Propagation of mix decoy loop statistics

David Stainton
Eva Infeld
Leif Ryge

Abstract

In the context of continuous time mixing stategies such as the memoryless mix used by Katzenpost, n-1 attacks may use strategic packetloss. Nodes can also fail for benign reasons. Determining whether or not it's an n-1 attack is outside the scope of this work.

This document describes how we will communicate statistics from mix nodes to mix network directory authorities which tells them about the packetloss they are observing.

Table of Contents

Terminology

The following terms are used in this specification.

Wire protocol	Refers to our PQ Noise based protocol which currently uses TCP but in the near future will optionally use QUIC. This protocol has messages known as wire protocol `commands`, which are used for various mixnet functions such as sending or retrieving a message, dirauth voting etc. For more information, please see our design doc: wire protocol specification [https://github.com/katzenpost/katzenpost/blob/main/docs/specs/wire-protocol.md]
Providers	Refers to a set of node on the edge of the network which have two roles, handle incoming client connections and run mixnet services. Soon we should get rid of `Providers` and replace it with two different sets, `gateway nodes` and `service nodes`.
Epoch	The Katzenpost epoch is currently set to a 20 minute duration. Each new epoch there is a new PKI document published containing public key material that will only be valid for that epoch.

1. Design Overview

Nodes (mixes, gateways, and providers) need to upload packet-loss statistics to the directory authorities, so that authorities can label malfunctioning nodes as such in the consensus in the next epoch.

Nodes currently sign and upload a Descriptor in each epoch.

In the future, they would instead upload an UploadDescStats containing:

- Descriptor

- Stats
- Signature

Contains a map from pairs-of-mixes to the ratio of count-of-loops-sent vs. count-of-loops-received.

2. Tracking packet loss and detecting faulty mixes

Katzenpost lets different elements in the network track whether other elements are functioning correctly. A node A will do this by sending packets in randomly generated loops through the network, and tracking whether the loop comes back or not. When it comes back, it will mark that as evidence, that the nodes on the path of that loop are functioning correctly.

Experimental setup, node A:

- Data: each network node `A` collects a record of emitted test loops in a certain epoch, their paths and whether they returned or not. Importantly, each loop is the same length and includes l steps.
- A segment is defined as a possible connection from a device in the network to another, for example from a node in the layer `k` to a node in the layer `k+1`. Each loop is a sequence of such segments.
- Each node `A` will create 3 hashmaps, `sent_loops_A`, `completed_loops_A` and `ratios_A`. Each of these will use a pair of concatenated mixnode ID's as the key. The ordering of the ID's will be from lesser topology layer to greater, e.g. the two-tuple (n, n+1) which is represented here as a 64 byte array:

```
var sent_loops_A map[[64]byte]int
var completed_loops_A map[[64]byte]int
var ratios_A map[[64]byte]float64
```

- Every time the node A sends out a test loop, for each segment in the loop path, it will increment the value in `sent_loops_A`.
- When a test loop returns, for each step in the loop path, it will increment the value in `completed_loops_A`.
- Generate a new map entry in `ratios_A` for each mix-node-pair `p`, if `sent_loops_A[p]==0` set `ratios_A[p]=1`. Else `ratios_A[p] = completed_loops_A[p]/sent_loops_A[p]`
- Plot the resulting distribution, and calculate the standard deviation to detect anomalies. Have the node report significant anomalies after a sufficient time period as to not leak information on the route of individual loops.
- Anomalies may have to be discarded if the corresponding `sent_loops_A[p]` is small.

You would expect the distribution of values in `completed_loops` to approximate a binomial distribution. In an absence of faulty nodes, `ratios` should be 1, and when there are some faulty nodes values at faulty nodes should approach 0 (if the node doesn't work at all), and be binomially distributed at nodes that can share a loop with faulty nodes.

Therefore each mix node generates a statistics report to upload to the dirauth nodes, of the struct type:

```
type LoopStats struct {
Epoch           uint64
MixIdentityHash *[32]byte
Ratios          map[[64]byte]float64
}
```

The report is subsequently uploaded to the directory authorities, which combine the reports of individual nodes into a health status of the network and arrive at a consensus decision about the topology of the network.

3. Uploading Stats to Dirauths

Stats reports are uploaded along with the mix descriptor every Epoch. A cryptographic signature covers both of these fields:

```
type UploadDescStats struct {
Descriptor []byte
StatsReport []byte
Signature []byte
}
```

Statistics reports collected during the XXX period of time, that is, the time between descriptor N +1 upload and descriptor N+2 upload, are what will affect the topology choices in epoch N+2 if the dirauths collectively decide to act on the very latest statistics reports in order to determine for example if a mix node should be removed from the network:

```
| ---------------- epoch N ---------------- | ---------------- epoch N+1 ------
| ----------- UD_N+1 ---------------------- | ------------ UD N+2 -------------
```