POP^4: Probabilistic On-Packet Per-Path Predicate Profiling

{ Michael Colavita, Harvard University; Luke Melas-Kyriazi, Harvard University; Garrett Tanzer, Harvard University }

Abstract

Detecting faults along a packet's path using in-network telemetry traditionally requires fixed-length metadata to be attached to the packet. Because this structure is ill-suited to communicating information of variable length, a refinement of the approach called *probabilistic* in-network telemetry has emerged, which distributes this information across multiple packets in the same flow. We propose an algorithms that allows each switch on a packet's length k path to report a Boolean value, which may represent a self-detected failure, across $\Theta(k)$ packets using $O(k \log^2 k)$ computation time at the end host. We derive this algorithm as an interpolation of two algorithms that require $\Theta(k)$ packets & $O(k^3)$ reconstruction time and $\Theta(k \log k)$ packets & $\Theta(k \log k)$ reconstruction time, using a novel analysis of the (n, m) coupon collector's problem (collecting m coupons for each of n categories), in which nm is held constant. We experimentally validate our results with simulations, publicly available at pop4.

1 Introduction

Detecting and collecting information about faults is an important aspect of monitoring and managing large data centers. Traditionally, discovering and transmitting such information without the intervention of the control plane has imposed a significant burden on the data plane's normal operating capacity.

A naive approach to in-network telemetry (INT) places actionable information within fixed-length header fields on a single packet, allowing the end host to recover this information after receiving a single packet. However, this approach scales poorly when the information that must be transmitted is variable length, e.g. proportional to the path length, and packet headers become bloated with fields that are often left unused. Recent developments in path profiling [7, 6, 11, 25, 3] have advanced this paradigm, introducing a new approach called *probabilistic* INT, in which this variable-length information is distributed in small fields across multiple packets.

These works mainly focus on the problem of recovering the path across switches that a packet in a flow traverses in the network. In contrast, our work focuses on the high-level problem of communicating Boolean values from switches along a known path to the end

host. A practical instantiation of this problem occurs when switches are able to self-detect certain kinds of faults, and must communicate these to the network administrator in order to have them resolved. When an arbitrary number of switches along a particular path might experience these faults, it is difficult to communicate the necessary information using the deterministic INT framework.

Taking inspiration from prior works in probabilistic INT like POP^3 [3] and from constructions in network coding, we propose a new algorithm for fault detection (and more generally, the problem of communicating Boolean predicates on switches) that uses $1 + \log\log k$ bits in each of $\Theta(k)$ packets, and requires $O(k \log^2 k)$ computation time at the end host. We present this algorithm as a particularly interesting interpolation of two more naive algorithms, achieving 1 bit in each of $\Theta(k)$ packets & $O(k^3)$ reconstruction time and $1 + \log\log k$ bits in each of $\Theta(k \log k)$ packets & $\Theta(k \log k)$ reconstruction time respectively. We characterize the tradeoff space between these extremes of communication cost and reconstruction cost using a novel analysis of the (n, m) coupon collector's problem (collecting m coupons for each of n categories), when nm is held constant.

This paper is structured as follows: In Section 2, we review related work path profiling—honing in on the probabilistic INT framework of POP^3—as well as highlighting some lessons from random linear fountain codes. In Section 3, we present the two algorithms at the boundaries of our tradeoff space, optimized for communication cost (i.e. number of packets received) and reconstruction time. In Section 4, we generalize and interpolate these algorithms to form the aforementioned tradeoff space, and analyze the performance at several key parameter configurations. In Section 5, we validate our theoretical results through simulation; our code and resulting data & visualizations are publicly available at pop4.

2 Related Work

Path profiling, fault detection, and network telemetry have been the subject of extensive literature in the networking community. These works address a variety of different tasks (e.g. debugging high-latency flows or profiling operator-specified flows) and operate under different sets of assumptions (e.g. whether the length of the flow is known, or whether the flow path is subject to change randomly). Since our paper builds upon work in path profiling, in this section we review popular approaches to path profiling and related tasks in network telemetry. In the following section, we give a detailed explanation of the POP^3 path profiling algorithm [3]. We then discuss work related to our methods of analysis, including fountain codes and the Double Dixie Cup problem.

Path profiling, the task of identifying the paths traversed by packets in a network, is an essential component in many network monitoring and debugging systems. One line of work in path profiling focuses on identifying and debugging specific flows selected by network operators [23, 26]. These include [26], which proposes a packet-level telemetry system for network administrators to track packets affected by faults, and [23], which presents a system for fine-grained monitoring of traffic flows based on operator queries. These approaches limit themselves to a subset of all flows, so they are not equipped to quickly address malicious

flows or cope with high-latency flows.

Duffield and Grossglauser [5] propose measuring traffic for all flows by randomly subsampling packets. They used a hash function of the packet content to determine whether to sample the packet, sharing the same hash function across the network in order to collect trajectory samples. However, sampling a fraction of all flows struggles to scale and quickly detect faults when the size of the network and number of packets grows large.

Other path profiling techniques that attempt to profile all flows generally take one of three classes of approaches. One class of approaches mirrors packets directly at the switches [10]. These methods struggle to scale to large datacenters because they can quickly introduce excessive quantities of additional (mirrored) traffic to the network. For example, trace analysis via mirroring on a $100Tbps$ network with CPUs that process traffic at line rate (10 Gbps) would consume $3,200$ CPUs [26].

A second class of approaches stores information on the switches and periodically hands it off to a global collector, which aggregates information on all switches [21]. Although this idea mitigates some of the problems of traffic mirroring, the limited memory of network switches means that collectors have to collect frequently, potentially as often as every 10ms. If the collectors collect at such a high frequency, we may run up against the issues of bandwidth and processing similar to the mirroring approach.

Finally, a growing number of methods store profiling information on the packets directly [7, 3, 20, 21, 11, 25]. These methods have the disadvantage of taking up bits that could otherwsie be used for packet contents, but if the number of these overhead bits is small they avoid the high computing and bandwidth cost of the other approaches.

Under the In-Network Telemetry (INT) framework [7], switches can write their ID onto packet headers as they process them. The last switch can then read off the path information for each packet. The primary drawback of INT is that it takes up a non-trivial amount of space on the packet header: 8 bytes to write the INT header and 4 bytes per hop to write the switch ID. For a path of length 5, for example, this takes 28 bytes of a packet of total size around $1KB$.

Related work on IP traceback introduced the idea of switches selectively sampling packets on which to write their switch ID [6]. However, as these systems are designed to identify attacks with many origins, they need a large number of packets (often over 100), which is suboptimal for path profiling and fault detection.

Inspired by this work on IP traceback, [3] proposed Probabilistic On-Packet Path Profiling (POP^3), a probabilistic variant of INT requiring less space on the packet header.

Under POP^3, the receiver collects path information from multiple packs in a flow and uses the aggregate information to reconstruct the path. As a result, it is possible to profile paths using only a few bits per packet. We describe the POP^3 framework in detail in the following section.

2.1 Background

2.1.1 POP^3

POP^3 aims to achieve path profiling with minimal overhead, high accuracy across network topologies, and easy implementation on commodity programmable switches. It achieves this by only transmitting information about one switch per packet and enabling the receiver to reconstruct the entire flow path from a sequence of packets in the same flow.

The algorithm has two phases: (1) processing—the encoding process that occurs on switches—and (2) reconstruction—the decoding process performed by the receiver once N packets have been received. Both exploit the presence of two globally predetermined hash functions g and h in order to avoid communication.

During processing, at at each hop in the path, a switch writes its fingerprint (a hash of the switch ID and the packet using g) on the packet with probability r_i, sampled by treating a hash of the switch ID and hop number using h as uniform. If a previous switch digest is already present, it is overwritten. By setting $r_i = 1/i$ à la reservoir sampling, the algorithm ensures that probability of each of the switches in the path being the final switch written on the packet is the same: $1/k$, where k is the number of hops in the path.

During reconstruction, the receiver effectively reverse-engineers the writing procedure for the fingerprints it has received, using the fact that it knows g, h, k, and the universe of switch IDs. The details of the reconstruction algorithm are not directly relevant to POP^4.

Ben-Basat et al. [3] shows that after $N = O(k(\log(k/\delta) + \log(|V|)/b))$ packets have been received, where $|V|$ is the number of switches and b is the number of bits in the digest (the switch hash), the POP^3 algorithm recovers the correct path with probability $1 - \delta$. Ben-Basat et al. [3] shows this result by transforming the analysis into a Dixie Cup problem (explained below).

Our algorithm builds off POP^3 general framework but tackles a different problem. Rather than reconstruct the path, we recover the faults on a flow given that the path has already been found (with POP^3 or another method). Like POP^3, we sample switches probabilistically, write information onto the packet header, and perform all non-trivial computation at the receiver. Whereas POP^3 employs reservoir sampling to obtain a uniform distribution over packets, we use random linear combinations, a technique inspired by fountain codes.

2.1.2 Analysis

Our analysis of the communication-optimal algorithm below is inspired by techniques from coding theory, notably fountain codes [14]. Fountain codes are a class of randomized codes designed for transmitting information over erasure channels. Traditional methods for transmitting a file over such channels deal with packet loss by introducing a backward channel and re-sending dropped packets. Fountain codes provide an alternative in which the sender transmits a (possibly endless) supply of encoded packets ("drops") from a file, each a fraction $1/K$ of the size of the original file. When the number of packets received K' is slightly larger than K, the entire file may be reconstructed by the receiver.

The simplest family of these codes, random linear fountain codes, encodes the K source packets with a random binary matrix G which is known to both the sender and receiver. When the receiver receives K linearly independent encoded packets, the original message may then be decoded by inverting the $K \times K$ matrix G. If the receiver has received N encoded packets, where N is slightly larger than K, then the receiver can find a linearly independent subset of K encoded packets with high probability (detailed below), so the receiver can reconstruct the original file.

The reconstruction phase of our communication-optimal algorithm resembles that of a random linear fountain. We use the result from MacKay [14] that if the number of excess codes received is $E = N - K$, the probability of failure (i.e. fewer than K linearly independent rows) is bounded above by:

$$\delta(E) \leq 2^{-E}$$

In the following section, we use this result to bound the number of packets necessary to detect all faults in a network path.

3 Algorithms

3.1 Problem Setup

We consider a flow F within a network represented as a graph of switches. Let $k = |F|$ be the number of switches in the path. Packets p_j hop along this path; each packet p_j and switch s_i has a unique ID. We assume the path is known to the destination, and we aim to collect Boolean values (predicates) $x_i \in \{0, 1\}$ from the switches s_i along the path. For example, this information may be an alert that something in the switch is faulty.

Finally, we assume that our hash functions, denote h, output independent uniformly random numbers given inputs p_j and s_i. In our case, these numbers are just single bits, but we may use more bits as desired (in which case we would simply determine the x_i more quickly).

Note: In this and the following sections, we will refer to our task as "fault detection" for ease of comprehension and because fault detection is the most immediate use-case of our methods. However, we emphasize that our algorithms may be used to collect any predicate held by the switches.

3.2 Optimizing for communication cost

We first present an algorithm that minimizes communication cost, at the expense of reconstruction cost. At a high level, we use the fingerprints $h(s_i, p_j)$ as a random matrix and solve a linear system of equations to recover the Boolean predicate values. More concretely, the algorithm is:

- *Input*: Stream of packets with 1 extra header bit.
- *Processing:*

1. At each switch s_i that experiences a fault, for each packet p_j, overwrite the header bit b_j with the XOR of $h(s_i, p_j)$ and the current header bit b_j.
2. At the receiver, collect the packet IDs p_j and the header bits b_j.

- *Reconstruction:*
  1. Given N packets (p_j, b_j), construct the $N \times k$ matrix $\hat{A}$ containing each packet-switch hash; that is, $\hat{A}_{i,j} = h(s_i, p_j)$.
  2. Select a linearly independent set of k rows from the N rows of $\hat{A}$, forming a $k \times k$ matrix A. As shown below, this is possible with high probability.
  3. Solve the linear system $Ax = b$ over the finite field F_2.
  4. The resulting vector x corresponds to the faulty switches.

From an algorithmic perspective, we are interested in the number of packets N that must be collected in order to reconstruct the vector x of faulty switches with high probability. We have a clear lower bound on N by an entropy argument. Since there are 2^k configurations of switches, it takes at least k bits to reconstruct the switches.

For an upper bound on N, we note that as proven in MacKay [14], if we send $k+E$ random vectors, the probability of failure is bounded by 2^{-E}. Then we succeed with probability $1-\delta$ if we send $k + \log_2(1/\delta)$ random vectors.

Finally, we can derive the expected value of N. Note that there are 2^k unique vectors that can be produced by this scheme. When the first vector arrives, any vector is linearly independent except for the 0-vector. After j vectors have arrived, there are 2^j linear combinations of vectors that can be produced by the received vectors. Any vector that does not fall into this set is linearly independent. As each vector is produced with equal probability, the probability that a randomly received vector will be linearly independent after j have been received is $1 - 2^{j-k}$. Thus after j vectors have been received, the number of vectors we need to receive before a linearly independent one is encountered is $\frac{1}{1-2^{j-k}}$. By linearity, we have

$$\mathbf{E}[N] = \sum_{j=0}^{k-1} \frac{1}{1-2^{j-k}} = \sum_{j=1}^{k} \frac{1}{1-2^{-j}} = k + \sum_{j=1}^{k} \frac{2^{-j}}{1-2^{-j}} \leq k + \sum_{j=1}^{k} \frac{1}{2^{j-1}} \leq k+2$$

Thus $\mathbf{E}[N] \leq k+2$. For a tighter bound, using the Lambert series [24]

$$\mathbf{E}[N] \leq k + \sum_{j=1}^{\infty} \frac{2^{-j}}{1-2^{-j}} \leq k + \frac{\ln 2 - \phi_{1/2}(1)}{\ln 2} \leq k + E$$

Note that E is the Erdős–Borwein constant, or $E \approx 1.6067$.

3.3 Optimizing for reconstruction cost

Next, we present an algorithm optimized for reconstruction cost. In this setting, we have k switches that we aim to collect information about separately, and we obtain information

about one switch per packet. If we could select a switch uniformly at random for each packet, this process would become a coupon collector's problem with parameter k.

One way to select uniformly at random would be to using reservoir sampling, as in POP^3. However, this approach would not generalize to $k \neq m$ (below). We use a more flexible method, but we note that this method requires $\log \log k$ bits of extra space per packet.

We describe this method in the general setting that we would like to split the switches up into m groups (here $m = k$) of equal size in a deterministic manner. We do so by having the sender send an over-estimate of the number of groups m; that is, we send the smallest integer q such that $2^q \geq m$. This takes $\log \log m$ bits and leads to at most $2\times$ overhead (that is, our estimate $\hat{m} = 2^q$ of m is at most 2 times the true m). When a packet with this information reaches switch s_i, the switch takes its own TTL number modulo the estimated number of groups $\hat{m}$. If this number matches the group index on the packet, the switch writes its information on the packet. This procedure partitions the switches into $\hat{m}$ equally-sized groups, each of which is selected by write on each packet with equal probability.

Note that this does require the sender to know the length of the path, which we assume is known as the path is known to the destination. Also note that path length information is not necessary for $m = 1$; for this reason, it is possible to perform the $m = 1$ algorithm with only 1 bit of total overhead.

4 Trading off between communication and reconstruction cost

We present a family of algorithms with a user-tunable parameter m that generalizes and interpolates between the two algorithms described above. These algorithms are composed of two phases, a processing phase and a reconstruction phase. During the processing phase, switches selectively write onto packets with information about their fault condition. Only $1 + \log \log k$ bits are required on the packet header for $m = \omega(1)$, and only 1 bit is required for $m = \Theta(1)$. During the reconstruction phase, the receiver reconstructs the set of faulty switches using the aggregate information from a set of packets.

We divide the flow into k/m groups of m switches and perform this procedure to each group separately. We can set m high, for example $m = k$, to optimize for communication cost (i.e. the number of packets that must be sent before we can recover all the faults). Conversely, we can set m low, for example $m = 1$, to optimize for reconstruction cost (i.e. the complexity of recovering the set of faults from the received packet information).

At a high level, the algorithms function as follows:

1. Divide the flow into $n = k/m$ groups of switches.
2. For each packet, choose a group p as described in Section 3.2. If switch s_i in group p experiences a fault, calculate $x_{i,j}^{(p)} = h(s_i, p_j)$. Overwrite the bit in the packet header with the XOR of the current bit and $x_{i,j}^{(p)}$.
3. Collect "enough" (defined below) packets $p_1, \ldots, p_N$ and corresponding $x_{i,j}^{(p)}$.

4. For each group p, reconstruct the set of faulty switches by solving a system of linear equations over F_2. Return all faulty switches.

In Section 4, we give an analysis of the coupon collector's problem with fixed $n \cdot m = k$. In Section 4.1.7, we apply this analysis to better understand our algorithm (above) with groups of switches of size m.

4.1 Coupon collector's problem with fixed nm

We now consider a set of related coupon collector's problems that allow us to analyze various communication-reconstruction tradeoffs. Suppose we wish to collect n distinct classes of coupons. For each class of coupons, we wish to collect m of that class. We consider the setting in which $nm = k$ and analyze the asymptotic behavior as k grows to infinity. Let $X(n, m)$ be the number of coupons we need to collect before we have m of each of the n classes. Note that X corresponds to N in the section above.

4.1.1 Fixed $m = 1$

Several variants of this formulation are well-studied. Consider, for example, the setting in which $n = k$ and $m = 1$. This corresponds to the classical coupon collector's problem, in which one coupon must be collected for each of k classes [15]. It is known that:

$$\begin{aligned} E(X(k,1)) &= k \log k + O(k) \\ &= \Theta(k \log k) \end{aligned}$$

4.1.2 Fixed $m \neq 1$

Next, suppose that m is a fixed constant independent of k. We therefore have that $n = k/m$. Newman and Shepp [17] showed that

$$\begin{aligned} \mathbf{E}[X(k/m, m)] &= \frac{k}{m} \log \frac{k}{m} + \frac{k}{m}(m-1) \log \log \frac{k}{m} + O(k/m) \\ &= O\left(\frac{k}{m} \log k + k \log \log k\right) \end{aligned}$$

4.1.3 Fixed $n = 1$

The case where $n = 1$ corresponds to a degenerate extreme of the coupon collector's problem. In this case, we simply want to collect k coupons of a single class. This requires that we receive exactly k coupons. Thus we have

$$\begin{aligned} \mathbf{E}[X(1,k)] &= k \\ &= \Theta(k) \end{aligned}$$

4.1.4 $m = \log k$

We now consider a novel tradeoff between m and n, in which $m = \log k$. We show that this tradeoff allows us to maintain $\Theta(k)$ expected coupons, while providing much better reconstruction time in our fault-detection setting.

Let X_i be the number of coupons received for class i after a total of a coupons have been received. Suppose we have that $\Pr(X_i \leq m-1) \leq p(a)$. This is an upper bound on the probability that after receiving a coupons, we do not have enough for class i. Let X be the minimum of the X_i. Note that if $X \geq m$, then our coupon collection is complete. By the union bound, we have $\Pr(X \leq m-1) \leq n \cdot p(a)$.

To bound our expectation, consider the following process. Suppose that we collect a coupons. If we have collected at least m for each of the n classes, we terminate. Otherwise, we discard all coupons and try again. Note that in this setting, we terminate no earlier than the original coupon collection algorithm would. Thus our expectation is at least that of the true process. This corresponds to a geometric distribution with success probability $1 - np(a)$. We expect $\frac{1}{1-np(a)}$ trials to be required, each consuming a coupons. We thus have $\mathbf{E}[X(n,m)] \leq \frac{a}{1-np(a)}$.

We now derive such a bound by Chernoff. Suppose that a coupons have been received. The number of coupons received for class i is distributed as $X_i \sim \text{Binom}(a, 1/n)$. We construct a Chernoff bound to bound the probability that we have not collected at least m coupons of this class after a have been received. Applying the Chernoff bound derived in Mitzenmacher and Upfal [15]:

$$\begin{aligned} \Pr(X_i \leq b) &\leq \Pr(|X_i - \mu| \geq \mu - b) \\ \Pr(|X_i - \mu| \geq \delta\mu) &\leq 2e^{-\mu\delta^2/3} \end{aligned}$$

We apply this bound for $\mu = \frac{a}{n}$ and $\delta = \frac{\mu - b}{\mu}$. This bound holds for all $b < \mu$. Substituting with $b = m - 1$, $a = ck$ (for an arbitrary constant c), and our values of n and m in terms of k, we obtain the bound

$$\Pr(X \leq m-1) \leq \frac{2k^{1-\frac{(-1+c)^2}{3c}}}{\log k}$$

Substituting into our geometric formulation, we have

$$\mathbf{E}[Y] \leq \frac{ck}{1 - \frac{2k^{1-\frac{(-1+c)^2}{3c}}}{\log k}}.$$

Let $c = 5$. We therefore have that

$$\begin{aligned} \mathbf{E}[Y] &\leq 5k\left(1 + \frac{2}{-2 + k^{1/15}\log k}\right) \\ &= 5k + o(1). \end{aligned}$$

Thus we have that

$$\mathbf{E}\left[X\left(\frac{k}{\log k}, \log k\right)\right] = \Theta(k).$$

4.1.5 $m = \log^z k$

Next, consider $m = \log^z k$ for any $z > 1$. We again apply the Chernoff bound from above, yielding the bound

$$\mathbf{E}[Y] \leq \frac{ck}{1 - 2\exp\left\{-\frac{(-1+c)^2 \log^z k}{3c}\right\} k \log^{-z} k}.$$

Note that for any $z > 1$, the ratio of the above quantity to k goes to 1 as $k \to \infty$. We therefore have that

$$\mathbf{E}\left[X\left(\frac{k}{\log^z k}, \log^z k\right)\right] = \Theta(k).$$

We note that this bound fails for $z < 1$. We conjecture that $z = 1$ is the transition point for asymptotically linear behavior.

m	$\mathbf{E}[X(n,m)]$	Asymptotic constant	Notes
1	$\Theta(k \log k)$	1	
c	$\Theta(\frac{k}{c} \log k + (c-1) \log \log k)$	1	Constant c
$\log k$	$\Theta(k)$	$\leq -W_{-1}(e^{-2}) \approx 3.146$	
$\log^z k$	$\Theta(k)$	1	$z > 1$
k	$\Theta(k)$	1	

Figure 1: Summary of asymptotic results for coupon collector with fixed $nm = k$

4.1.6 Improving constant factors

A tighter bound for the $m = \log k$ case can be achieved by using the optimal Chernoff bound.

$$\begin{aligned}\Pr(X_i \leq b) &\leq \frac{\mathbf{E}[e^{-tX_i}]}{e^{-tb}} \\ &= e^{bt}\left(\frac{e^{-t} + n - 1}{n}\right)^a\end{aligned}$$

This quantity is minimized at $t = \log(a-b) - \log b - \log(n-1)$. Simplifying, we obtain the bound

$$\Pr(X_i \leq b) \leq \left(\frac{a-b}{n-1}\right)^{b-a}\left(\frac{a}{n}\right)^a b^{-b}.$$

Let X be the minimum of the X_i. Per the union bound, we have that

$$\Pr(X \leq b) \leq n \left(\frac{a-b}{n-1}\right)^{b-a} \left(\frac{a}{n}\right)^{a} b^{-b}.$$

Substituting into the geometric bound for our expectation and optimizing asymptotically, we obtain the bound $\mathbf{E}[Y] \leq ck + o(k)$ for $c = -W_{-1}(e^{-2}) \approx 3.146$ where W is the Lambert-W function.

4.1.7 Numerical estimation of expectation

For our newly derived results with $m = \log^z k$ and $z \geq 1$, we perform numerical integration on the distribution of coupons required to validate our asymptotic bounds. It can be seen per Newman and Shepp's argument [17] that the expected number of coupons required is $\mathbf{E}[\min(Z_1, \ldots, Z_n)]$ where the $Z_i \sim \text{Gamma}(m)$ are independent.

Our results are summarized in Figure 2. These results are consistent with linear aysmp-

	$z = 0.5$	$z = 1$	$z = 1.5$	$z = 2$
$k = 10^1$	2.94	2.60	1.71	1.47
$k = 10^2$	3.83	2.24	1.71	1.38
$k = 10^3$	4.09	2.32	1.70	1.30
$k = 10^4$	5.03	2.57	1.65	1.30
$k = 10^5$	5.27	2.60	1.69	1.30
$k = 10^6$	5.51	2.63	1.67	1.29
$k = 10^7$	6.27	2.75	1.66	1.28
$k = 10^8$	6.48	2.76	1.67	1.27
$k = 10^9$	6.68	2.77	1.66	1.26
$k = 10^{10}$	6.87	2.84	1.65	1.26

Figure 2: Ratio of $\mathbf{E}\left[X\left(\frac{k}{\log^z k}, \log^z k\right)\right]$ to k for various k and z

totic behavior for all $z \geq 1$. We note that the constant factor for $z = 1$ appears a bit smaller than our upper bound. For $z > 1$, constant factors continue to decrease as $k \to \infty$. Results for $z < 1$ support our conjecture of $z = 1$ being the transition point for linear behavior.

4.2 Application to POP^4

Disregarding for a moment any overhead to achieve linear independence in the collected rows, we expect to require $\mathbf{E}[X(n, m)]$ packets to complete our linear system. Once the linear systems are complete, we must solve n linear systems of size m. Using naive matrix inversion, this takes time $O(nm^3)$.

Now, suppose we use $m = \log k$. Per above, this requires $\Theta(k)$ packets in expectation. This matches the asymptotic behavior of our original algorithm. Furthermore, the reconstruction time is improved to $O(k \log^2 k)$, a significant improvement over the original $O(k^3)$.

Therefore, by partitioning our coupon collector's problem into small independent groups, we maintain our asymptotically expected number of packets while significantly improving reconstruction time.

In general, using $m = \log^z k$ for $z \geq 1$ yields $\Theta(k)$ expected packets with a reconstruction time of $O(k \log^{2z} k)$. Under the bounds we have derived, this is optimized at $z = 1$.

4.2.1 Linear independence

Note, however, that we may need to receive more than m packets per cluster of nodes. This occurs when one of the rows received is linearly dependent on other rows. In this case, the packet is discarded and we must wait for another. MacKay [14] showed that the probability that we fail to obtain m linearly independent rows after receiving $m + c$ is at most 2^{-c}.

In order to show that our expected runtime remains linear, we collect dm coupons for some constant d for each class. We then bound the probability that less than dm coupons are received using our earlier Chernoff bound, and bound the probability that any cluster does not have sufficient linearly independent rows using a union bound. The probability that any cluster does not have sufficient linearly independent rows is therefore at most $n2^{-(d-1)m}$. Let F be the event in which we do not have sufficient linearly independent rows.

Our Chernoff bound now uses $\mu = \frac{a}{n}$ and $\delta = \frac{\mu - dm}{\mu}$. We again substitute $a = ck$ (for an arbitrary constant c), and our values of n and m in terms of k, we obtain the bound

$$\begin{aligned}\Pr(X \leq dm \cup F) &\leq \frac{2k^{1-\frac{(-1+c)^2}{3c}}}{\log k} \\ &\leq \frac{k\left(2^{-(d-1)\log k} + 2k^{-\frac{(c-d)^2}{3c}}\right)}{\log k}\end{aligned}$$

Let Y be the number of packets needed to collect a linearly independent set of m for each cluster. Letting $c = 8$ and $d = 3$ and substituting this expression into our geometric bound for the expectation, we obtain the bound

$$\begin{aligned}\mathbf{E}[Y] &\leq \frac{8k}{1 - \frac{2k^{-1/24} + 4^{-\log k} k}{\log k}} \\ &= 8k + o(k)\end{aligned}$$

Thus we retain linear asymptotic behavior, even when ensuring linear independence.

Finally, note that because of the $\log \log k$ bits used to estimate the path length, m may differ from the intended value of $\log k$ by a factor of 2. Substituting $m = 2 \log k$ into our Chernoff bound yields an upper bound of $\mathbf{E}[Y] \leq 12k + o(k)$.

4.2.2 Probability bounds

Finally, note that we can extend these expectations to make correctness guarantees with high probability if desired. Per Markov's inequality, $\Pr(Y \geq 2\mathbf{E}[Y]) \leq \frac{1}{2}$. Thus, we send $2\mathbf{E}[Y]$

packets. If we have obtained our n linearly independent systems, we terminate. Otherwise, we attempt the process again. Thus to achieve success probability $1 - \delta$, we simply send $2 \log_2 (1/\delta) \mathbf{E}[Y]$ packets.

m	Expected Packets	Reconstruction Cost	Notes
1	k	$O(k^3)$	
$\log k$	$ck + o(k)$	$O(k \log^2 k)$	$c \leq 12$
$\log^z k$	$k + o(k)$	$O(k \log^{2z} k)$	$z > 1$
k	$k \log k + O(k)$	$\Theta(k \log k)$	

Figure 3: Upper bounds on expected packets required and reconstruction consts for various values of m. For $m = \log k$ or $m = \log^z k$, each packet includes $1 + \log \log k$ bits. For others, each packet includes 1 bit.

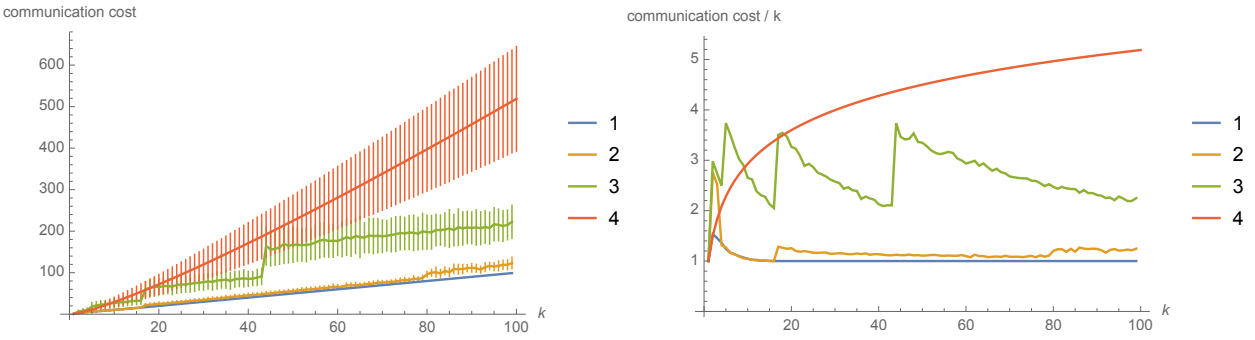
5 Evaluation

We now simulate the communication cost of our end-to-end algorithm in order to verify the correctness of our theoretical results. Because the reconstruction cost is a deterministic function of n and m, we compute it for concrete instance sizes without simulation. Furthermore, because an evaluation of POP^4's real-world performance would require a separate low-level implementation inside the network stack and a high-bandwidth network testbed, we leave this aspect of evaluation to future work. Our evaluation code, data, and visualizations are publicly available at pop4.

Our simulation exploits a key optimization simplifying the problem environment: it lacks any notion of network topology or individual switches, but rather acts on random rows in $\{0, 1\}^m$ as a primitive. This is sound because successful reconstruction depends only on the system of equations for each cluster having full rank.

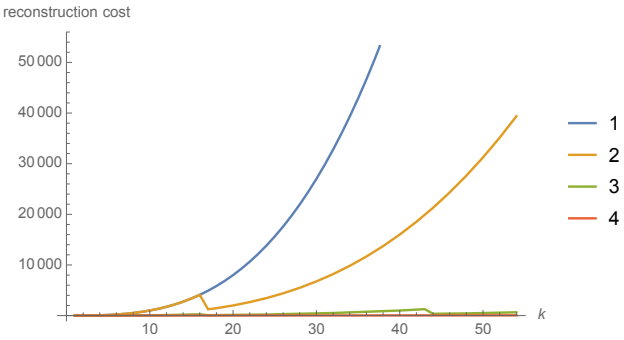
The main bottleneck when scaling our simulation to larger instances is that we need to perform Gaussian elimination multiple times in order to detect the minimum number of messages that would enable reconstruction. Note that this is not necessary when using the algorithm in a standard setting, as one can choose to wait for a number of packets that will succeed with high probability. We only attempt reconstruction once we reach the minimum number of rows that might be invertible, and we remove the linearly dependent rows at each step to reduce the instance size for the next inversion, but this process still dominates the runtime of the simulation. Luckily, the simulation is performant enough for the instance sizes relevant to this domain.

We performed $N = 250$ trials for k in $[2, 100)$ across 4 instantiations of POP^4: first, $n = 1$ (the optimum for communication cost); second, $n = \frac{k}{\log^2 k}$; third, $n = \frac{k}{\log k}$; and fourth, $n = k$ (the optimum for reconstruction cost). These configurations are labelled $1 - 4$ respectively in figures with all four points of comparison.

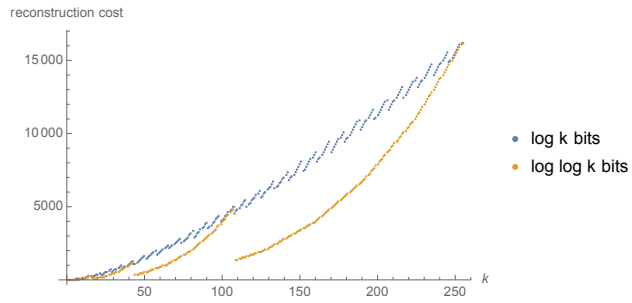


(a) Mean of communication cost (number of sum bits that must be received) over k, across the four chosen configurations of POP^4. Error bars represent one std. deviation.

(b) Mean of the ratio between communication cost (number of sum bits that must be received) and k over k, across the four chosen configurations of POP^4.



(c) Reconstruction cost (complexity of matrix inversion) over k, across the four chosen configurations of POP^4.



(d) Reconstruction cost (complexity of matrix inversion) over k for configuration 2, when using $\log k$ vs. $\log \log k$ bits to transmit the path length.

Figure 4: In the legends, the label 1 corresponds to $n = 1$, 2 to $n = \frac{k}{\log^2 k}$, 3 to $n = \frac{k}{\log k}$, 4 to $n = k$.

Our experiments increased our confidence that our asymptotic characterizations of the tradeoff space are correct; as seen in Figures 4a and 4d, configurations $1 - 3$ have linear communication cost, with 1 and 2 tending to a constant factor of 1 as k increases, and 3 to a constant factor > 1 that is dominated by our upper bound. It does appear that there is still opportunity to improve the constant factor in our upper bound for the third configuration, from 12 to approximately 3. Meanwhile, Figure 4c demonstrates the disparity in reconstruction cost across the algorithms, even for small instance sizes $k < 50$. Figures 4a, 4b, and 4d show that rounding artifacts from using $\log \log k$ bits to represent the path length do not substantially harm POP^4's performance.

6 Conclusion

In this work, we proposed a family of algorithms for fault detection that offer a variety of compromises between communication cost and reconstruction time, building off recent work in probabilistic in-network telemetry [3] and taking inspiration from fountain codes [14]. A particularly interesting compromise is $m = \log k$, which delivers both linear expected runtime and $O(k \log^2 k)$ reconstruction time. Simulations under a wide variety of parameter settings

support these theoretical results. Although we focus on the setting of fault detection, our algorithms may be used to efficiently collect any Boolean value from switches in a flow, assuming the path of the flow is known. Future work in this direction includes implementing our algorithm on commodity programmable switches, verifying its performance with real-world network topologies, and extending the ideas presented here to related problems in networking.

References

[1] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 503–514. ACM, 2014.

[2] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 419–435, Renton, WA, April 2018. USENIX Association. ISBN 978-1-939133-01-4. URL `https://www.usenix.org/conference/nsdi18/presentation/arzani`.

[3] Ran Ben-Basat, Gianni Antichi, Jan Kucera, Michael Mitzenmacher, and Minlan Yu. Pop3: Probabilistic on-packet path profiling. 2019.

[4] Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82 – 87, 1976. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(76)90071-5. URL `http://www.sciencedirect.com/science/article/pii/0020019076900715`.

[5] N. G. Duffield and Matthias Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, 9(3):280–292, June 2001. ISSN 1063-6692. doi: 10.1109/90.929851. URL `http://dx.doi.org/10.1109/90.929851`.

[6] Michael T. Goodrich. Probabilistic packet marking for large-scale ip traceback. *IEEE/ACM Trans. Netw.*, 16(1):15–24, February 2008. ISSN 1063-6692. doi: 10.1109/TNET.2007.910594. URL `http://dx.doi.org/10.1109/TNET.2007.910594`.

[7] T.P.A. W. Group. In-band network telemetry (int) dataplane specification. January 2018.

[8] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 139–152. ACM, 2015.

[9] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 71–85, 2014.

[10] Doug Hegge, Charles Lindsay, Theodore Ross, Krishna Narayanaswamy, and Barry Spinney. System and method for flow mirroring in a network switch, December 27 2001. US Patent App. 09/791,517.

[11] Kihong Park and Heejo Lee. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 1, pages 338–347 vol.1, April 2001. doi: 10.1109/INFCOM.2001.916716.

[12] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*, 2015.

[13] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 311–324, 2016.

[14] David JC MacKay. Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068, 2005.

[15] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis. Cambridge University Press, 2017. ISBN 9781107154889. URL `https://books.google.com/books?id=E9UlDwAAQBAJ`.

[16] Michael Mitzenmacher. Digital fountains: A survey and look forward. In *Information Theory Workshop*, pages 271–276. IEEE, 2004.

[17] Donald J Newman. The double dixie cup problem. *The American Mathematical Monthly*, 67(1):58–61, 1960.

[18] Christian Esteve Rothenberg, Carlos Alberto Braz Macapuna, Maurício Ferreira Magalhães, Fábio Luciano Verdi, and Alexander Wiesmaier. In-packet bloom filters: Design and networking applications. *Computer Networks*, 55(6):1364–1378, 2011.

[19] Pegah Sattari. *Revisiting IP Traceback as a Coupon Collector's Problem Thesis*. PhD thesis, University of California, Irvine, 2007.

[20] Pegah Sattari, Minas Gjoka, and Athina Markopoulou. A network coding approach to ip traceback. In *2010 IEEE International Symposium on Network Coding (NetCod)*, pages 1–6. IEEE, 2010.

[21] Alex C Snoeren, Craig Partridge, Luis A Sanchez, Christine E Jones, Fabrice Tchakountio, Stephen T Kent, and W Timothy Strayer. Hash-based ip traceback. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 3–14. ACM, 2001.

[22] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Simplifying datacenter network debugging with pathdump. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 233–248, 2016.

[23] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. Stroboscope: Declarative network monitoring on a budget. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 467–482, Renton, WA, April 2018. USENIX Association. ISBN 978-1-939133-01-4. URL `https://www.usenix.org/conference/nsdi18/presentation/tilmans`.

[24] Eric W. Weisstein. Erdős-borwein constant. URL `http://mathworld.wolfram.com/Erdos-BorweinConstant.html`.

[25] B. Wu, K. Xu, Q. Li, Z. Liu, Y. Hu, M. J. Reed, M. Shen, and F. Yang. Enabling efficient source and path verification via probabilistic packet marking. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–10, June 2018. doi: 10.1109/IWQoS.2018.8624169.

[26] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 479–491. ACM, 2015.