

The Sessions of Theseus

Long-term PCAP storage without losing your session identity

Owen McGill

Arkimeet 2026

Follow-up to "Arkime Stream Processing with Kafka" — Arkimeet 2023

A confession before we start

- The tools you will see today — live-saver, packet-signaler, packet-packer — are 100% human-designed and human-implemented
- This presentation, however, was assembled with the help of an LLM

The Ship of Theseus

- One ship; over the years every plank replaced — is it still the same ship?
- Same question for sessions: rebuild the PCAP, point it at new files — still the same session?
- This talk is about making the answer *yes* — or your queries break

Where we left off — Arkimeet 2023

- The Kafka plugin decouples session indexing from the capture nodes
- SPI processors: indexer, filterer, WISE, matcher — all still in production
- The last slide of that talk was a sketch: filter SPIs, reconstruct the PCAPs, store them in S3
- This talk is what happened after that sketch — and the detour in between

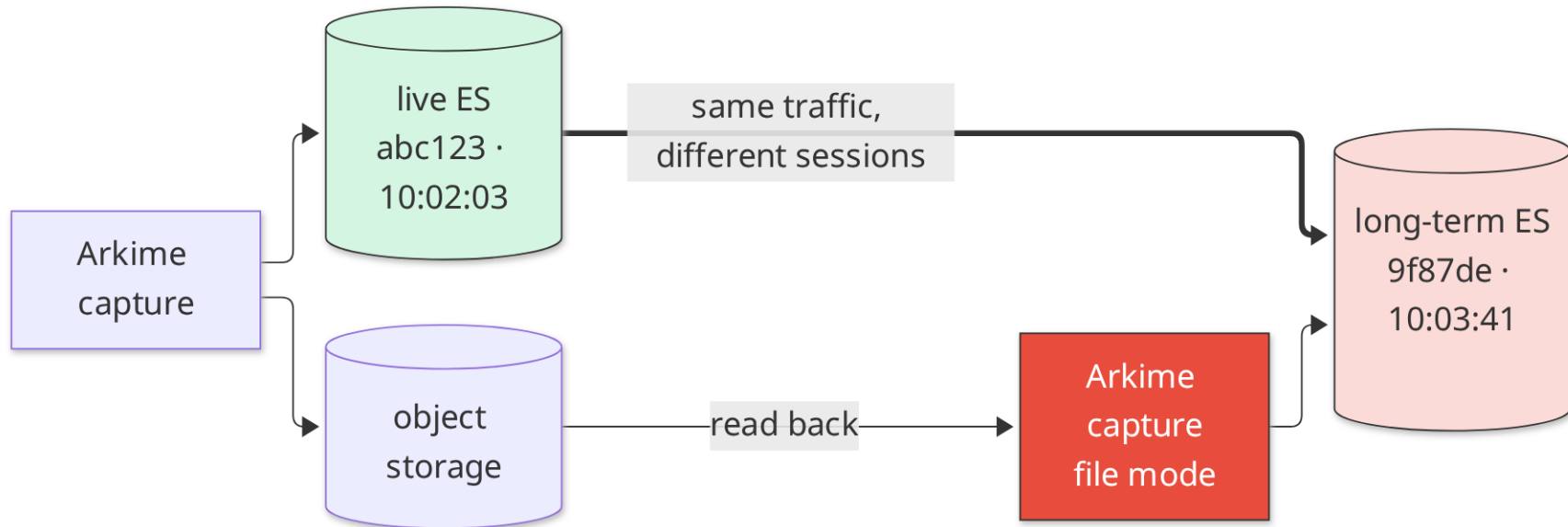
[2023 slides](#)

[2023 recording](#)

We had two Arkime clusters. They disagreed.

The old long-term flow

- PCAPs stored in object storage, then read back and reprocessed through Arkime in file-reading mode
- Indexed into a separate long-term Elasticsearch cluster
- Same packets, different sessions



The best idea we had at the time

Re-capturing is re-creating

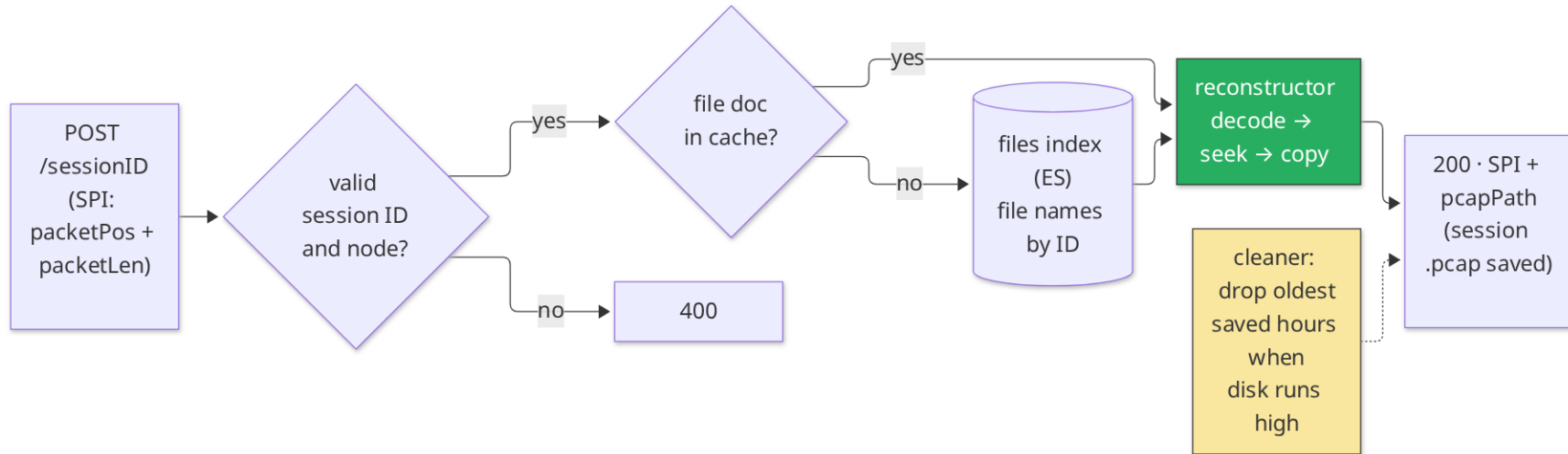
- Arkime session identity is minted at capture time
- Session boundaries depend on how capture sees the packets: file rotations, timing, node context
- Run the same packets through Arkime again and you get a *new* capture — new session IDs, new timings
- Not a bug — you told Arkime to capture, and it captured

Rewind: why we reconstructed PCAPs at all

- Arkime viewers kept fetching the same documents from the files index — over and over
- Every session in the UI means PCAP lookups, and mostly the same hot file documents
- Elasticsearch saw load that scaled with *viewing*, not with capture
- We wanted a cache in front of those lookups, close to where the PCAPs live

live-saver: session PCAPs on demand

- Small HTTP service on every capture node, next to the capture process
- File-doc lookups hit a small in-memory cache — the cache that started all this
- PCAP Proxy serves the saved session PCAPs over HTTP for the next stage



POST a session, get back the session with a pcapPath

Arkime already writes the map

- Every packet's position and length rides in the SPI: packetPos + packetLen
- File documents only name the raw capture files — file ID to file name, resolved through a cache
- The positions use gap0, Arkime's compact encoding — which we implemented *three times* before getting it right 🙄
- Reading a session back out is decode, seek, copy — no parsing, no re-capture
- We follow the map Arkime already drew

From cache to long-term storage

- live-saver gave us per-session PCAPs on cheap local disk
- The obvious next step: store them durably, index them there
- So we pointed Arkime at the stored PCAPs — file mode — and indexed into the long-term cluster
- ...you have already seen how that ended

The insight: don't re-capture — repack

- The live cluster already minted the perfect sessions — IDs, timings, SPI fields
- Long-term needs the *same documents*, pointed at *new files*
- live-saver's saved session PCAPs are mostly small and number in the millions — keeping them as-is is a many-small-files problem
- So aggregate session PCAPs into larger multi-session PCAPs — the exact shape Arkime capture writes naturally
- To the viewer, a packed file is indistinguishable from a real capture's file

Reprocess vs repack

Reprocess — our first attempt

Stored
PCAPs



Arkime
capture
file-reading
mode



Long-term
ES
new session
IDs ×
new timings
×

Repack — what we shipped

Session
PCAPs

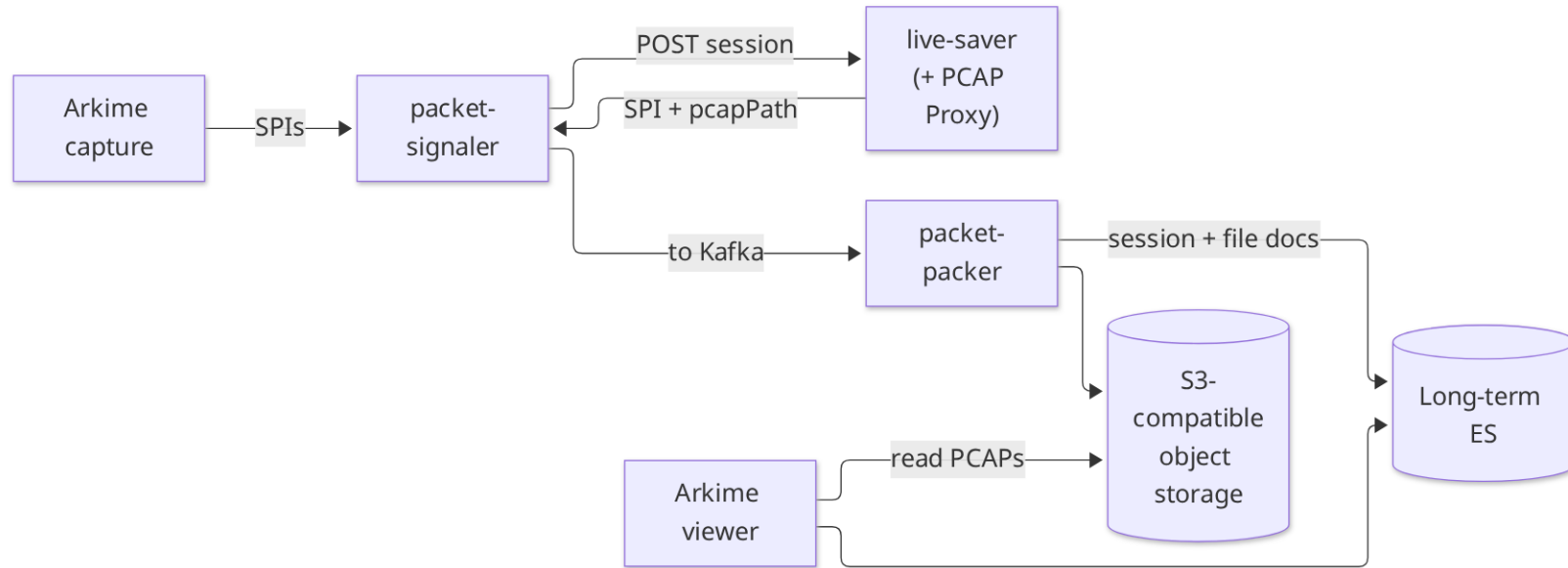
Original SPIs
session IDs
✓ timings ✓

packet-
packer

Long-term
ES
same
sessions,
re-pointed
at packed
files

Session identity survives the move

The pipeline today



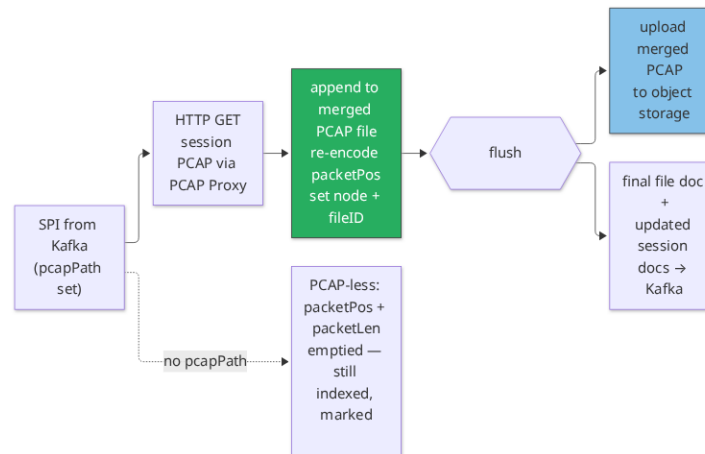
- All three tools are Kafka stream processors — the same pattern as our 2023 SPI processors
- Only live-saver touches the capture nodes; everything else runs in the compute pool
- Two Elasticsearch clusters: capture (files index) and long-term (sessions + files)

packet-signaler: the orchestrator

- Consumes the filtered SPI stream and POSTs each session to live-saver on the owning capture node
- Routing comes from the SPI itself — node name plus cluster DNS; no registry, no configuration
- 409 — capture still writing the packets — backoff and retry
- Give up, and the session is still indexed, just PCAP-less
- 404 — raw PCAP already rotated away — drop it, and count it per node

packet-packer: sessions into Arkime-shaped files

- Streaming: one pass per SPI — decode, fetch, append, re-encode
- Session IDs and timings untouched — only the PCAP references change
- One routing edit: the bulk `_index` is re-bucketed — cut from the session ID
- PCAP-less sessions: `packetPos`/`packetLen` dropped — the viewer will not try loading a PCAP



Same sessions, new files

The result

- Same session IDs, same timings, same SPI fields — on both clusters
- A hunt written against the live cluster replays against long-term
- To the viewer, long-term is just another Arkime cluster
- The PCAPs are addressable objects now — the reuse vision from 2023 is back on the table

What we learned

- Session identity is minted at capture time — never re-capture what you can re-point
- Work with Arkime's grain: multi-session PCAP files plus file docs
- Small tools, one job each, wired together by Kafka — the 2023 recipe, still winning
- Long-term should be indistinguishable from a capture cluster

Questions?

- live-saver · packet-signal-er · packet-packer — plus PCAP Proxy
- The 2023 sketch, delivered — session identity intact

Thank you

Owen McGill

Appendix: implementation details

Optional slides, not part of the talk — here for the Q&A. The gore: the packet map, live-saver internals, signaler backoff, packer internals, session time buckets.

Appendix: the packet map

- packetPos rides in the SPI: a negative value is a negated file ID — all following packets come from that file
- Any other value: the packet's absolute byte offset within the current file
- Zero: "same gap as the previous packet" — run-length compression
- packetLen: the parallel array of packet lengths
- File documents only name the files — no positions there
- live-saver re-encodes positions for the single-session file; packet-packer re-encodes them again for the merged file

Appendix: live-saver internals

- singleflight: concurrent requests for the same session share one reconstruction
- 409 — premature EOF: capture is still writing the packets, retry later
- 404 — raw file rotated off disk, or the file doc is gone from ES
- 303 cross-node redirect exists but we run with it off: routing is SPI plus DNS, so a 303 means a misroute, and the signaler treats it as an error
- Cleaner: watches disk and inode usage; removes the oldest saved hour-folders past the threshold

Appendix: packet-signaler internals

- Bulk format in, bulk format out — no new Kafka schema
- Node name from the SPI, replica suffix stripped, plus the cluster DNS suffix
- 409 backoff: 1 s doubling, gives up once the next wait would pass 20 s — about five attempts
- Backoff exhausted → SPI produced without pcapPath: still indexed, PCAP-less
- 404 or any other error → SPI dropped, counted per node in `pcaps_dropped_total`

Appendix: packet-packer internals

- Partial file doc sunk at file creation — sessions never reference a missing file
- Failed flushes requeue, max three retries, then `uploads_failed_total`
- PCAP-less sessions still indexed: `packetPos` + `packetLen` emptied, node marked — the emptied `packetPos` announces to the viewer not to attempt loading a PCAP in the session detail view
- Object layout: `bucket / date / hostname / shard-date-fileID.pcap` — the shard prefix spreads objects to avoid hot-spotting
- File docs in the Arkime files index with ID `hostname-fileID`; sessions in time-bucketed indices

Appendix: session time buckets

- The packer rebuilds the bulk action's `_index`: `<prefix>sessions3-<timebucket>` — `_id` stays the session ID
- Arkime session IDs begin with the capture time: `250131h10-...` is year, month, day, hour
- The bucket is cut from the ID, never from the clock — a session packed hours late still lands in the bucket it was captured in
- Granularity is one env var: hourly `250131h10`, daily `250131`, weekly `25w05`, monthly `25m01`
- Long-term inherits capture's own `sessions3-*` index layout — hunts, aliases and retention keep working unchanged