one outbound call plus the phone re-registering (5 distinct calls/registrations), which is plenty to prove the transport.

Finding 2 — The caller ID is in plain text. It is not encrypted.

I read every SIP message as clear text — there were **zero** encrypted (TLS) SIP packets. If RingCentral were encrypting the signaling, Wireshark could only ever print `Application Data` ; it could never show you the words `INVITE` , `REGISTER` , or `200 OK` . It shows them, so the data is readable. Here is a real call setup from the file, with the customer’s own numbers partly masked, showing both where the caller identity sits *and* where OPUS shows up:

```
# Call setup (INVITE) pulled straight from the capture – delivered over TCP/5090
INVITE sip:1678361####@sip.ringcentral.com:5060 SIP/2.0
From: "POS Extension" <sip:1720912####@sip.ringcentral.com:5060> ← caller identity
lives here
To: <sip:1678361####@sip.ringcentral.com:5060>
Contact: <sip:1720912####@10.1.10.147:12466;transport=TCP>
... inside the same message, the audio offer (SDP):
m=audio 12664 RTP/AVP 111 0 18 101
a=rtpmap:111 opus/48000/2 ← OPUS = the voice audio (no caller ID)
a=rtpmap:0 PCMU/8000 ← other audio codec choices
a=rtpmap:18 G729/8000
a=rtpmap:101 telephone-event/8000 ← touch-tone (DTMF) digits
```

Note this example is an **outbound** call (the “caller” here is the customer’s own POS extension dialing out), so it does not contain an outside caller to pop. On an **inbound** call, the outside caller’s name and 10-digit number sit in that same `From` line — that is exactly what we read,

and we already confirmed it is present and unencrypted in the companion analysis for this site (Tech-Response-RingCentral-SIP-CallerID-08JUN26).

Finding 3 — The “OPUS” packets are just the voice audio.

The **5,313** packets Wireshark labels `OPUS` are the RTP media stream — the actual sound of the call — flowing between the phone (`10.1.10.147`) and RingCentral’s media server (`208.87.40.213`). Wireshark prints “OPUS” because it recognized the codec the two ends agreed on in that audio offer above. There is no caller ID in any of it.

3. The actual dispute: “but it reports caller ID intermittently”

Two separate facts combine here. First, what this capture proves on its own: the signaling at this site is delivered **TCP-only, in cleartext**. Second, what we know from our own product: the legacy Vertex is built to read **UDP** SIP and is not designed to reassemble SIP delivered over TCP. Neither fact alone settles the argument — together they do: the caller ID is readable, just not by this device on this transport.

Here is how to think about the “intermittent” claim, because it is the crux. The Vertex reads caller ID by watching the call setup (the `INVITE`) cross the network and pulling the name and number out of it. That works cleanly when the call setup arrives as **UDP**: UDP is *datagram-oriented* — each call setup is delivered as its own self-standing unit the box can grab and read. That is the workflow the legacy Vertex was designed around, and it is how this customer ran for years.

Over **TCP** it is a different animal. TCP is *stream-oriented* — a single SIP message can be spread across several packets that must be collected and reassembled in the right order before the caller ID can be read, and the legacy Vertex was never built to do that reassembly. **So whether it manages to read any given TCP call setup is left to chance:** a short one that arrives in a single packet may be read, while anything that spans multiple packets — or any piece dropped by the switch mirror along the way — cannot be put back together.

Why “intermittent” is exactly what a transport mismatch looks like. This is the behavior to expect any time a UDP-era passive reader is pointed at TCP-delivered signaling: it succeeds on the setups it happens to receive whole and fails on the ones it does not, with no way to tell the difference in advance. And the size headroom is already tight — in this capture even a plain outbound call setup was **1,295 bytes, about 86% of one standard network packet** — so the larger inbound setups that carry an outside caller’s name and number can cross the line where TCP splits a message across packets. A setup that splits, or a packet the mirror drops, is one the legacy box cannot reassemble. That on-again/off-again result is the **symptom of the transport mismatch**, not the device doing its job.

Do not let “I saw it pop once” reopen the question. A caller ID appearing now and then does *not* prove the legacy unit is a fit on a TCP RingCentral site. It only proves the signaling is readable — which we already know. It is **impossible to make the legacy**

Vertex reliable on this transport, and intermittent caller ID is not a product we can stand behind, quote, or support. Treat “works sometimes” as “does not work.”

4. What to tell the customer, and what to do

Lead with the good news, because it is true: this is **not** an encryption problem, and it is **not** an “OPUS” problem — their caller ID is being delivered in plain text and is fully readable. The single issue is that it now arrives over **TCP**, which the legacy hardware cannot follow reliably. Three ways to fix it, cheapest first:

Worth knowing before you call RingCentral: RingCentral is *not* a TCP-only platform. Their network still supports plain, unencrypted SIP over **UDP on port 5060** in SIP-trunk, manual-provisioning, and Bring-Your-Own-Carrier/SBC setups (per RingCentral’s published network-requirements and BYOC/SIP-trunk configuration guides). What they now default to for their *hosted, auto-provisioned* desk phones is **TLS/TCP**, and that traffic rides RingCentral’s outbound-proxy port **5090** — which is exactly what we see at this site (devices *send* to 5090 and *receive* on 5060). So “this site is on TCP” is a **provisioning state on this account**, not a hard limit of RingCentral — which is why option 1 below is worth pushing on.

1. **Ask RingCentral to provision these phones for UDP SIP (port 5060).** Transport is a provisioning choice, not something baked into RingCentral (see the note above), so this is a real request — not wishful thinking. If their support can move these endpoints back to UDP, that most likely restores the exact delivery that fed the POS for years — on the hardware already on site, for free. **The catch:** because RingCentral now defaults its hosted phones to TLS/TCP, whether UDP is available depends on the account and phone type. Ask them directly; if they will not offer UDP for these phones, go to option 2 or 3.
2. **Put a small on-site SIP converter (SBC) in front of the Vertex.** The phones talk to it over **UDP** on the LAN — the leg the Vertex taps and reads — while it carries the calls up to RingCentral over TCP. This gives the legacy box back the UDP it understands. Because RingCentral is hosted, this usually needs their bring-your-own-device / SIP-trunk blessing, so confirm with them before buying anything.
3. **Move to a TCP-capable Vertex (the Vertex Quantum platform).** This is the only one of our units designed to read SIP-over-TCP directly, while keeping the same passive, plug-in install. It is the right answer when the carrier will not go back to UDP and the customer does not want a converter in the call path.

Regardless of which path: confirm the Vertex is on a properly configured switch mirror / SPAN that copies *both directions* for every phone without dropping packets. A passive tap that only sees half the traffic cannot read a clean call setup even on UDP.

5. Quick reference — spotting this yourself in Wireshark

- **Use our filter** `tcp.port == 5090 || sip`. The broader `tcp.port==5060/5061 || udp... || rtp || sip` filter in the screenshot above largely misses RingCentral (its SIP is on TCP

5090, not 5060) and the `rtp` term just floods the view with audio.

- **If you can read `INVITE` / `REGISTER` / `200 OK` in plain English, it is not encrypted.** Encrypted SIP would show only `Application Data`.
- **Ignore the `0PUS` rows for caller-ID work** — that is the voice audio. The caller ID is in the `From` header of the `INVITE`.
- **Check the transport on the call setups.** If they say `transport=tcp` and there is nothing on UDP, the legacy Vertex is the wrong tool for that site — route it to option 1, 2, or 3 above.