Randomized Scrooge Ideas

Assume we are sending messages between cluster 1 of size N_1 and cluster 2 of size N_2 where each cluster has f_1 and f_2 faulty (byzantine or crash) nodes respectively.

Implementation Ideas

Generally, we have been implementing a deterministic version of scrooge where we send messages totally determined by message sequence ids. This will look at how randomization can help to reduce latency (and handle proof of stake) and also how privacy can help with dynamic adversaries. To get a ballpark number of each strategy's performance, let $N_1 = N_2 = 73$ and let $f_1 = f_2 = 24$, the randomized strategies variables are independent of network size, but I picked this based on the randomized ack counting constants. The randomized strategy variables only depend on the ratio of faulty to honest nodes.

Deterministic Scrooge: WLOG assume that we are sending messages from network 1 to network 2.

- The j-th retry of message i is sent to node $(i+j)\%N_2$ by node $(i+j)\%N_1$
  - Each message contains $\{Data, AckId\}_{NetworkSig}$ so the receiver can verify it should receive the message and broadcast it
- A message i is observed as being received after $f_2 + 1$ nodes acknowledge message i.
- There are at most $(f_1 + f_2 + 1)$ senders of each message
  - The j-th retry will only be sent after $(f_2 + 1) \cdot j = 25$ messages fail to acknowledge message i
  - This gives a worst case latency of $(f_2 + 1) \cdot (f_1 + f_2) = 14400$ acknowledgements (quadratic in network size)
    - This number is somewhat high, but helped by cumulative acks
- Works against a static adversary

Randomized Scrooge: WLOG assume that we are sending message from network 1 to network 2.

- On Protocol start, each network coordinates to generate a random seed
- The j-th retry of message i is sent to node $RNG_{seed}(2j + 1 + 2ik_{send})\%N_2$ by node $RNG_{seed}(2j + 2ik_{send})\%N_1$
  - Each message contains $\{Data, AckId, seed\}_{NetworkSig}$ so the receiver can verify it should receive the message from the sender and broadcast it
- A message i is observed as being received after $k_{ack} = 24$ nodes acknowledge message i.
  - This has a $> 99.999999999\%$ chance of being correct which is the same chance of S3 durability
- There are at most $k_{send} = 44$ senders of each message

    - The j-th retry will only be sent after k_{ack} messages fail to acknowledge message i
    - This gives a worst case latency of $k_{ack} \cdot k_{send} = 1056$ sends (constant in network size)
    - This has a $\sim 99.999999999\%$ chance of delivering the message to an honest node (I forgot to calculate this probability in tandem with probabilistic ack counting)
- Works against a static adversary

Randomized Private Scrooge

Even with privacy, we still need to have (with high probability) 44 pairs of senders+receivers of each message. This can probably be done with the same technique using VRFs as the algorand block proposal selector algorithm. Nodes would then receive a proof that the sender was elected to send message i and that the sender's VRF would've sent the message to the receiver.

The ordering of senders can be done in two ways.

1. We can have w.h.p. $\sim k \geq 1$ nodes elect themselves to send the jth retry of message i, where if node x is elected, it sends to $VRF_x(j + ik_{send})\%N_2$
    - This needs more crypto operations than option 2
    - There is perfect ordering of resenders, but the average number of selected pairs is likely much higher than 44 as I don't think we can elect exactly one node for each resend? The number of elected responders should be normally distributed fwiw
    - INSIGHT: This is really just trying to do private but verifiable (approximate) leader election in a trustless environment 44 times in a row. Maybe there is theory we can use?
2. We can have w.h.p. $\sim k \geq 44$ nodes privately elect themselves at once to send message i, and then if node x is selected, it treats itself as the $((i + k_{send} * x/N_1)\%k_{send})$th resender of the message.
    - The standard error of this ordering is 4.7 positions for each sender which means probably there may be multiple (probably like 4-10) honest nodes sending out the same message at once.
    - Somewhat poor ordering, but the number of selected pairs is probably closer to 44?

If we can then pick/order 44 pairs of senders/receivers of each message then we have the same latency as the above Randomized Scrooge section, but since the selection is private we can withstand a dynamic adversary.

General Analysis

When running scrooge to send messages form network 1 to network 2, it will pick multiple pairs of nodes $(i, j) \in \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}$ to transfer messages. In general a message will fail to be delivered if all pairs (i, j) had at least one faulty node. For this analysis in the deterministic case, I assume the adversary is static but private. In the randomized case, the adversary may be dynamic.

Deterministic Worst Case: The minimal sized set of pairs $(i, j) \in \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2}$ that is guaranteed to have at least have at least one faulty node are $f_1 + f_2 + 1 \leq \min(N_1, N_2)$. Note that if the inequality doesn't hold then this quantity is insufficient.

Proof: We can construct a set of $m < f_1 + f_2 + 1$ faulty pairs of nodes. For any fixed list of pairs, $(v_1, u_1), \ldots, (v_m, u_m)$, we have $|\{v_i \mid i \leq f_1\}| \leq f_1$ and $|\{u_i \mid f_1 < i\}| \leq f_2$. Therefore we can label all the nodes in both sets as faulty which then makes all pairs in the list have at least one faulty node. Then the list $(0, 0), (1, 1), \ldots (f_1 + f_2, f_1 + f_2)$ is guaranteed to have at least one good pair as otherwise there would be at least $f_1 + f_2 + 1$ faulty nodes between the clusters.

Randomized Failure Probability: At any instance in time, the total number of faulty pairs is $F = f_1 N_2 + N_1 f_2 - f_1 f_2$, the probability a randomly picked pair is faulty is then $\frac{F}{N_1 N_2}$. The chance that K pairs chosen at random are all faulty is $\left(\frac{F}{N_1 N_2}\right)^K$, otherwise stated the number of pairs chosen needed to provide a success rate of p is $K \geq \log_{F/(N_1 N_2)}(1 - p)$. For example, Amazon's S3 has durability of 99.999999999%, this same success probability can be achieved with $K \geq 44$ regardless of total network size if $N_1 = N_2$ and $f_1 = f_2 = N_1/3$ (note that these parameters also make $\left(\frac{F}{N_1 N_2}\right) = 5/9$).

Proof: At any instance of time, we can calculate the number of faulty pairs with inclusion exclusion. There are $f_1 \cdot N_2$ pairs with a faulty node in the first entry, $N_1 \cdot f_2$ with a faulty node in the second entry, and $f_1 \cdot f_2$ with faulty entries in both. The total number of total faulty pairs is then $F = f_1 N_2 + N_1 f_2 - f_1 f_2$.

Deterministic Ack Counting: When a node in network X wants to know with certainty that message i was delivered, it waits for $f_{1-X} + 1$ messages that acknowledge message i. This is the smallest set that is guaranteed to have at least one honest node that actually received message i.

Randomized Ack Counting: When a node in network X receives a message that was delivered to it at random (verify using a VRF), there is a f_{1-X}/N_{1-X} chance it was sent by a malicious node. For L many acknowledgements, the chance they were all malicious is $(f_{1-X}/N_{1-X})^L$. In other words, the number of messages needed to maintain some probability p of hearing from at least 1 honest node is $L \geq \log_(f_{1-x}/N_{1-X})(p)$. To match amazon's S3 durability of 99.999999999%, only 24 acks are needed regardless of network size if $f_{1-X} = N_{1-X}/3$.

Randomized Sender Ordering: If we pick on average K pairs of sender/receivers privately at random, can we still have some coordination between resends? As in ideally the ith resender only sends a message if the "first" $i - 1$ pairs were malicious, but the privacy makes it unclear who was selected, yet alone who's selected first. The best strategy is if the i-th node is privately at random selected from network 1 as a sender, treat it as the $(i \cdot K/N_1)$-th resender of the message. This strategy then has standard error of $K \cdot (iN_1 - i^2)/N_1^2 \leq \sqrt{K/2}$ which is about 4.7 for $K = 44$. Load balancing can be achieved by shifting responsibility each message, as in, the 5th sender of the 5th message is actually the first sender. In general, node i is sender $(j + i \cdot K/N_1)\%K$ of message j.

Proof: For all K of the selected senders, there is an i/N_1 chance the selection precedes node i (assuming nodes are numbered $0 \ldots N_1 - 1$). This makes the position of node i a Binomial random variable with mean $K \cdot p = K \cdot i/N_1$ and variance $K \cdot (iN_1 - i^2)/N_1^2$. The variance is maximized when $i = N_1/2$ which yields variance of $K/2$ or standard deviation of $\sqrt{K/2}$.