BLUEDOT

COMMUNITY EDITION · ALPHA

BlueDot Quantum IPoE BNG

Theory of Operations — Data Plane & Control Plane

An operator- and NOC-facing theory of operations for the BlueDot Quantum IPoE BNG. It explains how the control plane and data plane divide the work, what happens to a subscriber from the first DHCP Discover through connection, rate limiting, captive-portal renewal, and teardown, and what an operator sees, configures, and watches while the box is running. It is written for the person who runs the BNG, not the person who compiles it.

Document Type	BlueDot Theory of Operations
Component	Quantum IPoE BNG (IPv4, Phase 1)
Edition	Community Edition — Alpha

Audience	ISP Operators & NOC
Source Tree	`ipoe/bng` (`main`)
Classification	**BlueDot Proprietary**
Date	July 2026

Contents

Introduction

This document explains how the BlueDot Quantum IPoE BNG **works in operation** — what it does with a subscriber's traffic, how a subscriber moves from "just plugged in" to "connected and paying," and what you, the operator, see and control while the box runs. It is a *theory of operations*, not an installation guide or an API reference. The goal is that after reading it you can look at the BNG's health page, its session list, and its metrics and know what the numbers mean and why they move.

A BNG (Broadband Network Gateway) sits at the edge of your access network. Its job is to take raw packets coming in from subscriber routers and turn them into **managed, authenticated, rate-limited, and accounted sessions** — deciding who is allowed on, how fast they may go, where their traffic is sent, and how much they used. The BlueDot BNG does this job for IPoE subscribers on commodity x86 hardware, and it is built to replace a chain of MikroTik-class boxes with a single architecture that grows by adding nodes rather than by swapping in a bigger box.

The single most important idea in this document — the one everything else hangs on — is the split between two "planes":

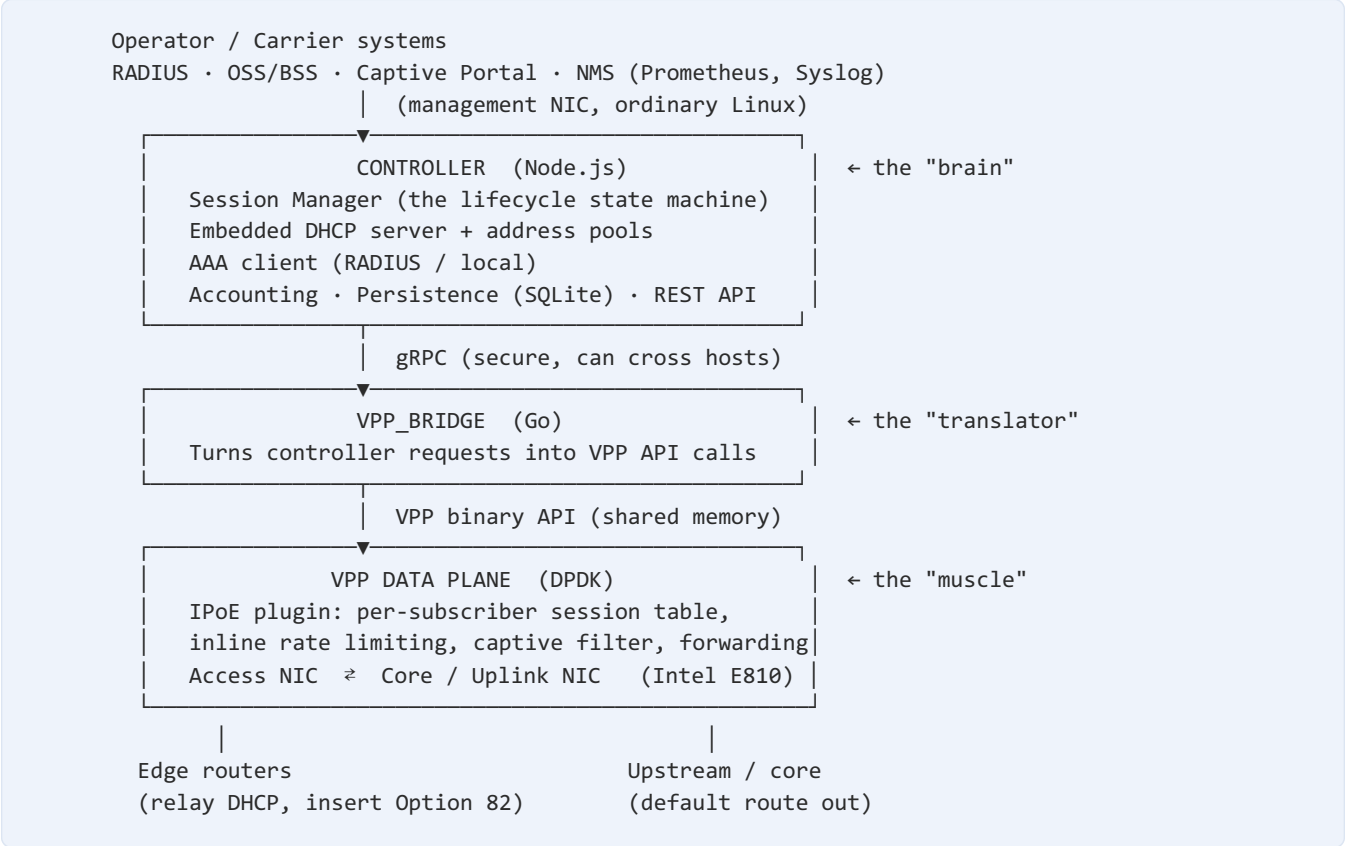
- The **data plane** forwards every byte of subscriber traffic at line rate. It is fast, simple, and never talks to a subscriber-management database on a per-packet basis.
- The **control plane** holds all the intelligence: who is who, what plan they are on, when their session started, when it expires, and what to tell the data plane to do.

Understanding where a given behaviour lives — in the fast data plane or in the thinking control plane — is the key to operating and troubleshooting the BNG. This document is organised around that split: first the shape of the system, then the data plane, then the control plane, then a walk-through of the full life of a subscriber, and finally the practical business of running the box.

Scope and one caveat. The product is deliberately narrow: **IPoE only** (no PPPoE), **IPv4 only** in this phase, an **L3-routed access network** (no VLAN per subscriber), and commodity hardware. One caveat colours the whole document and is repeated where it matters: this **Alpha, Community Edition** build ships with a **local authentication back-end** standing in for a full RADIUS server. Everything else — the data plane, DHCP, the session lifecycle, captive portal, accounting counters, persistence, and the management API — is real and running. The production RADIUS client is a well-defined future addition, noted at each point where it changes what you would see.

The System at a Glance: Two Planes, Three Processes

The BNG is not one program. It is three cooperating processes, and knowing which one does what is the foundation for everything that follows.



The data plane — VPP. This is the muscle. It runs in userspace on DPDK and drives the two high-speed NICs: an **access-facing** NIC (toward subscribers, via edge routers) and a **core/uplink-facing** NIC (toward the internet). Once a subscriber is set up, *every* one of their packets is handled here, at line rate, with no involvement from the controller. This is what lets a single node carry on the order of 100,000 concurrent subscribers on a commodity Dell R360-class server.

The translator — vpp_bridge. A small Go process that connects the controller to VPP. The controller does not speak VPP's low-level API directly; it makes a handful of high-level requests ("add this session," "change this rate," "give me the counters") and vpp_bridge turns them into VPP calls. Because the link between them is gRPC, the controller can even run on a *different machine* from the data plane.

The brain — controller. A Node.js process that holds everything intelligent: the **Session Manager** (the state machine that tracks each subscriber's lifecycle), the **embedded DHCP server** and address pools, the **AAA client** that authenticates subscribers, the **accounting** engine, the **persistence** store that survives restarts, and the **REST API** that you and your other systems use to see and control the box.

The management NIC is an ordinary Linux interface — not DPDK. All of your out-of-band traffic rides here: RADIUS (UDP 1812/1813, and 3799 for CoA), the gRPC link to vpp_bridge, the REST API, Prometheus scrapes, and SSH.

Start-up order matters: VPP first, then vpp_bridge, then the controller. The controller expects the data plane to be reachable when it comes up so it can reconcile and, if needed, re-program sessions.

Fast path vs. slow path

The whole design turns on keeping the fast path pure. In steady state, the controller sees **none** of a connected subscriber's traffic. Only two tiny classes of packet ever leave the data plane and go "up" to the controller (this is called *punting*):

1. **DHCP from an unknown subscriber** — someone new trying to get online, or a known subscriber renewing/releasing their lease.
2. **Inbound RADIUS CoA** — the AAA server asking to change or disconnect a session (in the target design; see the AAA and scope sections).

Everything else — all actual subscriber data — stays in VPP and is forwarded, policed, filtered, and counted without the controller ever touching it. When you understand that "in steady state, nothing is punted," you understand why the box can forward at line rate and why the controller's CPU stays low even with a full session table.

The Data Plane: How Subscriber Traffic Is Handled

Once a subscriber is connected, everything that happens to their packets happens in VPP. This section is the operator's view of that fast path.

Subscribers are identified by their IP address

Because the access network is **L3-routed** (the edge router is a true routing hop, not a bridge), the BNG identifies each subscriber by the **IP address it assigned them** — their *framed IP*, a /32 host address. There is no VLAN per subscriber and no reliance on the subscriber's MAC for forwarding. The edge router relays the subscriber's DHCP to the BNG, inserts **Option 82** (circuit and remote identifiers that say *which port on which edge router* this subscriber is behind), and sets `giaddr` to identify itself.

For you as an operator this has a few concrete consequences:

- A subscriber's session is looked up by their IP, both for traffic they send (source IP) and traffic they receive (destination IP).
- The BNG enforces **anti-spoofing** on every upstream packet: the source IP must equal the subscriber's assigned IP, or the packet is dropped. A subscriber cannot send traffic from an address they were not given.
- Traffic to and from the internet is delivered through the **edge router**, whose address the BNG learned from `giaddr`. The BNG resolves the edge router's MAC once and shares it across every subscriber behind that router — so the neighbour table stays small even at 100,000 subscribers.

What happens to each packet

Every subscriber packet passes through a compact sequence of checks. The elegance of the design — and the reason it is fast — is that a **single lookup** of the subscriber's session pulls everything the data plane needs into one place: the anti-spoof reference, the "is this subscriber behind the captive portal?" flag, and the two rate-limiting meters (one for each direction). There is no separate firewall table and no separate rate-limit table to consult.

Upstream (subscriber → internet):

1. **Look up the session** by the packet's source IP. No session (and not DHCP)? Dropped — an unprovisioned address gets nothing (deny by default).
2. **Anti-spoof:** source IP must match the session's assigned IP.
3. **DHCP interception:** a DHCP packet from a known subscriber (a renew or release) is punted to the controller.
4. **Captive check:** if this subscriber is behind the walled garden, only DNS and web traffic *to the portal* survive; everything else is dropped (see the captive-portal section).

5. **Rate limiting:** the packet is measured against the subscriber's upstream rate and coloured green, yellow, or red (below).
6. **Forward** toward the core/uplink NIC and out to the internet.

Downstream (internet → subscriber): the mirror image — look up the session by destination IP, apply the captive check in reverse, meter against the subscriber's downstream rate, and deliver toward the subscriber via the edge router.

Rate limiting: green, yellow, red

Every subscriber gets two rate limiters, one upstream and one downstream, built right into their session. They implement the standard **two-rate, three-colour** policing model (RFC 2698). Two numbers define each direction:

- **CIR** — the *committed* rate, the speed the subscriber is guaranteed.
- **PIR** — the *peak* rate, the most they may ever burst to.

Each packet is then coloured:

Colour	Meaning	What happens
Green	Within the committed rate (≤ CIR)	Forwarded, counted
Yellow	Bursting above CIR but within peak (≤ PIR)	Forwarded, counted
Red	Exceeds the peak rate (> PIR)	**Dropped**, counted

This colouring is not just a forwarding decision — it is a **measurement**. For every subscriber the data plane keeps a full set of counters split by colour, by direction, and by unit (bytes and packets). That means you can see, per subscriber, not just *how much* they used but *how much was in-contract (green), how much was burst (yellow), and how much was shed at the cap (red)* — in each direction. A subscriber whose red counter is climbing is constantly hitting their limit: that is both a quality-of-experience signal (a candidate for an upsell) and a health signal. Community BNGs typically give you two numbers per subscriber; this one gives you a far richer picture, and it costs nothing on the fast path.

Alongside the per-subscriber counters, the data plane keeps six **node-wide** band counters (green/yellow/red × up/down). These let you read what the *whole box* is doing by QoS band in a single cheap query, without walking the session table — ideal for a high-frequency dashboard.

What the data plane does not do

It never authenticates, never assigns addresses, never decides policy, and never writes to a database. It only forwards, polices, filters, and counts, based on the session state the controller installed. Every decision it makes was pre-loaded by the control plane. That is the whole point of the split.

The Control Plane: The Subscriber Lifecycle

The controller is where subscribers are born, change state, and are torn down. At its heart is the **Session Manager**, an explicit state machine that models the life of every subscriber.

The five states

State	Meaning
AUTHENTICATING	A new subscriber's DHCP Discover arrived; the BNG is asking AAA whether to admit them. Transient.
OFFERING	AAA said yes; an address has been offered and the data plane pre-programmed. Waiting for the subscriber to confirm. Transient.
CONNECTED	Fully online. Traffic flows at line rate; accounting and timers are running. This is the normal steady state.
CAPTIVE	Behind the walled garden. The subscriber keeps their IP but can reach only the captive portal — used for unknown subscribers and for expired prepaid sessions.
TERMINATING	Being torn down. Final accounting is sent, then the session is removed. Transient.

Three different kinds of event drive these transitions, and it helps to keep them straight because they show up differently in logs and metrics:

- **DHCP events** punted up from the data plane (Discover, Request, Release).
- **Timer events** inside the controller (session expiry, idle expiry, accounting intervals).
- **Operator / AAA events** via the REST API or RADIUS CoA (disconnect, activate, re-rate).

Onboarding: DHCP and AAA are one act

In an ordinary network, DHCP and authentication are separate systems that do not know about each other. In a BNG they cannot be — the address you hand out, the decision to admit the subscriber, and the programming of the data plane must be a **single coordinated act**. That is why the BNG **embeds its own DHCP server** inside the controller rather than relaying to an external one.

The onboarding handshake, at operator altitude, runs like this:

1. A new subscriber's **DHCP Discover** reaches the BNG (relayed by the edge router, carrying Option 82). It matches no session, so it is punted to the controller. → **AUTHENTICATING**

2. The controller **authenticates** the subscriber with AAA, using the identifiers it has (Option 82 circuit/remote IDs, the subscriber's MAC, the edge router identity).
3. On **accept**, the controller **resolves the service plan** (the rates), **allocates an IP** from a pool (or uses one AAA specified), **programs the data plane** with the new session, arms an offer-expiry timer, and sends the **DHCP Offer**. → **OFFERING**
4. The subscriber replies with a **DHCP Request** to confirm. The controller creates the lease, **starts accounting**, arms the session timers, and sends the **DHCP Ack**. → **CONNECTED**

From that moment the subscriber is in the fast path and the controller sees nothing more of them until a renewal, a release, a timer, or a policy change.

On a **reject** — AAA cannot identify the subscriber — the BNG does not simply drop them. It assigns an address, programs the session **with the captive flag on**, and offers the lease anyway, landing the subscriber directly in **CAPTIVE** so they can reach the portal and sign up.

A note on pools and addresses. Address pools are simple: each is just a range and a lease time. A **single subscriber pool is recommended** for most operators. There is deliberately **no "captive pool"** — captive is a property of the *session*, not of the address range, so a subscriber keeps the same IP whether they are connected or behind the portal. Multiple pools are supported only for address-management convenience (region, scale), never to separate captive from full-access subscribers.

Authentication and service plans

RADIUS is the intended authority for *who* a subscriber is and *what* they may do. All available identifiers are sent to the AAA server and its policy decides. Two BlueDot-specific conveniences sit on top:

- **A local plan table.** You can define service tiers (their rates) **locally on the BNG** and have AAA simply name the plan. This means most operators never have to ship a vendor dictionary into their RADIUS server. When a plan name comes back, the BNG looks it up locally and applies those rates.
- **A guarantee that no session is ever unpoliced.** If AAA returns explicit rates, they are used; if it names a plan, the plan is used; if it returns neither, a default (Bronze) plan is applied and a warning is logged. A subscriber is never left with no rate limit.

For operators who *do* want RADIUS to carry the full rate set, BlueDot defines eight vendor attributes (up/down × committed/peak × rate/burst) under its assigned enterprise number (**PEN 65664**). That path activates with the production RADIUS client.

> **Alpha caveat.** This build runs a **local AAA provider** instead of a live RADIUS server. It has an *open* mode (admit everyone at a flat rate — the free-tier default) and a *gated* mode (admit known lines, send unknown-but-identifiable lines to the captive portal, reject terminated lines). The plan-resolution logic above is real and runs regardless of which back-end answers. The production RADIUS client — real sockets, retransmit and failover, the CoA listener, and the RADIUS accounting record assembly — is the defined future addition.

Changing a live subscriber: re-rate and disconnect

A connected subscriber's rate can be changed **without disconnecting them**. Two paths do this:

- **Operator plan edit (works today).** Editing a plan via the REST API triggers a **live re-rate**: the controller walks the session table and updates the data-plane rate limiter for every connected subscriber on that plan, in place. No re-DHCP, no interruption.
- **RADIUS CoA (target design).** The AAA server can send a *Change-of-Authorization* to re-rate a subscriber, or a *Disconnect-Message* to drop them. The subscriber stays connected through a re-rate; a disconnect tears the session down cleanly. The CoA listener is part of the production RADIUS client.

Teardown, and why order matters

A subscriber leaves the CONNECTED state for several reasons, all funnelled through one teardown path that stamps a standard **terminate cause** so your billing system knows *why* the session ended:

Cause	Meaning	Trigger
1	User-Request	Subscriber sent DHCP Release
4	Idle-Timeout	No traffic for the idle period
5	Session-Timeout	Plan time elapsed → goes to **CAPTIVE**, not down
6	Admin-Reset	Operator disconnect, or RADIUS CoA disconnect
15	Service-Unavailable	Recovered after a restart but too old to keep

The teardown does one thing in a very specific order: it **reads the subscriber's final traffic counters from the data plane first, sends the closing accounting record second, and only then removes the session.** If it removed the session first, the final bill would be stale or lost. This ordering is a genuine correctness rule, not an incidental detail.

The Life of a Subscriber, End to End

It is worth walking the whole arc once, because it ties the two planes together and mirrors what you will actually see in the session list and metrics.

1. **A new subscriber powers on.** Their router sends a DHCP Discover; the edge router relays it to the BNG with Option 82. The data plane has no session for them, so it punts the packet to the controller. *(A brief AUTHENTICATING state.)*
2. **The controller admits them.** AAA accepts, a plan is resolved to a pair of rates, an address is drawn from the pool, the data plane is programmed, and an Offer goes out. *(OFFERING.)*
3. **The subscriber confirms.** Their DHCP Request comes back, the lease is created, accounting starts, timers are armed, and the Ack is sent. *(CONNECTED.)*
4. **They use the internet.** Every packet now flows entirely through the data plane at line rate — looked up by IP, checked for spoofing, metered green/yellow/red, forwarded. The controller sees nothing. Meanwhile a background loop periodically reads their counters, sends interim accounting, and watches for idleness.
5. **Their prepaid time runs out.** The session-timeout fires. Rather than disconnect them, the BNG closes accounting (cause 5), flips the captive flag, and moves them to **CAPTIVE** — *keeping their IP*. Their next web request lands on the "top up to continue" page. Nothing is disconnected; nothing re-DHCPs.
6. **They pay.** The portal calls the BNG's activate endpoint. The captive flag clears, accounting restarts, timers re-arm, and they are **CONNECTED** again — same IP, no interruption.
7. **They go home / power off.** Their router sends a DHCP Release (or they simply go idle, or an operator disconnects them). The BNG reads their final counters, sends the closing accounting record with the right cause, and removes the session. The address returns to the pool.

Every step above is visible to you: the state distribution in the metrics, the per-subscriber counters, the DHCP protocol counters, and the accounting records all narrate this arc.

The Captive Portal and the Prepaid Model

The captive portal is central to the target business — prepaid fixed-wireless — so it is worth understanding precisely.

How the walled garden works. When a subscriber is in CAPTIVE state, the data plane permits exactly three things and silently drops everything else:

- DNS (UDP/53) to the portal,
- HTTP (TCP/80) to the portal,
- HTTPS (TCP/443) to the portal.

How the redirect works — DNS steering. A captive subscriber's DHCP lease hands out **the portal's DNS server as their only resolver**, and that server answers *every* lookup with the portal's address. Combined with the walled garden, any website the subscriber types resolves to — and can only reach — the portal. There is no in-path HTTP interception to configure; the redirect is a consequence of DNS plus the filter.

Why expiry goes to CAPTIVE, not disconnect. This is the prepaid-first rule. When a subscriber's plan time elapses, keeping their IP and pushing them behind the portal means their next browser action is a "top up to continue" page — the exact conversion path a prepaid operator depends on. A forced disconnect and re-DHCP would break that experience. So Session-Timeout is a *demotion to captive*, and only a Release, idle-out, or explicit disconnect actually removes the session.

Entering and leaving captive:

Into CAPTIVE	Out of CAPTIVE
AAA rejected an unknown subscriber at onboarding	Portal calls `POST /bng/sessions/{ip}/activate` after payment
A connected subscriber's plan time expired	(which clears the flag, restarts accounting, re-arms timers → CONNECTED)

Accounting: Measuring What Subscribers Use

Accounting is how usage becomes billable records. The BNG emits three kinds:

- **Start** — sent when a subscriber connects (counters at zero).
- **Interim** — sent periodically during the session, carrying cumulative usage.
- **Stop** — sent at teardown, carrying final usage and the terminate cause.

Two details shape what the numbers mean:

- **Only green + yellow are billed.** Red traffic was *dropped* at the policer — the subscriber never received it — so it is excluded from the billed octet counts. Red is still reported through the API for diagnostics; it just is not on the bill.
- **Large counters are split for RADIUS.** VPP counts in 64-bit, but RADIUS octet fields are 32-bit, so each count is split into the octet field plus a "gigawords" field (units of 4 GB). Your billing system reassembles them; byte counts stay exact well into the petabyte range.

Each session carries a stable accounting ID across its Start/Interim/Stop, and echoes back verbatim any opaque `Class` value the AAA server attached — the BNG never inspects it.

> **Alpha caveat.** The counter substrate — the per-colour, per-direction byte and packet counts, refreshed from the data plane — is real and running today. The assembly of those counts into RADIUS accounting *packets on the wire* is part of the deferred production RADIUS client.

Timers and Scale: What Bounds a Node

Three time-based jobs run for every connected subscriber: **interim accounting**, **session expiry**, and **idle detection**. Done naïvely — a separate timer per job per subscriber — that would be hundreds of thousands of timers on a full node, all firing and all forcing the controller to save state, producing a stampede that has been measured to badly hurt latency even at a thousand sessions.

The BNG instead folds all of this into a single batched loop (informally, the *tickler*). On each pass it:

- reads the counters for all due subscribers in **one** bulk request,
- writes the updated state **once**, and
- emits the interim accounting records that are due, and tears down any subscriber that tripped idle or session-timeout in that same pass.

Idle detection needs no timer at all: it is derived from the counters. If a subscriber's total traffic has not moved since the last poll, their idle streak advances; when it reaches the idle limit, they are torn down (cause 4). Session expiry is likewise a simple comparison of "now" against a stored deadline.

For you, the operational takeaways are:

- **A node carries on the order of 100,000 concurrent subscribers** on commodity R360-class hardware. The limit is memory/cache behaviour, not raw CPU. (A future per-worker-table option would raise this to roughly 400,000–500,000, but that is out of scope for this release.)
- **Growth is horizontal.** One controller can drive multiple data-plane nodes; you add capacity by adding a node, under the same operator-facing configuration, not by replacing the box.
- The controller **measures its own** save timing (see Observability). Those numbers are your early warning that a node is approaching its ceiling.

Persistence and Recovery: Restarts Don't Drop Subscribers

Because the controller and the data plane are separate processes, a restart of one must not disconnect subscribers being carried by the other. An integrated single-box BNG can afford to lose all sessions on restart; a disaggregated one cannot, because a controller restart that dropped every session would knock 100,000 paying customers offline while the data plane was still happily forwarding their traffic.

The BNG therefore treats session state as **durable**. It is written to a local database on every state change. The clever part is that the things you cannot save — live timers, sockets — are reconstructed on start-up from **absolute expiry timestamps** stored with each session. On restart, every timer is simply rebuilt from its deadline; a deadline already in the past just fires on the next pass.

Recovery then takes one of three forms depending on what actually restarted:

Situation	What the BNG does
Controller restarted, data plane stayed up	Rebuild the controller's in-memory state and re-arm timers from the saved deadlines. **No data-plane changes** — VPP still holds the live sessions.
Controller and data plane both fresh	Rebuild controller state, then **re-program every connected session** into the data plane so the two match.
Data plane diverged (its state no longer trusted)	Tear down the untrusted data-plane state and re-program it from the controller's session records.

Two more operational guarantees:

- **Upgrades preserve live sessions.** A package *upgrade* keeps all state, so applying an update does **not** disconnect subscribers. (A fresh *install* starts clean.) This is a deliberate BNG-specific behaviour.
- **Stale sessions are closed cleanly.** After a long outage, any session older than a recovery-age limit (default 24 hours) is discarded with a proper closing accounting record (cause 15), so the billing ledger always sees a clean boundary.

Configuration: What You Set, and Where

Configuration lives in two places, split by **ownership**, not convenience. A setting lives in exactly one place; there is no merge logic to reason about.

File-owned (`bng.json`, read once at start-up). The security-sensitive and rarely-changed settings: RADIUS server lists and **shared secrets** (injected from environment variables, held only in memory, and *never* written to the database or exposed through the API, so a state dump cannot leak them), the captive-portal address, upstream DNS servers, and non-AAA defaults. Changing file-owned config means a controller restart — there is no live reload.

API-managed (in the database, changed only through REST). The things you tune day to day:

What	Endpoint (orientation only)	Notes
Address pools	`PUT/DELETE /bng/config/pools/{name}`	Range and lease time. Must exist before subscribers can get addresses.
Service plans	`PUT/DELETE /bng/config/plans/{name}`	Editing a plan **live re-rates** every connected subscriber on it.
Edge routers	`PUT/DELETE /bng/config/edgeRouters/{giaddr}`	Optional map of edge-router → gateway IP; falls back to the relay address if absent.

The guiding principle across all integration is **poll over push**: the BNG exposes a stable API surface and your other systems (OSS/BSS, CRM, IPAM, NMS, the portal) query it on their own schedule. The BNG does not push data outward (its only outbound initiation is RADIUS), which keeps it from being coupled to any one vendor's schema. Every API call is authenticated with an API key.

Operating the BNG: What to Watch

This is the section to keep by the NOC dashboard. Everything here is available on the management NIC via the REST API and Prometheus.

The one-glance health check

`GET /bng/health` rolls the box up to a single status — **ok**, **degraded**, or **fault** — using a deliberate rule:

- **fault** — the data plane (VPP) is down, **or** all RADIUS endpoints are unreachable. Subscribers are affected; act now.
- **degraded** — exactly one of a redundant pair of RADIUS endpoints is unreachable. Still serving; investigate.
- **ok** — serving normally.

Notably, a **full address pool is reported but does not, by itself, drive the status to fault** — it is an operational condition to act on (add addresses or a node), not a service-down event. The same underlying signals are also exposed as Prometheus gauges (`vpp_up` , `radius_up`) so your Alertmanager can page on them directly. This pull-based model replaces SNMP traps.

The metrics that tell the real story

Signal	What it tells you	Watch for
Sessions by state	How many subscribers are onboarding / connected / behind the portal	A pile-up in OFFERING (onboarding failing); a surge in CAPTIVE (mass expiry or AAA rejecting)
Per-subscriber red counter	A subscriber constantly hitting their rate cap	Upsell candidates; also abuse
Node-wide QoS bands	What the whole box is doing by colour, this second	Aggregate red climbing = policers shedding load
DHCP counters (discovers, offers, requests, acks)	Onboarding health and conversion	Discovers with no matching acks = onboarding trouble; a reset to zero = controller restarted
Pool utilisation	Address headroom per pool	Rising utilisation *before* it becomes exhaustion
Session cap / onboard-refused	The box is full and refusing new subscribers	A climbing refusal count = time to add a node (relevant to the free-tier cap)
Controller save timing	The controller's own persistence latency	Rising values = approaching the per-node scaling ceiling

Two read paths for counters

There are deliberately two ways to read usage, and they behave differently:

- The **accounting view** (`GET /bng/sessions`) is the billing-grade snapshot, refreshed on the interim cadence. Use it for reconciliation and per-subscriber detail.
- The **telemetry view** (`GET /bng/counters` , `GET /bng/counters/{ip}`) is a *pure read* straight from the data plane — it takes no locks and triggers no save, so you can poll it as fast as you like for live dashboards without slowing the control plane. Large counters cross the API as decimal strings so they never lose precision.

Structured JSON logs go to stdout (pipe them to your log collector or SIEM); Prometheus is scraped from `/metrics` ; and the authoritative request/response shapes live in the generated OpenAPI spec, not in this document.

Scope and Caveats

Two different kinds of boundary apply to this box, and it helps to keep them apart. **Scope** decisions are deliberate choices about what the Quantum IPoE BNG *product line* is and is not — they are stable and will not change from release to release. **Caveats** are specific to *this* Community-Edition Alpha build — they describe where the running software stops short of the full product design today, and each has a defined path to being filled.

Scope — Product-Line Decisions

These are intentional edges of the product, not gaps to be closed:

- **IPoE only.** No PPPoE anywhere in the product line, and no 802.1X/EAP. Subscribers are IP-over-Ethernet, relayed to the BNG by the edge router.
- **L3-routed access, no per-subscriber VLANs.** Subscribers are keyed by their assigned IP; the edge router is a true routing hop that relays DHCP and inserts Option 82. There is no VLAN-per-subscriber or QinQ model.
- **Notifications are pull-based.** Prometheus gauges + Alertmanager replace SNMP traps in the default deployment. SNMP and syslog-over-TLS to a SIEM are per-deployment add-ons, not part of the default design.
- **Policing, not shaping.** Rate control is RFC 2698 two-rate three-colour policing (drop on red). Queue-based shaping / AQM tiers are out of scope for the product line and would be additive on top of the policer.
- **No hard data-plane volume cap.** RADIUS remains the authority for subscriber policy; the BNG reports usage and honours CoA-driven enforcement. Small cycle-aligned overage is expected and acceptable by design.

Caveats — Specific to This Release

These describe where this build stops short of the full design, each with a defined path forward:

- **Authentication is local in this build.** The production RADIUS client — live sockets, retransmit and dual-server failover, the CoA/Disconnect listener, and RADIUS accounting-record assembly — is the defined next addition. Today the box runs a local AAA provider (open or gated) and can operate **standalone with no RADIUS server at all**, which is exactly the Community-Edition free-tier on-ramp. Design surfaces that depend on RADIUS (the eight QoS vendor attributes, inbound CoA) exist but light up with that client.
- **IPv4 only on this version.** Address assignment, forwarding, policing, and accounting are IPv4 throughout.
- **No IPv6 on this release.** IPv6 — dual-stack or IPv6-only, DHCPv6, and prefix delegation — is deferred until prioritised for a future version.

- **Single-node capacity in this release.** A node carries on the order of **100,000 concurrent subscribers** on commodity R360-class hardware. The per-worker session-table option that would raise a single node to roughly 400,000–500,000 is not in this build; scale beyond a node's ceiling is achieved by adding nodes.

None of the caveats are gaps in the running system so much as the seam between a **focused Alpha** and the full product design: a line-rate, horizontally-scalable, deeply-observable IPoE BNG that a small or mid-size ISP can stand up on commodity hardware, run standalone today, and federate into RADIUS when the production client lands.

Prepared from the BlueDot Quantum IPoE BNG source tree (`ipoe/bng`, `main`) — the architecture summary, the life-of-packet/-event flows, and the engineering overview. Field-level request/response shapes are authoritative in the generated OpenAPI spec and the source, which change faster than this document.