Tips for designing protocols using `libminisketch`

Sending a sketch is less efficient than just sending your whole set with efficient entropy coding if the number of differences is larger than *log2(2b choose set_size) / b*.

In most applications your set can be hashed to entries just large enough to make the probability of collision negligible. This can be a considerable speedup and bandwidth savings. Short hashes (<128 bits) should be salted with an unpredictable value to prevent malicious inputs from intentionally causing collisions. Salting also allows an entry missed due to a collision to be reconciled on a later run with a different salt. Pre-hashing may not be possible in some applications, such as where there is only one-way communication, where the confidentiality of entry origin matters, or where security depends on the total absence of collisions.

Some element sizes are faster to decode than others; see the benchmarks in the readme.

Almost all the computational burden of reconciliation is in minisketch_decode(). Denial-of-service attacks can be mitigated by arranging protocol flow so that a party requests a sketch and decodes it rather than a construction where the participants will decode unsolicited sketches. Decode times can be constrained by limiting sketch capacity or via the max_count argument to minisketch_decode().

In most cases you don't actually know the size of the set difference in advance, but often you know a lower bound on it (the difference in set sizes).

- There are difference size estimation techniques such as min-wise hashing[1] or random projections[2], but complex estimators can end up using more bandwidth than they save.
- It may be useful to always overestimate the sketch size needed to amortize communications overheads (*e.g.* packet headers, round trip delays).
- If the total data sent would end up leaving you better off having just sent the whole set, per above, then you can send the set in response to a failure but leave out as many elements as the size of the previously sent sketch. The receiver can decode the partial set and use the data they already have to complete it, reducing bandwidth waste.
- Additional elements can be sent for a sketch as few as one at a time with little decode cost until enough data is received to decode. This is most easily implemented by always computing the largest sketch size and sending it incrementally as needed.
- Because sketches are linear you can adaptively subdivide to decode an overfull set. The sender uses a hash function to select approximately half their set members and sends a sketch of those members. The receiver can do the same and combine the result with the initially sent sketch to get two sketches with roughly half the number of members and attempt to

decode them. Repeat recursively on failure. This adaptive subdivision procedure makes decode time essentially linear at the cost of communications inefficiency. Minisketches can also be used as the cells in an IBLT for similar reasons.

Less efficient reconciliation techniques like IBLT or adaptive subdivision, or overheads like complex estimators effectively lower the threshold where sending the whole set efficiently would use less bandwidth.

When the number of differences is more than 2b/2-1 an alternative sketch encoding is possible that is somewhat smaller, but requires a table of size 2b; contact the authors if you have an application where that might be useful.

References

- [1] Broder, A. *On the Resemblance and Containment of Documents* Proceedings of the Compression and Complexity of Sequences 1997 [PDF]
- [2] Feigenbaum, Joan and Kannan, Sampath and Strauss, Martin J. and Viswanathan, Mahesh. *An Approximate L1-Difference Algorithm for Massive Data Streams* SIAM J. Comput. 2003 [PDF]