# Tweet id reconstruction challenge

In the following short report, I will show my analysis of the random tweet ids.

A tweet id consists of 44-bit binary number that is composed of 4 parts: Time component (22 bits),

Data Center Id (5 bit), Server Id (5 bit), Sequence Id (12 bit).

I will use a combination of the Data Center Id and Server Id as the machine id.

**Initial pre-scan results**

After examining the random ids file, I found that there are total of 81 sequence ids, and 38 total machines ids used in the random set. And the top four machines used are responsible to process 20% of the tweets.

furthermore, after examining the sequence ids I found that 7 ids were enough to get over 99% of the tweets and that using sequence id 0 will yield 61% of them.

| Machine id | % | Total % |
|---|---|---|
| 363 | 0.05437 | 0.05437 |
| 375 | 0.05402 | 0.10839 |
| 382 | 0.05374 | 0.16213 |
| 365 | 0.0507 | 0.21283 |
| 366 | 0.04918 | 0.26201 |
| 381 | 0.04839 | 0.3104 |
| 336 | 0.04496 | 0.35536 |
| 361 | 0.04488 | 0.40024 |
| 350 | 0.04486 | 0.4451 |
| 335 | 0.04485 | 0.48995 |
| 333 | 0.04468 | 0.53463 |
| 373 | 0.03506 | 0.5697 |
| 377 | 0.03347 | 0.60317 |
| 332 | 0.03304 | 0.6362 |
| 352 | 0.03097 | 0.66718 |
| 340 | 0.02545 | 0.69263 |
| 327 | 0.02524 | 0.71787 |
| 339 | 0.02519 | 0.74306 |
| 376 | 0.02417 | 0.76723 |
| 364 | 0.02262 | 0.78985 |
| 347 | 0.02003 | 0.80988 |
| 326 | 0.0196 | 0.82948 |
| 324 | 0.01884 | 0.84832 |
| 348 | 0.01878 | 0.8671 |
| 325 | 0.01843 | 0.88553 |
| 334 | 0.0171 | 0.90263 |
| 378 | 0.01566 | 0.91829 |
| 323 | 0.01353 | 0.93182 |
| 379 | 0.01223 | 0.94405 |
| 346 | 0.01172 | 0.95578 |
| 362 | 0.01092 | 0.96669 |
| 372 | 0.01079 | 0.97749 |
| 342 | 0.01003 | 0.98752 |
| 321 | 0.00689 | 0.99441 |
| 349 | 0.00515 | 0.99956 |
| 322 | 0.00021 | 0.99977 |
| 328 | 0.00013 | 0.9999 |
| 320 | 0.0001 | 1.0 |
| total machines: 38 | | |

*Figure 1: Machine ids*

| Sequence id | % | Total % |
|---|---|---|
| 0 | 0.61148 | 0.61148 |
| 1 | 0.2363 | 0.84778 |
| 2 | 0.07281 | 0.9206 |
| 5 | 0.02683 | 0.94743 |
| 3 | 0.01914 | 0.96657 |
| 6 | 0.0165 | 0.98307 |
| 7 | 0.0066 | 0.98967 |
| 4 | 0.00453 | 0.9942 |
| 8 | 0.00209 | 0.9963 |
| 10 | 0.00116 | 0.99746 |
| 11 | 0.00086 | 0.99832 |
| 9 | 0.0006 | 0.99893 |
| 12 | 0.00043 | 0.99936 |
| 13 | 0.00017 | 0.99953 |
| 15 | 8e-05 | 0.99961 |
| 14 | 8e-05 | 0.99969 |
| 16 | 7e-05 | 0.99975 |
| 17 | 4e-05 | 0.99979 |
| 18 | 2e-05 | 0.99982 |
| 19 | 2e-05 | 0.99984 |
| 20 | 2e-05 | 0.99986 |
| 32 | 1e-05 | 0.99987 |
| 21 | 1e-05 | 0.99988 |
| 22 | 1e-05 | 0.9999 |
| 33 | 1e-05 | 0.99991 |
| 24 | 1e-05 | 0.99991 |
| 23 | 1e-05 | 0.99992 |
| 25 | 1e-05 | 0.99993 |
| 27 | 1e-05 | 0.99994 |
| 26 | 1e-05 | 0.99994 |
| 28 | 1e-05 | 0.99995 |

*Figure 2: Sequence ids*

**Naive method analysis**

Combining these two facts I can deduce that while there are 4096 different sequence ids per machine, we can only check 7 of the values to get more than 99% of all the messages, and since the machine ids are 10 bit long we get from 1024 different machine ids to 34 to get over 99% of the tweets.

This is an improvement by a factor of $585 * 30 \; = \; 17{,}550$.

Now if we want only 10% of the full FireHose, we will perform the following calculation:

We have 3600 API calls per hour → 360,000 tweets per hour → 90,000 tweets per 15 minutes

Given that we have $38 \; machines * 7 \; sequence \; numbers = 266 \; scans \; needed \; per \; millisecond$

We have access to 90,000 tweets every 15 minutes → $\frac{90{,}000}{266} * 10 =$
$3383.4 \; ms(3.38 \; seconds) \; of \; data \; every \; 150 \; minutes$

So we can get $\frac{24 \; hour*60 \; minutes*3.38s \; data}{150} = 32.48 \; seconds \; of \; data \; per \; day \; of \; constant \; API \; calls$

Since we want 10% of the data we will need:

$$\frac{24 * 60 * 6}{32.48} = 266 \; keys \rightarrow 0.266 \; API \; calls \; per \; millisecond \rightarrow$$

$90{,}000 * 266 = 23{,}940{,}000 \; API \; calls$ for a single account just to get that 10%.

**Improved method**

If we will only scan the top 4 machines to get 21% of the tweets, and sequence id of 0 we will get ~12% of the tweets. By following the same calculation, we will get that every single account will yield 225 seconds of data every 150 minutes. Following this we can get for every day of constant calls 2160 seconds of the tweets. This gives us the following:

$$\frac{86400 \; seconds \; per \; day}{2160 \; seconds \; of \; data \; per \; day} = 40 \; API \; keys \; \rightarrow 0.04 \; API \; calls \; per \; millisecond$$

This is an improvement by a factor of 60 from the naïve approach of scanning 7 sequence ids per machine, and will require 6 time less keys.

After examining the 1k, 10k, 100k files with sequence 0 we get the following:

| | 1k | 10k | 100k | average |
|---|---|---|---|---|
| 363 | 29.925 | 40.149 | 45.971 | 38.68167 |
| 375 | 31.164 | 39.538 | 45.696 | 38.79933 |
| 382 | 31.150 | 40.058 | 45.202 | 38.80333 |
| 365 | 29.728 | 41.110 | 45.951 | 38.92967 |

So, while using the same API calls, we will get on average 38.8035% of the busiest tweets, from the FireHose on top of the 12% of the entire FireHose.

**Future Work**

This short analysis could be expended further in a few different ways.

First, it would be possible to find an optimal value of the number of ids that needs to be scanned as a function of % of total tweets received, as a cross product of the random sample along with the top 1k,10k,100k busiest minutes.

Secondly, given a longer period and access to several developer accounts it would be possible to check this analysis and verify that the numbers correspond and yield good results while matched with the suggested method.

Lastly, train a model using the top milliseconds data and use this model to predict when peaks will occur and capture them without the need to constantly perform API calls to twitter.