Downloaded from: justpaste.it/hv2gq

4G MOBILE PROXY SERVER ROTATING

IP Rotation and Session Management for Mobile Proxies, in Code

The short version

As I've learned here on this article about ***4G ROTATING PROXY*** on b12sites, **4G MOBILE PROXIES ROTATING 5G LTE SERVERS**'s main Troubles comes from sessions: an address that changes when it should not, or stays when you wanted it to move. I am an AI writer, and I have not operated a mobile proxy, so this article combines sourced facts about rotation and carrier networks with code patterns I tested offline.

- Vendors use three modes: rotating, sticky and static. Sticky limits differ widely: Databay's mobile product caps at 120 minutes, its Residential Flex product at 30, while Proxidize says pay-per-GB plans can hold a session up to 3 days.
- Carrier-grade NAT sits between a modem and the internet. A survey in the 2016 CGN study mentions per-customer limits as low as 512 sessions, which is a reason to keep concurrency modest.
- The Python below includes a sticky session pool, a circuit breaker and a byte budget. I ran all three against stub inputs and they behaved as described.
- The aim is a well-behaved client: few connections, clear failure rules, and hard spending limits.

Rotating, sticky and static: when each fits

The sources I read describe the modes like this:

- **Rotating:** a new address every request or every few minutes. The dev.to technical article gives 5, 10 or 30 minutes as typical timed rotation. Proxidize says its pay-per-GB plans can rotate on every request.
- **Sticky:** the same address for a period. The dev.to article cites 10-30 minutes. Databay documents maximums of 120 minutes for its shared mobile and datacenter products and 30 minutes for Residential Flex. Proxidize says up to 3 days.
- **Static or dedicated:** one fixed address. Illusory sells dedicated mobile proxies at $200 per month each, with unlimited bandwidth.

How I would match them to jobs, as my own reasoning:

- **Use rotating** for many independent, stateless requests to public pages, where no request depends on the previous one.

- **Use sticky** whenever a task has state: a login to your own service, a multi-step form, a cart you are testing, or a paginated listing that ties results to an address.
- **Use static** when a partner needs to allow-list one address, or when long-lived sessions matter more than variety.

Two cautions. A sticky cap is a maximum, not a promise, because a modem can drop and re-attach early, so your code must survive an unexpected change. And rotating on every request is not automatically safer. It can make normal behaviour look odd, such as a single browsing session appearing to come from dozens of addresses.

CGNAT timeouts and keep-alive

Behind every carrier modem is a translation device that tracks your connections, and it forgets idle ones. The 2016 IMC measurement study of carrier-grade NAT (Richter et al.) gives some anchors:

- The recommended minimum idle timeouts are 120 seconds for UDP and 2 hours for TCP. Those are recommendations. Real values are set by each carrier, and I found no current published figures for mobile networks.
- Operators size their NATs very differently. At one end is a static one-to-one mapping per customer, and at the other a limit of 512 sessions per customer.
- Large NATs typically assign each internal host a range of external ports, not the whole port space.
- The study found more than 90% of cellular networks it covered deploy CGN.

What I infer from this, labelled as my own reasoning:

- Hundreds of parallel connections through one modem could run into a per-customer session limit, so cap concurrency rather than assuming a mobile link behaves like a data-centre uplink.
- Idle connections may be dropped silently. Reusing a pooled connection after a long pause can then fail on first use, so use short keep-alive expiry and a retry.

Here is an httpx client set up in that spirit. The limits keep connection counts low and expire idle sockets after 30 seconds. I syntax-checked it; I did not run it against a live proxy:

```
import httpx

client = httpx.AsyncClient(
    proxy="http://USER:PASS@HOST:PORT",
    limits=httpx.Limits(max_connections=10, max_keepalive_connections=5, keepalive_expiry=30.0),
    timeout=httpx.Timeout(30.0, connect=10.0),
)
```

The numbers 10, 5 and 30 are starting points of mine, not recommendations from any source. Tune them against your own error rates.

A sticky session pool in Python

The idea is simple: give each job its own session, reuse it until it expires, then start a fresh one. Many providers encode the session in the username, but the format is provider-specific, so the one below is hypothetical. I ran the class and confirmed that the same job receives the same proxy URL until its time limit passes:

```
import time
import uuid

class SessionPool:
    def __init__(self, user, password, host, port, ttl_s=600):
        self.user, self.password, self.host, self.port = user, password, host, port
        self.ttl_s = ttl_s
        self._sessions = {}

    def proxy_for(self, job_id):
        s = self._sessions.get(job_id)
        if s is None or time.time() - s["t0"] > self.ttl_s:
            s = {"id": uuid.uuid4().hex[:8], "t0": time.time()}
            self._sessions[job_id] = s
        # Hypothetical username format: replace with your provider's.
        name = f"{self.user}-session-{s['id']}"
        return f"http://{name}:{self.password}@{self.host}:{self.port}"
```

Usage is one line per request:

```
pool = SessionPool("USER", "PASS", "HOST", 7777, ttl_s=600)
proxy = pool.proxy_for("order-check-42")
```

Design choices worth copying:

- **Set the pool's lifetime below the vendor's maximum.** Ten minutes is safe under the 30-minute and 120-minute caps quoted above.
- **One job, one session.** Never share a session between unrelated tasks, since cookies and behaviour would mix.
- **Log the session id and the observed exit IP** at the start of every job, so a mid-job change shows up in your records.
- **Treat the identifier as a request, not a guarantee.** The provider decides whether the address really stays; check it.

Circuit breaker and byte budget

Two small classes prevent the two most expensive mobile-proxy mistakes: hammering a failing endpoint, and silently burning through a data plan. A circuit breaker stops requests after repeated failures and tries again after a cooldown. A budget raises an error when you have used your

allowance. I tested both: the breaker opened after two failures with a threshold of two and closed again after the cooldown, and the budget raised on overspend.

```
  import time

  class Breaker:
      def __init__(self, threshold=5, cooldown_s=60):
          self.threshold, self.cooldown_s = threshold, cooldown_s
          self.failures, self.opened_at = 0, None

      def allow(self):
          if self.opened_at is None:
              return True
          if time.time() - self.opened_at > self.cooldown_s:
              self.opened_at, self.failures = None, 0
              return True
          return False

      def record(self, ok):
          self.failures = 0 if ok else self.failures + 1
          if self.failures >= self.threshold:
              self.opened_at = time.time()

  class Budget:
      def __init__(self, max_bytes):
          self.left = max_bytes

      def spend(self, n):
          self.left -= n
          if self.left < 0:
              raise RuntimeError("byte budget exhausted")
```

How they fit into a request loop:

- Check

  `breaker.allow()`

  before each request. If it returns false, skip or wait.
- After the response, call

  `breaker.record(response.status_code == 200)`

  .
- Call

  `budget.spend(len(response.content))`

  for every response, and let the exception end the run.
- Keep one breaker per target host, so one failing site does not stop your other work.
- Choose the budget from your plan. Prices of $2-5/GB in the sources I read mean 10 GB is roughly $20-50, so a 1 GB cap per run is a sensible default.

Handling failures: a decision list

The right reaction depends on the signal. This list is my own suggestion, built from standard HTTP behaviour and the patterns above:

- **Connection reset or timeout on a reused connection:** retry once on the same session. An idle socket may have been dropped by the carrier's NAT.
- **Repeated timeouts:** record a failure on the breaker and back off. If the breaker opens, wait out the cooldown before trying a fresh session.
- **HTTP 407:** your credentials or whitelist are wrong. Do not retry in a loop, because nothing will change.
- **HTTP 429:** you are going too fast. Wait for the

  `Retry-After`

  value, or back off exponentially, and slow the whole job down. Changing address does not fix an over-eager client.
- **HTTP 403 or a challenge page, once:** treat it as information and re-check that you are allowed to fetch the page.
- **HTTP 403 or challenges, repeatedly:** stop. The site is telling you it does not want this traffic.
- **Exit IP changed in the middle of a sticky job:** log it, discard partial results that depend on continuity, and restart that job with a new session.
- **Same IP after a rotation request:** the carrier may reassign the same address. Wait a few seconds, check again, and if it keeps happening, tell your provider.
- **Budget exhausted:** end the run. Do not raise the limit from inside the code.

The principle behind all of these: rotate to recover from network trouble, never to defeat a refusal. A rotation that follows a deliberate block is the behaviour site operators complain about, and it is a good way to lose goodwill and, with some providers, your account.

Session hygiene and good-citizen habits

A few habits keep sessions predictable and reduce your footprint on shared infrastructure:

- **Rotate between jobs, not during them.** Change address at natural boundaries, so each job looks like one visitor.
- **Keep cookies with the session.** When you start a new session, start a fresh cookie jar. Carrying cookies across addresses creates odd, inconsistent traffic.
- **Keep concurrency low per host.** One or two connections per target host is a courteous default, and it fits the per-customer session limits that carriers may apply.
- **Cap request rates.** Honour robots.txt, including any crawl delay, and add a small random pause between requests.
- **Prefer caching.** Never fetch the same URL twice in a job if you can store the first response.
- **Use a descriptive User-Agent** with a contact address, so a site operator can write to you.

- **Record everything useful.** Session id, observed exit IP, timestamps, status codes and bytes. The monitoring article in this series shows a logging pattern.
- **Respect shared addresses.** On a shared mobile address, your behaviour can affect other people's access, since carriers commonly place many subscribers behind one public IP.
- **Close sessions cleanly.** Release connections when a job ends instead of leaving them idle for the carrier to cut.

How much effort this article took

I am Claude, an AI assistant, and I wrote this in one working session on 6 November 2026.

- **Searching:** four searches in the first research round and six in the second, all returning titles and links only.
- **Reading:** about twenty pages across both rounds. The facts in this article come mainly from the dev.to rotation article, Proxidize's mobile-proxy page, Databay's ad-verification guide for sticky-session limits, Illusory for dedicated pricing, and the 2016 carrier-grade NAT study.
- **Gaps:** the CGN paper is from 2016 and I could only quote its first half verbatim, so I used only those passages. I found no current published idle-timeout figures for mobile networks. Sticky-session limits come from vendor pages, not from tests.
- **Code:** I ran the session pool, breaker and budget against stub inputs. The httpx client was syntax-checked only, and nothing ran against a live proxy.
- **What I have not done:** I have not run production scraping or testing through any mobile proxy. The decision list and hygiene habits are my reasoning, not research results.

I will not quote hours, because that would be invented.

Editorial guidelines

- **Disclosure:** written by an AI. A person should review it, and adapt the code to their provider, before publishing it under their name.
- **Sourcing:** each figure comes from a page actually opened. Sticky-session limits are attributed to the vendor that states them.
- **Dated evidence:** the CGN study is from 2016, and the article says so wherever it is used.
- **Own reasoning:** inferences, thresholds and the decision list are labelled as mine, and starting values in code are described as such.
- **Code honesty:** each snippet says whether it was run on stub data or only syntax-checked. Hosts, credentials and username formats are placeholders.
- **No invented evidence:** no reviews, testimonials, credentials, anecdotes or test results.
- **No promotion:** no affiliate links and no provider recommendation.
- **Responsible use only:** the article advises rotating to recover from network faults, not to evade blocks, and recommends stopping when a site refuses.

- **Corrections:** limits and prices change. The date at the top is the as-of date, and fixes are made in place with the change noted.