draft-prz-lsr-hierarchical-snps-01 Optimizing CSNP Load at Scale

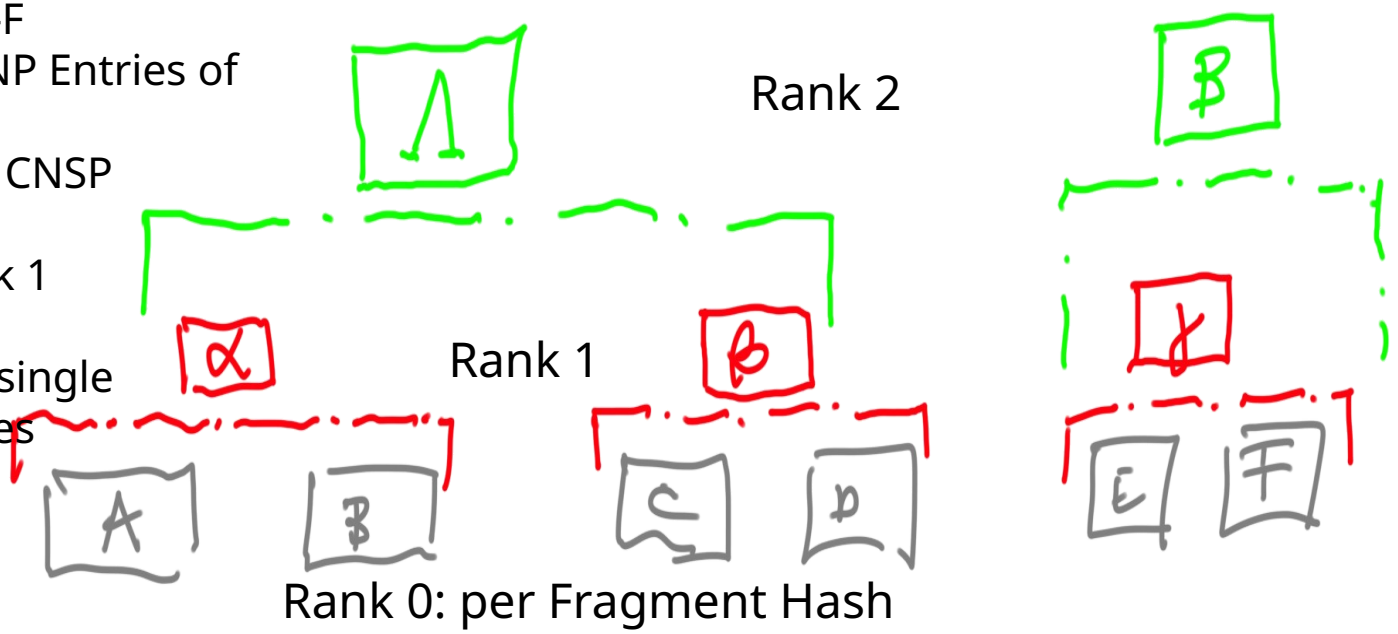
Tony P, HPE of former Juniper

What’s the Problem?

- CNSPs at massive LSDB size and lots adjacencies stress
  - Computing
  - I/O
- On stable network with correctly working flooding basically 100% of that periodic load is useless overhead
  - Well, it’s actually useful (very) if flooding ever hiccs up ;-)
  - Caching helps but not with I/O and it’s tricky to do when CSNP ranges within packets shift (re-packing)

Recycle of old idea with ISIS specific twists

- Merkle Trees published ‘87
- Tree of hashes subsuming lower rank (level/degree) hashes
- We call the bottom rank 0 for now
- Rank 0 are basically full CSNP entries A--F
- Rank 1 α-ɣ will be hash over a set of CSNP Entries of the Nodes
  - XOR’ing of **Fletcher Checksums** of CNSP entries (fragment-id, csum, seq#)
- Rank 2 ‘A—B will be hash over set of rank 1 hashes
- Top rank will end up in extreme case as single hash ‘Ω but normally probably few hashes
- Packets carrying all this will be called Hierarchical SNPs



Harder Part: Partitioning

- CSNP do communicate absence, not only presence
  - Only detectable if 2 ranges of elements can be compared
  - Range implies ordering (ideally total ordering)
- Back to sqrt(1), we need CSNP like ranges for HSNP checksums
- Given limited packet size there will be multiple HSNP packets in flight, each with a range in it
- Gotcha: Strong cryptographic hashes only easy to compare if ranges match

Bag of tricks

- HSNP ranges go from system-ID to system-ID (including pseudo nodes, thanks Les ;-)
- Infeasible to ”synchronize ranges with peers”
  - It would precondition a complex protocol to ensure e’one’s on the same page, basically partition sync
- Everything above rank 0 is using XOR of the sets (only known practical self-inverse hash function)
  - Trivial to keep cheap “on the fly per system-ID” hash value during LSDB changes
  - Possible to “recalculate mismatched ranges” easily
    - E.g. received range is somewhat “longer”, just XOR in the system-ID hashes until range hit
- Loosely packing
  - Put several system IDs into the same “range” up to 50% of amount generating a CSNPs “packet”
  - Slide ranges are similar to fragments, i.e. on “packet size overflow of CSNPs” move system IDs to next higher range and so on recursively on “overflows”

Synchronization

- Nodes exchange in hellos maximum rank they are willing to support
- In the perfect of all worlds normal exchange has 1 packet of high order with few hashes neighbors agree on
- Less perfect world will mismatch one of the hashes in N+1 rank
  - Node detecting responds with N rank packets covering the range in mismatched
  - Recursive “ping pong” until the slice of the database degrades into CSNPs and resulting flooding
    - This will need some kind of pacing mechanism
    - And some prevention from both sides going into disaggregation frenzy

Further Considerations

- Should we use standard CSNPs right after adjacency bring up
  - Less efficient, safer
- Something will need to be said about broadcast adjacencies
- **We will move to Fletcher to hash down a single fragment fingerprint based on performance and randomness quality measurements**

But “WHAT ABOUT” Hashes Not Being Perfectly Collision Free -> Some Quick & Dirty Probabilities Math with `genius`

- ISIS today is actually not –completely- reliable as far CSNPs go ;-)
  - Restart of router choosing same fragment ID and checksum colliding on new data
    - Probability is 1 : 65E3 (size of 16 bits fletcher) so 1E-5
    - Can persist up to 65E3 seconds
- Let’s talk about “probablility times worst case duration” as a “disaster metric”
- So, ISIS today has a 1E-5 * 65E3 or roughly a ”0.6” disaster in the protocol today
- And BTW, fletcher checksum is not perfect either, a 2 bit flip can slip it and the problem persist for lifetime of a fragment

HSNP Probabilities and ”Disaster Metrics”

- Let’s look at 2 levels maximum #fragments in a hash
  - Level 1 -> around 70 fragments (to trigger a single CSNP on mismatch of a hash)
  - Level 2 -> 80 entries of Level 1 (1 Packet covers then 80 * 70 ~ 5E3 entries)
  - Level 3 -> 80 * Level 2 (1 Packet covers 0.5E6 entries)
- What’s the HSNP hash packet disaster metric (two same hashes)
  - Birthday paradox https://en.wikipedia.org/wiki/Birthday_problem
  - Fragments refresh so we can expect a single hash to persist for 65K/#fragments seconds (it’s bits optimistic)
  - Level 1 = 2.8E-7 * 65E3/70 ~ 1E-4 disaster
  - Level 2 = 2E-3 * 65E3/(70*80) ~ 2E-2 disaster
  - Level 3 = 1 * 65E3 / 0.5E6 ~ 0.1 disaster
- So, “disaster metric” is still about 5x better for 0.5E6 fragments than what we have today in the protocol ;-)

What if we want to buy better metrics

- With full 64 bits fletchers and hashes we improve disaster metrics
  - .e.g for 0.5E6 fragments: 6.8E-9 * 65E3 / 0.5E6 ~ 8.8E-10 ~ basically ”nothing” ;-)
  - Because of much less dense packing probabilities drop to “basically nothing of nothing” ;-)
- With 40 bits fletcher (we’d need to invent it ;-)
  - For 0.5E6 fragments: 5E-2 * 65E3 / 0.5E6 ~ 6E-3
- Given a 0.1 disaster on 0.5E6 fragment we could mandate ”MUST send full CSNPs every 15 minutes”
  - 0.5E6 fragments is however around 6E3 CSNPs and with 10/sec we are looking at 10 minutes transmission time so we gain nothing really
- We could vary the seed of the fletchers but that doesn’t shift probabilities in 0.5E6 fragments case and generates huge amount of computation load

And Latest Thinking Based on Implemenation

- Levels seem unnecessary
- An implementation is free to choose any packing based on any strategy feasible
  - Less packing density for nodes that “oscillate fragments often”
  - “Maximum packing” on startup of adjacency after a link flap
  - “Minimum packing” when adjacency is being brought up first time
- Only constants seem to be
  - “once you hit hash mismatch on Level 1 ~ 70 fragments a CSNP MUST be sent”
  - “any mismatch covering more than 70 fragments MUST generate something with lower level of packing for the mismatched range” which kind of implies the first rule