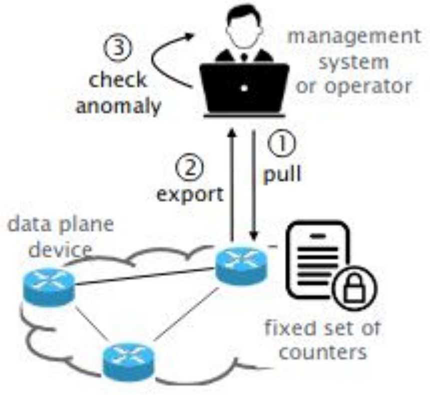
Paper Assigned: Stats 101 in P4(Towards In-Switch Anomaly Detection)

Paper Presentation by Shivam Patel

Basic Outline

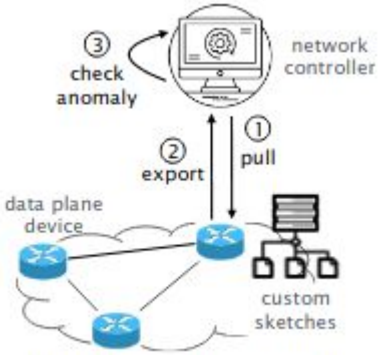
- One area that P4(Data plane Programmability) has helped is network monitoring.
- A large component of network monitoring is, to in some form or fashion compute statistics over the data/traffic occurring inside the network.
- The authors would like to construct an architecture for in switch anomaly detection, i.e. anomaly detection occuring autonomously in the network/devices without or with minimal human involvement.
- Given the added complexity of such a scheme, What do you all think the intent is? For what cases would having anomaly detection being done autonously be better than the approach on the right?



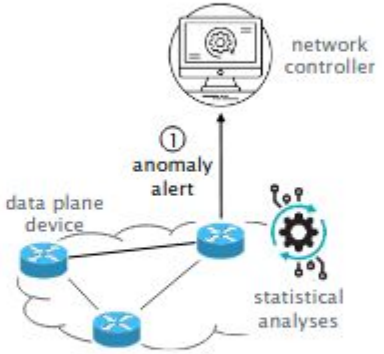
(a) Traditional approach.

A slightly better approach

- Approach B on the right is referred to as sketch(packet based) only, and is the simplest improvement on approach A, but is limited for situations needing the highest possible levels of reactivity(how?) since there exists a feature of periodically pushing and pulling data to the controller for analytics.
- Some examples the author provides to where Approach B doesnt work and Approach C is necessary are failure detection, DDOS protection, and Load Balancing. Speed more than that of a human, or periodically pushing and pulling



(b) Sketch-only approach.



(c) Envisioned approach.

Actual In-Switch Anomaly detection

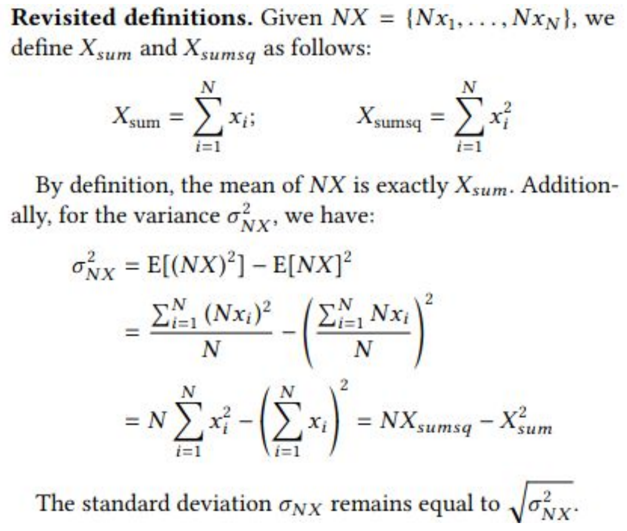
- Authors mention that there are numerous issues with this, inspite of it being in theory the fastest. Due to the nature of programmable switches(and maybe P4) themselves.
  - low memory availability
  - lack of native floating point numbers
  - lack of division
  - inability to "loop" without doing recirculation(reduces packet throughput when considereing the system as a whole)
- In particular this limits in numerous ways, the means by which we can compute the various basic statistics such as mean, variance, standard deviation, and the percentiles (example: median is 50% percentile)
- To Address this, the authors emphasize the following 2 ideas.
  - "tweak" the definitions of the statistics in question, as well as the online(sequential) algorithms that compute them.
  - "careful" engineering of match action tables and registers to avoid recirculation.

Statistics in P4

- Model: Suppose we want to track N values of interest X = [X1, ... Xn]
- The values are stored in a counter and updates per time t
- Given that the typical formulas for mean. variance, standard deviation depend on division and (for the last) square rooting, the new ones are done using a clever multiplication trick, along with an algorithm to estimate square roots to a nearby integer
- Instead of Computing a moving average on X, the authors suggest using NX = [N*X_1, ..., N*X_n]
- Then, the mean of NX is the sum of X, and in theory, can then be used to compare against new incoming values through developing this scheme for the mean further into variance and standard deviation using variants on the standard formulas for those statistics.

Variance and Standard Deviation

- There exist 3(4) problems with what's been discussed so far
- A separate counter in the form of a register will be used to keep track of the elements and counts of a distribution formed from the N values of interest, as well a register to store the statistics, and the values are updated per time interval t(100 ms)
    - what if we don't have enough memory due to large N?
    - How do we sequentially update the statistics that have non linearity?
    - How do we deal with square roots?
    - (for fun question, not from the paper) Lets see how much you all remember from the real stats 101 you had years ago, what are your comments on the last sentence on the right?



Revisited definitions. Given $NX = \{Nx_1, \ldots, Nx_N\}$, we define X_{sum} and X_{sumsq} as follows:

$$X_{\text{sum}} = \sum_{i=1}^{N} x_i; \qquad X_{\text{sumsq}} = \sum_{i=1}^{N} x_i^2$$

By definition, the mean of NX is exactly X_{sum}. Additionally, for the variance σ^2_{NX}, we have:

$$\begin{aligned}\sigma^2_{NX} &= \mathrm{E}[(NX)^2] - \mathrm{E}[NX]^2\\ &= \frac{\sum_{i=1}^{N}(Nx_i)^2}{N} - \left(\frac{\sum_{i=1}^{N} Nx_i}{N}\right)^2\\ &= N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2 = NX_{sumsq} - X^2_{sum}\end{aligned}$$

The standard deviation σ_{NX} remains equal to $\sqrt{\sigma^2_{NX}}$.

How to estimate Square root for use in Sample Standard Deviation

- Complicated at first glance, essentially packing a float based on how big the int is, doing shifts, and reassembling.
- Main ideas are essentially as follows
  - convert the integer to floating point representation,say f.
  - choose f's exponent(remember from your early systems classes?) as the index of the integer's most significant bit.
  - copy the rest of the integer aside from the MSB as the mantissa(kind of like the part after the decimal).
  - then do right bit shift by 1.
  - take the result and convert back into a new integer(result of the square root estimate) by putting the exponent part's value(add it up) as the index of the integer's most significant bit, and take the mantissa as the integer's least significant bits.

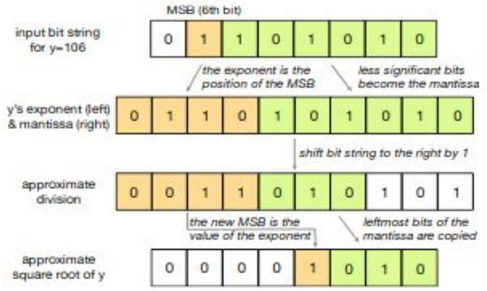
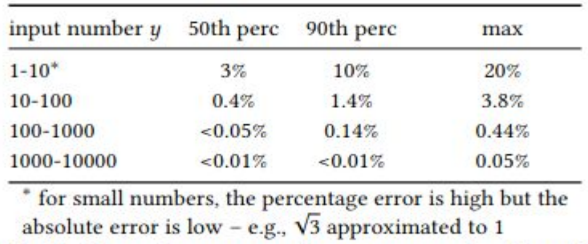

MSB (6th bit)
input bit string for y=106
0 1 1 0 1 0 1 0
the exponent is the position of the MSB
less significant bits become the mantissa
y's exponent (left) & mantissa (right)
0 1 1 0 1 0 1 0 1 0
shift bit string to the right by 1
approximate division
0 0 1 1 0 1 0 1 0 1
the new MSB is the value of the exponent
leftmost bits of the mantissa are copied
approximate square root of y
0 0 0 0 1 0 1 0

Figure 2: Example of our approximated square root algorithm: it approximates $\sqrt{106}$ to 10.



input number y	50th perc	90th perc	max
1-10[*]	3%	10%	20%
10-100	0.4%	1.4%	3.8%
100-1000	<0.05%	0.14%	0.44%
1000-10000	<0.01%	<0.01%	0.05%

[*] for small numbers, the percentage error is high but the absolute error is low – e.g., $\sqrt{3}$ approximated to 1

Table 2: Percentage error in square root estimation with respect to the fractional square root value.

Online computation of percentiles.

- Example given in the paper is for 50th percentile(median) but can done for any (1-a)th percentile for 0 < a < 1
- F = [f1, . . . , fn] is the frequency distribution
  - fi := count of Xi in registers / total # packets encountered(or time)
  - Again, since we can't divide, track NF = [Nf1, ..., Nfn] in registers
  - As well, keep track of the cumulative frequency of all the values less than the current median, and the cumulative frequency of all values higher than it.
  - Then for new data items, update the upper or lower cumulative frequencies and if the upper cumulative frequency becomes larger than the lower(or lower becomes smaller than upper) then increment/decrement the index in X that corresponds to the median.
- Performs poorly for sparse distributions, since the authors mentioned their procedure's implementation requires skipping indices that have frequency 0. So essentially to avoid "looping", only 1 increment can be done on the index that corresponds to the median per new packets/t

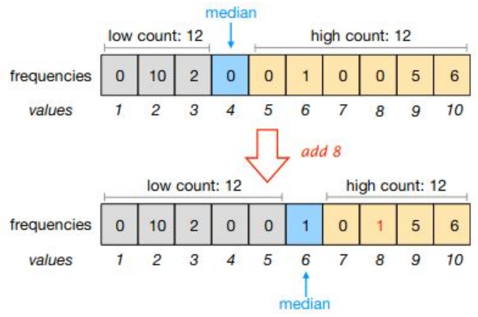
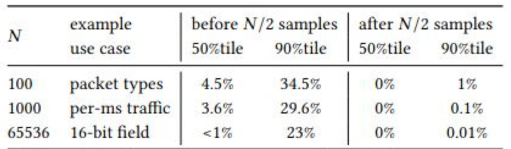


Figure 3: Example of our approximated algorithm to compute the median on a frequency distribution.



N	example use case	before $N/2$ samples 50%tile	before $N/2$ samples 90%tile	after $N/2$ samples 50%tile	after $N/2$ samples 90%tile
100	packet types	4.5%	34.5%	0%	1%
1000	per-ms traffic	3.6%	29.6%	0%	0.1%
65536	16-bit field	<1%	23%	0%	0.01%

Table 3: Median estimation error for distributions of N elements, over 20 repetitions per value of N.

Case Study/Experiment

- A load balanced network has of 36 destination hosts in 6 /24 subnets with /8 prefixes.
- Emulates monitoring system to detect traffic spikes at the destination hosts
- Monitoring system is implemented via a network switch running p4 code that does the statistical checks, and the network controller makes updates to a binding table(match action table that governs and fine tunes what operations to do to compute the statistics for the case at hand)
- Starts off monitoring only the /8 prefixes. then as the switch relays traffic spike alerts to the controller, the controller then updates the binding tables to include the whole /24 subnet in the tracking/analysis.
- Results are pretty good, time intervals in which to take samples range from 8ms to 2 seconds, and in all cases, the spike is detected in the first time interval after the spiker occurs, as well determining the location at which the spike is occuring at takes an additional 2 seconds due to data plane/control plane interactions.

$$E[f(X)] = \frac{\sum_{i=1}^{N} f(x_i)}{N}$$

Future Directions

- The authors of the paper have published the code in a library called Stat4.
- They acknowledge that the methods are pretty crude but simply wanted to show that schemes of this sort were possible. The following are some avenues of future research they think would be interesting.
- Suggest that methods can be extended to proactive actions, like when a surge begins reroute packets before congestion.
- Mention that a big concern is scalability, in the lack of memory(how can we get more?) and the standard problems that arise in distributed computing like synchronization and existence of (pseudo) parallel algorithms, whereas in centralized architectures that can be scaled easily for many cases.
- As well they mentioned that it would be interesting to see which other probability distributions can be included in the future, aside from the current uniform distribution being used(or normal distributions they get for free due to sum of uniforms converging to normals.)
- In my opinion it would be interesting to experiment with poisson and gamma distributions since they seem to model relevant phenomena, a possible implementation path would be to encode their cumulative distributions in match action tables that take as key a pair (a,b) and return a value that can then have a few arithmetic operations that result in the difference of the cumulative distribution at a and b.

$$0 < \mathrm{Var}[S_n] = \mathrm{E}[S_n^2] - \mathrm{E}^2[S_n] \quad \Leftrightarrow \quad \mathrm{E}^2[S_n] < \mathrm{E}[S_n^2] \quad \Leftrightarrow \quad \mathrm{E}[S_n] < \sqrt{\mathrm{E}[S_n^2]} = \sigma.$$