Machine Learning Engineer Nanodegree

Capstone Project

Anderas Lewitzki
February 02, 2019

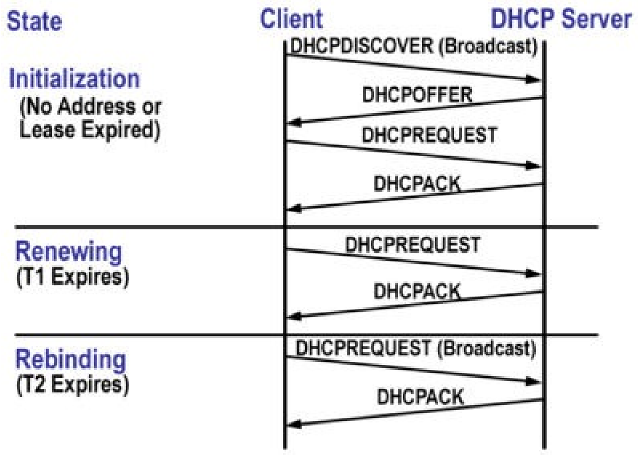
Realtime contextual and passive network outage detection

Code notebook for this project can be found at:
https://github.com/std3rr/realtime_contextual_and_passive_outage_detection/project_notebook.ipynb
(https://github.com/std3rr/realtime_contextual_and_passive_outage_detection/project_notebook.ipynb)

I. Definition

Project Overview

Dynamic Host Configuration Protocol[1] is the standardized method by which a computer network connected client dynamically leases its address. For Internet connected devices this mechanism for controlling the temporary leasing of addresses is normally managed by one or more servers at the Internet Service Provider (ISP).[2] In Sweden today its common to have a transport network, operated by an network operator, separate from the service provider. These service providers rarely has any insight into those networks but often still manage the address leasing through a Relay Agent Information option called option82.[3] "Clients" in this paper refer to the customer premise devices requesting an address. "Address", "IP" and IPaddress might be used interchangeably and refers to the same thing.[4]



We will explore the possibilities of real time network outage detection through anomalies in the sparse address renewal (DHCP) signaling. Real time in our case means scan and trigger a anomaly detection event as close as possible to the occurrence. The sampling rate in this setup is minute by minute counter of clients requesting a address lease. Even though the individual client events accounted for in the signal analyzed is very sparse. Contextual in this case refers both to the model, but also the data stream. In the data, by aggregating geographical zip code and logical network identifier. But also the method and model will be contextual as many decisions are due to the specific use case.

In our use case we want to apply this is on unmanaged open networks where the ISP has control over the DHCP. By applying this we could get a sense on local disturbances in these open unmanaged networks, where we normally don’t have any insight or surveillance regarding aggregation or transport network outages. Outages in these cases might be caused by hardware or software failures, power outages or fiber cuts.

By generating events for supposed network outage the service provider can more proactively and automatically prepare trouble tickets and trigger customer support call voice prompts, linked to the specific autonomous network and geographic area. Having a clear prepared voice prompt regarding the supposed network outage, besides pure goodwill towards customers, will mitigate the flow of customars calling technical support from having to speak to an human agent. As we're exploiting an already existing data flow, this method will be totally passive survailance towards the network and clients.

Normally you measure for example active client count which will start to decline first when the lease time has passed. This method will try to identify the underlying frequency pattern of requests comming in and will be able to identify anomalies up to half the lease time earlier, closer to the outage.

We will also look at how the population size per area and temporal coverage will affect the resolution of the detector.

[1] *Dynamic Host Configuration Protocol* [https://tools.ietf.org/html/rfc2131 (https://tools.ietf.org/html/rfc2131)](https://tools.ietf.org/html/rfc2131)

[2] *Internet Service Provider* [https://en.wikipedia.org/wiki/Internet_service_provider (https://en.wikipedia.org/wiki/Internet_service_provider)](https://en.wikipedia.org/wiki/Internet_service_provider)

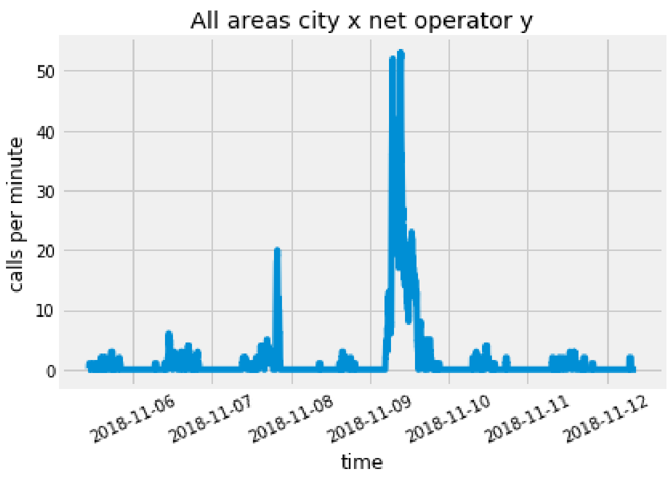
[3] *DHCP Relay Agent Information Option* [https://www.ietf.org/rfc/rfc3046.txt (https://www.ietf.org/rfc/rfc3046.txt)](https://www.ietf.org/rfc/rfc3046.txt)

[4] *IP Address* [https://en.wikipedia.org/wiki/IP_address (https://en.wikipedia.org/wiki/IP_address)](https://en.wikipedia.org/wiki/IP_address)

Problem Statement

Within our subject network operations we have access to management for alarm signaling and report disturbances. But for cases where we act pure service provider, using some other operator networks as transport, we're mostly blind for local occational outages.
One big problem is that when the operator got outages, we will not know about it until numerous customers call in to technical support. Also in these cases the customers won’t get any heads up in the voice prompt, as neither systems nor agents are aware of the outage.

We can see this as an increasing number of customers call from the same areas within a short timeframe. When this happens and we do not have any voice prompt in place customers will get through to human support agents that need to walkthrough customers to identify the problem and in worst cases also send out a technician to the customer site. In the long run that's very costly, both economically and in the form of customer trust.
Bellow is an example of unmanaged and uncaptured outage generating customer support calls. These are the kind of incidents we in the end will try to mitigate.



What we will do within this project:

- set up a system that mapreduces[5] these DHCP adress renewals into counters by minutes, network and geographical area
- export a subset of this data as csv to explore in jupyter notebook
- develop methods for analyzing, simulate the minute by minute process and detecting anomalies

[5] *Map Reduce* [https://en.wikipedia.org/wiki/MapReduce (https://en.wikipedia.org/wiki/MapReduce)](https://en.wikipedia.org/wiki/MapReduce)

Metrics

This is a continuous unsupervised problem and we don't have any stable ground truth to train on. The data that we have is very sparse, fragile and we can't really tell if things are true or false positives.

That being said we can always simulate an outage by degrading or zeroing the counters over a specific timespan. Then measure how the client population size will affect the resolution and delay of the detection.

What we want to measure here is how close to an actual outage we sense a big enough anomaly to trigger an event.

Normally you would probably measure how many active clients you have. In case of DHCP a client is considered active until its lease time expires. If lease time is one hour, it'll take, in best case, half that time until we can se a drop in activity.

The only thing we could say is if we detect a gradient sudden drop in DHCP followed by increasing calls that’s a success.
How early we detect an event is more explained by the amount of clients in the area and the spread in DHCP requests.
But this is on a higher level as we also need to measure how good the actual model is in detecting the decrease. In this case we can evaluate the last value against a moving average for the prediction.
So a huge problem here is that we do not have clear true or false positives / negatives.
Even in the case where we have a drop in DHCP requests followed by increase in calls we can’t assume that the dip explains all the calls.

As the resolution depends alot on client population per region and covered minutes over the interval, we want to measure how many minutes delay we can expect. We'll get back to this in the benchmark section.

II. Analysis

Data Exploration

Let us discuss the data and where it comes from. The base for this is counter metrics sampled every minute. All the clients on internet using dynamic addressing via DHCP sends DHCPREQUEST messages as defined by the RFC referenced in the first section.
When this happens a in-memory database update its client table setting the expire date, in our case as a unsigned integer. This is joined with a service table that holds a CRM[6] reference service id. As this is fixed service we use the CRM system to add features such as zip code and operator network. This is then streamed as JSON objects looking like this :

```
{"ts":1548461820,"netop":"NETWOPERATORX","zipcnt":[[117637,13],[117630,2],[117734,4],[117631
,6],[117638,4]]}
```

Timestamp is in Unix timestamp[7] To be able to expose the data without compromising confidentiality and potential for the sake of this project, we've hashed the operator and shifted zip codes. But we kept zip numbering distances intact, as it might be valuable in some future model.

To make it easy to import a second service repack this as csv, adding calls as feature from a backend service of the voice prompt system.
Then we've selected a subset sample population of a medium sized city with 141 areas for this experiment. Example of the semicolon separated (CSV) export final data:

ts	zip	dhcp	call
1542348720	44944	0	0
1542348780	44944	3	0
1542348840	44944	0	0
1542348900	44944	2	0
			...

The sample population used here generated 53000000 rows and span over one month between October and November of 2018.

Although the DHCP has a specified lease time configuration of 3600 seconds and the RFC specifies half lease, it's possible for the client to request a new ip address after a minimum of 60 seconds. This together with offsett delay and back off algorithms creates a jitter.

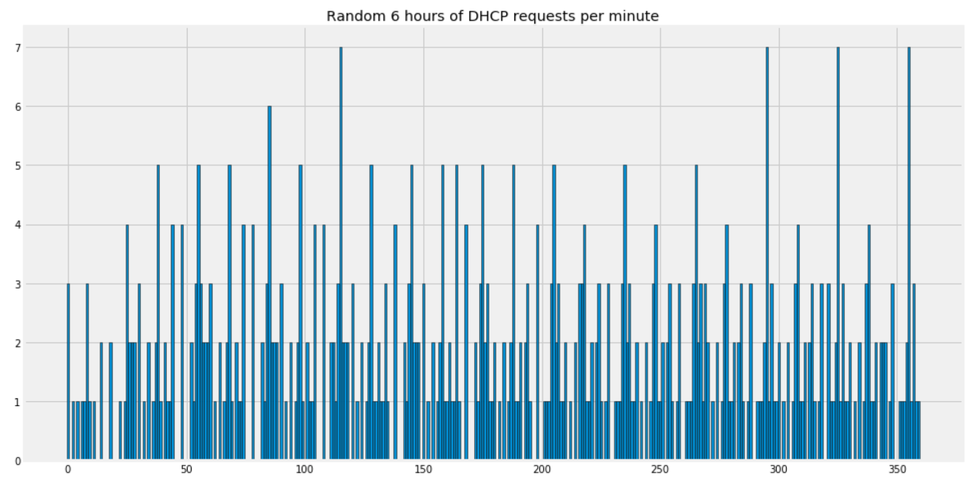
We have implemented a little collector service that uses the DHCP configured lease time to extract clients that renewed their ipaddress within last minute. This service then do a lookup against CRM system to sort the clients categoricaly into their respective registered zip-code area and

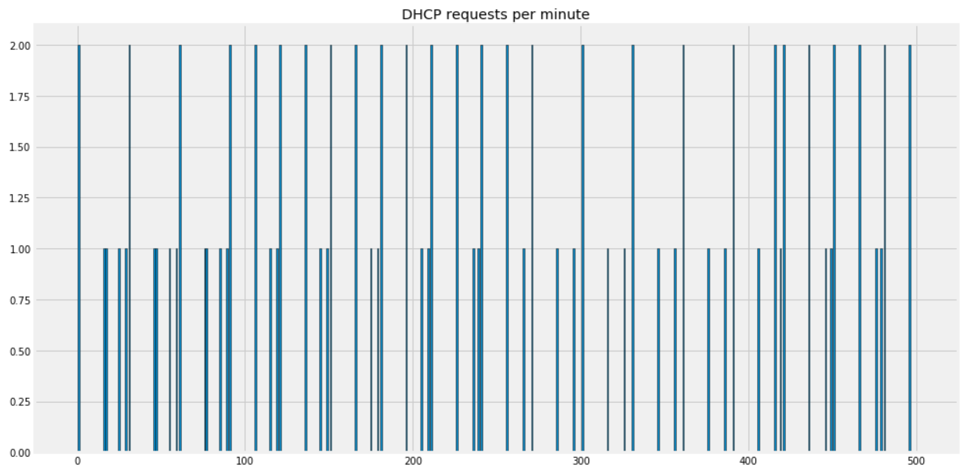
[6] *Customer Relationship Management (CRM)* [https://en.wikipedia.org/wiki/Customer-relationship_management (https://en.wikipedia.org/wiki/Customer-relationship_management)](https://en.wikipedia.org/wiki/Customer-relationship_management)

[7] *Unix Time* [https://en.wikipedia.org/wiki/Unix_time (https://en.wikipedia.org/wiki/Unix_time)](https://en.wikipedia.org/wiki/Unix_time)

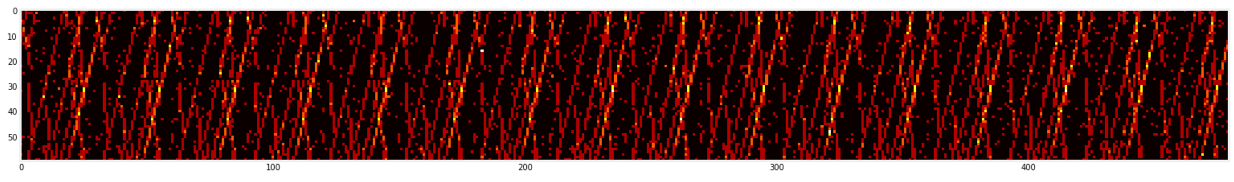
Exploratory Visualization

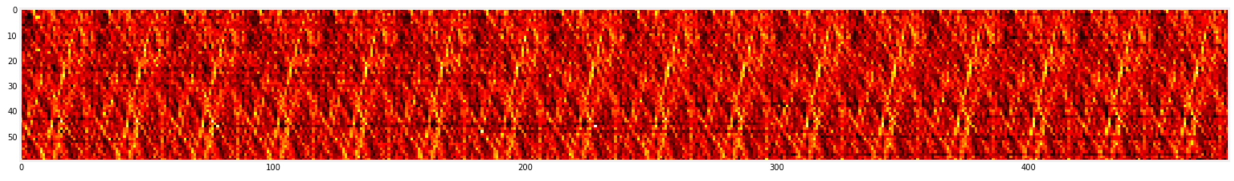
Bellow is two barcharts showing the magnitude and sparsity of during a random 6 hours. First one with higher coverage, more clients.
Second with fewer. This shows the clear recurring pattern and jitter.



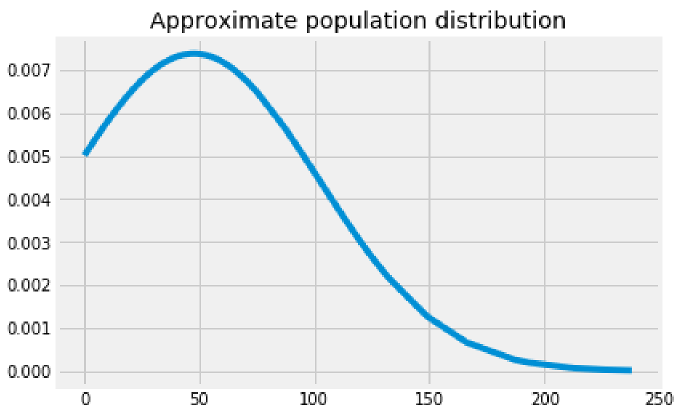


Bellow images where generated to observe and look at these patterns from another view point. Both images span 10 days where every pixel top left to bottom right represent minutes. Upper image shows an approximate of 8 clients and lower around 209, just to get a sense of different magnitudes.
We like you to pay particular attention to the diagonal lines, that clearly shows an negative and positive offset around the 30 minute interval. It's easy to see the interval through the repetitive pattern it generates. Also notice how these different offsets create a sort of cross weave that show local peaks in the form of light yellow when interference occur.





Bellow graph represents the distribution of clients per area. Notice the average pretty close bellow 50 clients.



Algorithms and Techniques

We'll implement an own method where we predict last minute by a moving average. We then calculate the percentage to get a relative distance between last minute and our prediction. This percentage is then checked against a threshold to evaluate the anomaly and trigger a potential outage recording. We will look more out of a code perspective in the implementation section.

We will implement what we could call Sparse Binary Weighted Moving Average. Sparse binary weighted meaning the weight multiplier is either 0 or 1 and sparsely distributed by the half-life frequency interval.
But instead of actually multiplying we will just get the discrete minutes by the interval detected. Thus just skipping the zeros.[8]

If W_m is our sliding window size in minutes and I is the measured interval.
Then n is the number of half-life DHCP lease intervals we want to backtrack.

$$n = W_m \backslash I$$

If x defines our counter values, m is the minute then θ_n is our sparse binary weighted moving average:

$$\theta_n = \frac{x_{(m-I)} + \cdots + x_{(m-In)}}{n}$$

Then we take the current minute value x_m and divide by our sparse binary weighted moving average θ_n to get the current values multiplier of the average M:

$$M = \frac{x_m}{\theta_n}$$

And $M \times 100$ is our anomaly percentage.

We will use a similar algorithm to calculate and verify the frequency interval of the data. For this we implement a function loosely inspired by discrete-time fourier transform (DTFT).[9]

For every frequency interval we want to test we get the Sparse Binary Weighted pattern as above, but instead of average we get the standard deviation. Then we get the argmin deviation from the range of intervals, thus getting the interval which yielded the least variance. As we will see in the implementation section and in our case, with very repetitive pattern, this will pin point the frequency interval very accurately.

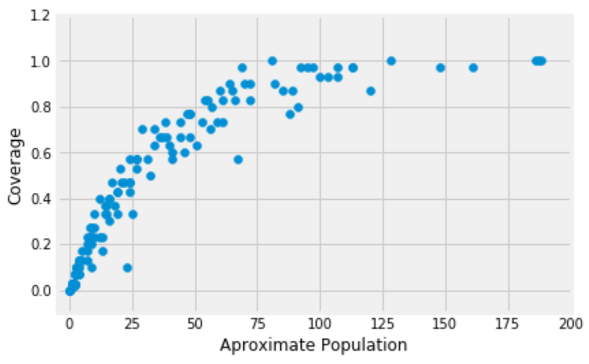
[8] *Moving Average* https://en.wikipedia.org/wiki/Moving_average#Weighted_moving_average (https://en.wikipedia.org/wiki/Moving_average#Weighted_moving_average)

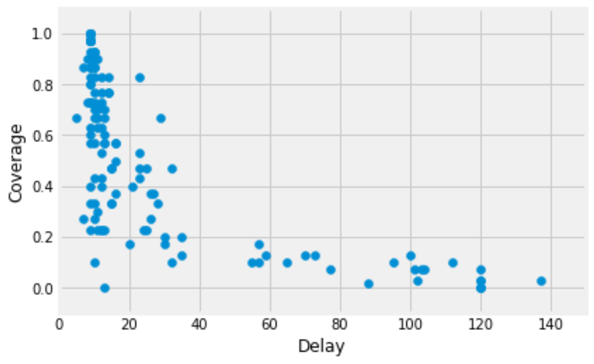
[9] *Discrete-Time Fourier Transform* https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform (https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform)

Benchmark

One benchmark that makes sense in this case is how early we detect an outage. Because that is what we essentially try to achieve with this method.
Normally when measure DHCP clients you measure how many active clients or leases there are over time. [10] That means a client gets removed from count first when lease expire. In the case of 1 hour we would start see a gradually decline after 30 minutes as the half time of the lease directly affects the delay.
This means that we would have a minimum delay of 30 minutes before we have any degrade to detect anomaly on, no mather what method used. By predicting the renewal event we mitigate that. The question is when and by how much?

Let's check.





Even though the exact delay dependes on which minute outage happen in relation to the exact distribution of covered minutes, we have a pretty clear elbow. To keep delay low we want to stay above 20% coverage.
The relation between population (number of clients) and coverage it generates is also very strong. In theory we could have 100 clients all clogged at minute 29 on the interval. This shows that it gets distributed and even out.

A real benchmark would require a parallel data counting and aggregation pipeline and there is no such setup. So we don't have anything to actually benchmark against. That is why we instead benchmark the model vs distribution. Which is very fundamenal and a valuable insight in this particular use case.

[10] *impl. example Monitor DHCP pools* http://folk.uio.no/trondham/software/dhcpd-pool.html (http://folk.uio.no/trondham/software/dhcpd-pool.html)

III. Methodology

Data Preprocessing

Raw data is stored as timestamps per ipadress and service in the backend of the DHCP server. In this case an in-memory store with timestamp indexed in the form of a MySQL NDB cluster slave.[11]
The data is sampled per minute then map reduced to a counter per network and zip code. Data is then written to disc where we can extract our test period and population.

In the next stage we attach the customer call counters by correlating zip code and network operator as described in the data exploratory section. As this might be sensitive information even in aggregated form we will use a placeholder for network and the areas zip codes shifted.

Lastly we produce the output csv format used in this project.

No more preprocessing is done.

[11] *NDB Cluster overview* https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster-overview.html (https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster-overview.html)

Implementation

My initial implementation plan contained the following steps

1. develop a function to estimate the most significant frequency
2. Set up a simulation loop itterating minute by minute with a sliding window
3. measure the anomaly percentage between last minute and a moving average.
4. set up a state machine to generate events start, detection, end, clear
5. output indexed events
6. visualize and evaluate

The pseudo code function for most significant frequency mentioned under Algorithms & Techniques looks like this:

```
get_min_std_interval():

    for interval between 1 and 120:
        get set of indecies aligning interval
        get mean average of the spread ( standard deviation )

    return the argmin spread of all intervals tested
```

This worked really well as we will see later in the Model Evaluation section. Also as stated before, we know the actual predicted renewal interval due to the DHCP server configuration for this network and protocol standard. This is just a independant method to dynamicaly and automaticaly predict the interval.

The method for whats called "Sparse Binary Weighted Moving Average" in the Algorithms & Techniques section is really straight forward and simple in code. As we have already predicted the interval we can exploit this by range stepping, where step is the interval predicted.

```
range(start, stop, step)
```

'start' dependes on how far back we want to lookback and evaluate and 'stop' defines this, latest minute.

At the core we just take the mean average of all preceeding interval minutes except the one we want to predict and evaluate.

Where 'right' is the minute predicted, set 'left' back in time:

```
left = right - lookbehind
```

get the binary weighted filtered window:

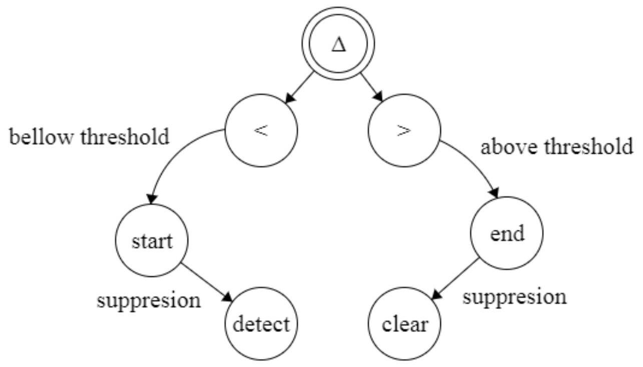
```
window = list(range( left, right, interval ))
```

Calculate a relative distance between the average predicted value and latest:

```
avg = np.mean( data.dhcp[ [x for x in window[:-1] if x is not skip_indices] ] )
last = data.dhcp[right]
mul = last/avg if avg and last else 0.0
```

This relative distance is then evaluated in a state machine to trigger events.
To avoid regression toward the mean I keep track of indicies marked as anomaly and skip those in the window analyzed.[12]
We will take a look at a twist to the lookback in the Refinement section where we adjust this window dynamicaly. Δ below represents the relative distance.[13]

start	relative distance bellow threshold
detect	suppresion time passed
end	we are back to normal
clear	reserved post anomaly window passed



[12] Regression toward the mean https://en.wikipedia.org/wiki/Regression_toward_the_mean (https://en.wikipedia.org/wiki/Regression_toward_the_mean)

[13] Relative distance https://en.wikipedia.org/wiki/Relative_change_and_difference (https://en.wikipedia.org/wiki/Relative_change_and_difference)

Refinement

The itterations done where actually a little diffrent than the planned implementation. at first i experimented with a continous cumulative sumary over the whole interval. Eventhough that aproach even out the curve and fill in blanks, I not only realised that it added overhead to the preprocessing pipeline, but also blurred the pattern. So what cumulative sum essentially does is just to add a kind of blur filter. In this case we want to preserv the resolution as much as we can. We'll discuss possible benefit though in the improvement section.

Stages:

1. write some kind of outer loop to simulate the minute by minute process.
2. established a inner process/loop for the sliding window.
3. implemented simple visalization to see whats happening

As we already had estabished a strong sens for the interval through DHCP server lease time configuration, rfc protocol documentation and visual ques, I first implemented a variant of the sparse binary weighted moving average. With the interval frequency established we could simply use the skipping parameter of numpys arange to create a index filter for the window we're processing. As the most important intelligence here is the analysed frequency interval I wanted to establish a method for automatic analysis of the most significant possible interval by the data itself. So:

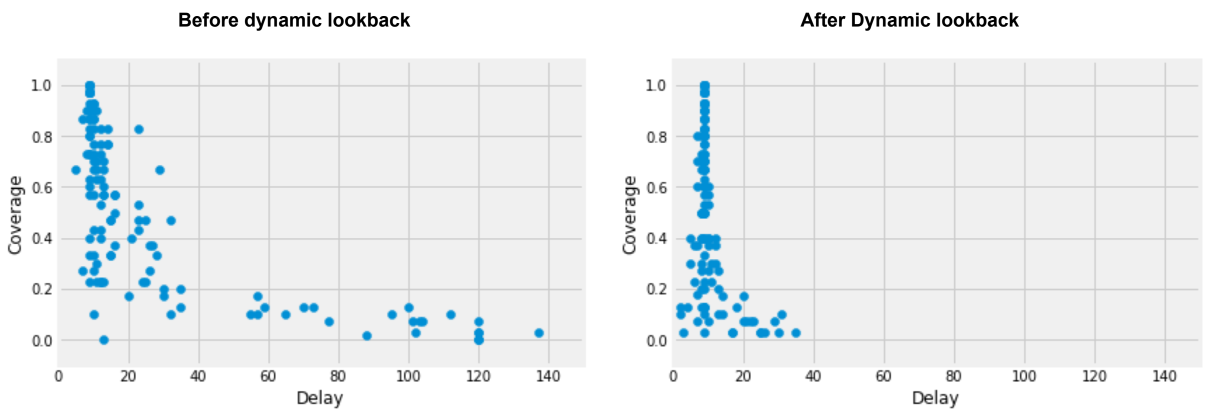
5. developed a method for predicting the overall interval frequency.

This is a key here as the overall inteval frequency is the most important component. From that we can approximate client population and temporal minute coverage.

6. implement state machine for event generation

After some itterations I implemented a dynamic lookback dependant on the detected anomaly window. Eventhough the 20% coverage threshold still holds, this little fix lowered the average delay alot. As we are keeping track of anomaly ticks by memorizing the indicies we want to skip. We use the length of this anomaly window to dynamicaly adjust the window by which we measure.

```
    _lookbehind = lookbehind + len(skip_indices)
```

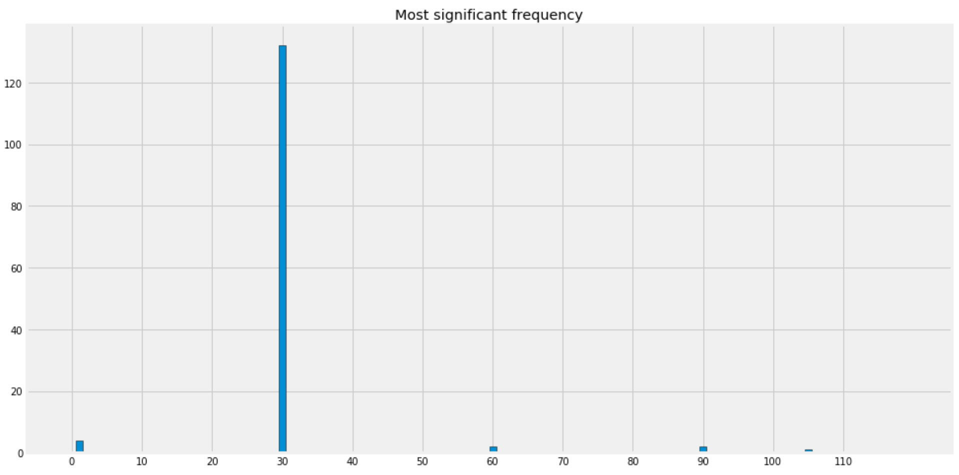


IV. Results

Model Evaluation and Validation

The spread of the clients over the lease time half life intervall frequency is a key here. This will directly impact the resolution of this method.

So how well could we predict the interval?



With resolution we refer to the delay between outage start and where we are confident enough to trigger a anomaly event. Its pretty easy to imagine the perfect coverage where all minute buckets are filled with a count.

In these cases we can expect a detection spot on the minute after suppress time. On the other end, having a small population or bad minute coverage will increase the delay and accuracy of the detection.

We can easy imagine that the most perfect coverage would be. That is, at least one client request per minute, every minute. This also means that the worst case is having only one client, or everyone stacked the same minute late in the time spectrum. So how long it will take until detection depends on the pattern of empty minute buckets over the time spectra of the frequency interval.

We can also conclude that the worst case will be as slow as measuring the actual active client count, where we need to wait the full lease time until degrading starts occuring in our graph.

If the distribution is right scewed it'll take longer than if the distribution is left scewed from or towards the outage start.

Thats why we want to consider low populated areas and distributions with low spread over the interval frequency outliers.

The coverage can be calculated simply by dividing minutebuckets by Frequency interval.

So the actual delay we will experience is dependant on the minute of outage occurrence ($OutageMinute$) given the spread of measurements pattern captured ($CoveragePattern$).

$$Delay = (OutageMinute|CoveragePattern)$$

Justification

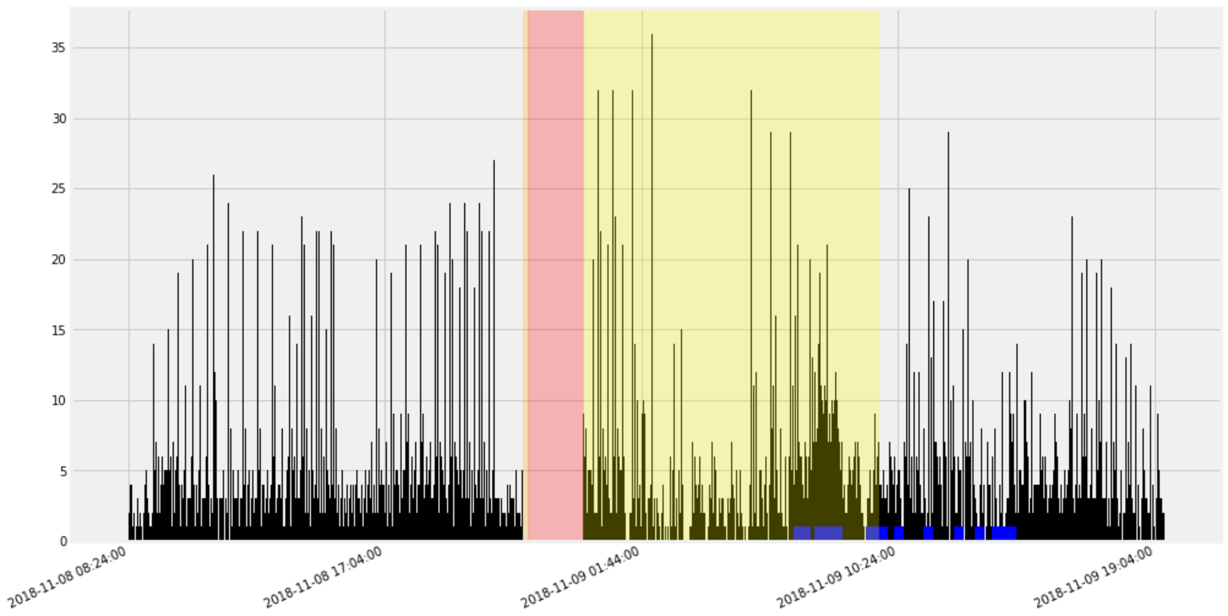
It is easy to justify this model due to the lack of prior resolution to the problem. And I think that we'd shown that you can get pretty decent result with a simple variant of moving average.
Also the big winning here was show establish a good limit for outliers regarding coverage. Even with 20% coverage in our test population we had max 18 minutes delay (including 10 minutes detect threshold) and still covered 98.33% of all clients.
Also areas covered represented 99.15% of customer calls.

V. Conclusion

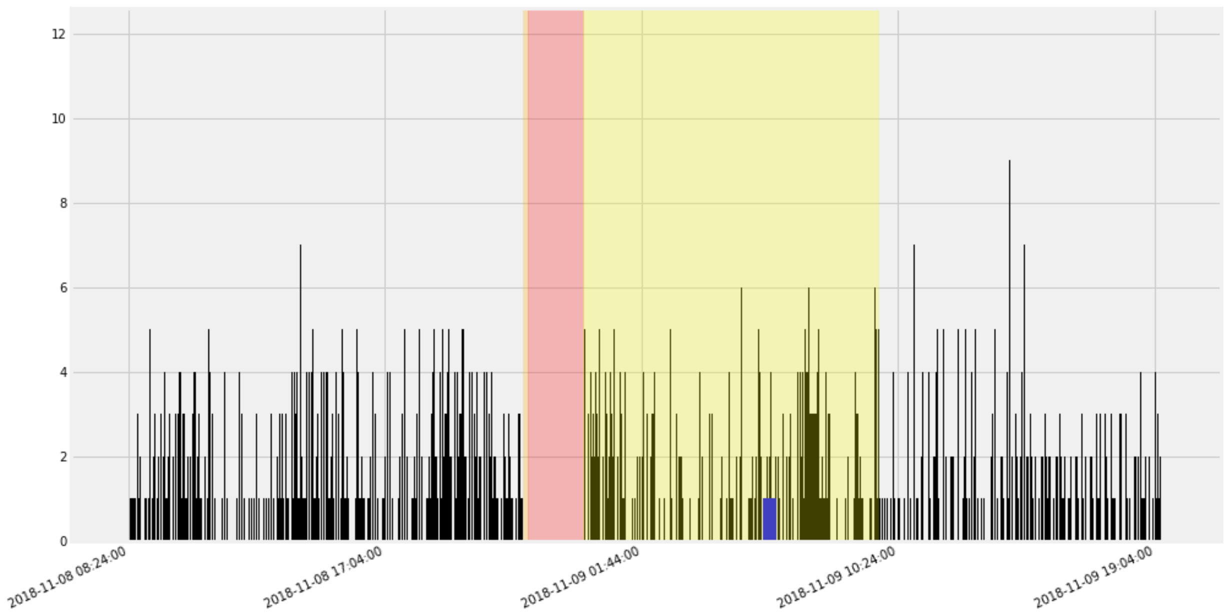
Free-Form Visualization

Here are some examples of detections. Red area is the active anomaly. Yellow area to the right of it is added to visualize arbitrary post outage active voice prompt. Orange area to the left is the predicted start. That is when we started suppose a outage, in contrast to the red "anomaly detected". We simulated an outage between 2018-11-08 21:44:00 and 2018-11-08 23:44:00. The blue bars is actual customer call ticks from the specific, region but is cosmetic in this rigged simulated outage.

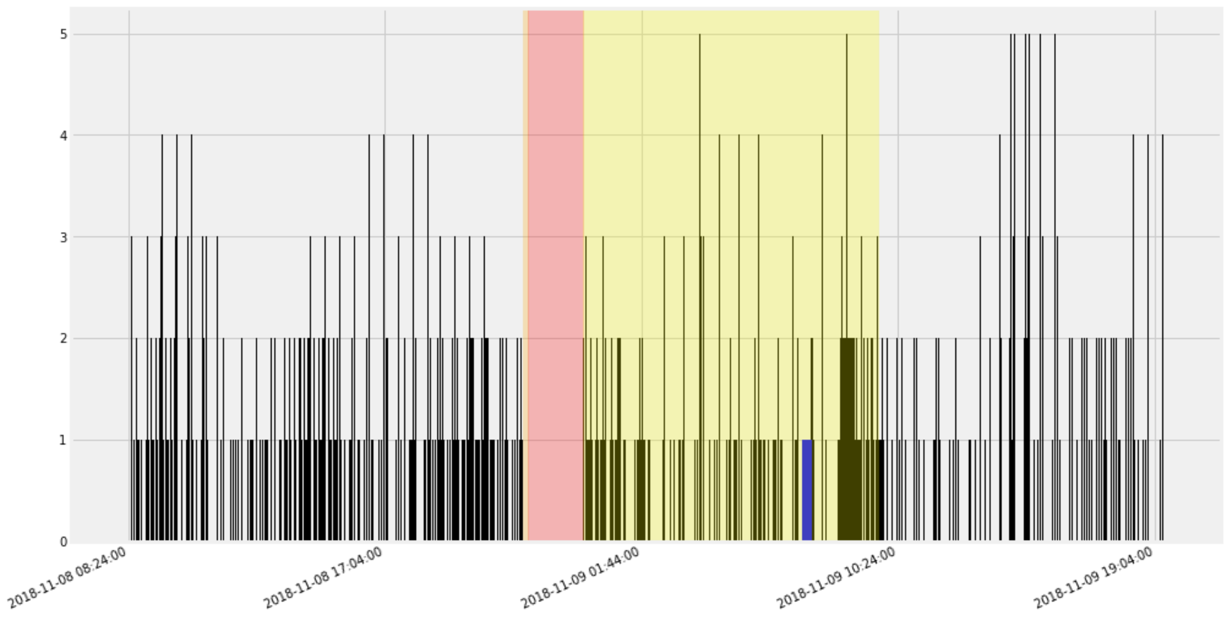
Approximate population: 187, coverage: 90%



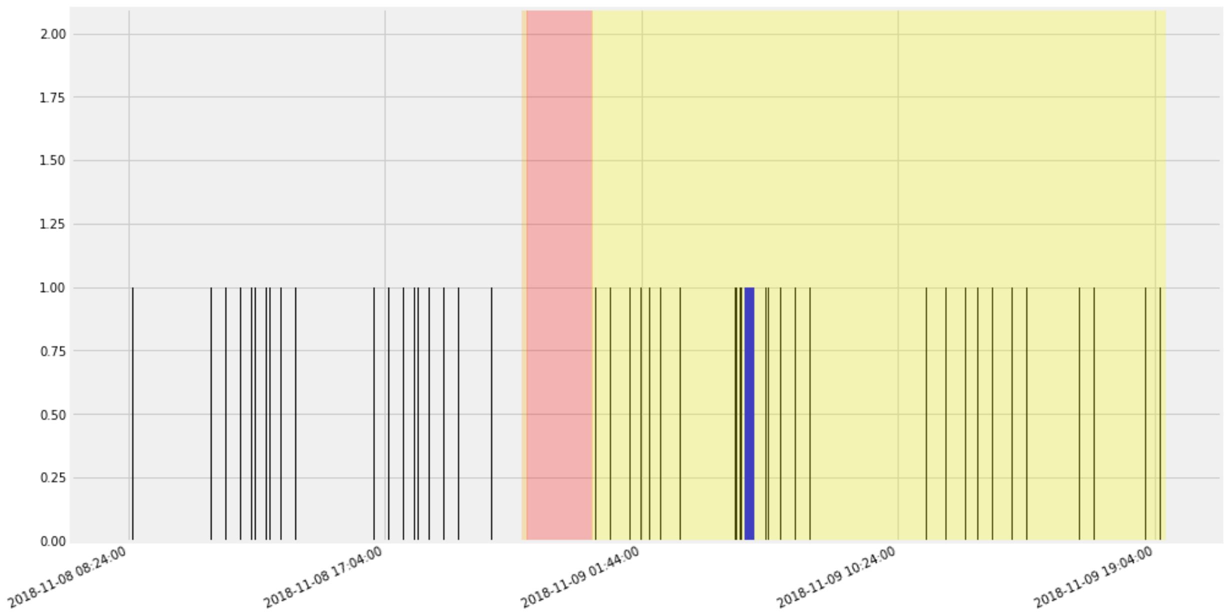
Approximate population: 47, coverage: 70%



Approximate population: 29, coverage: 60%



Approximate population: 2, coverage: 7%



Reflection

This was a first look and exploration of a very simple method to tackle and exploit the very special use case of detecting anomalies in jittered recurring time descrete counter patterns. In this particular project we used data generated by clients doing IP address renewal by the Dynamic Host Configuration Protocol.

One of the hardest problem with this was to keep it simple. One question that might arise is, why not use call counters as target for a supervised model. The reson for that is the minimal and sparse data. you would shift the problem toward finding and cropping outliers.
Also the time frame was hard for me personaly. To pick something so specific and "ungoogleble" might not have been vise but I wanted to explore something where I had to bruteforce my own thinking. Also to be able to anonymize and put this sample data online for others to maybe take on, feels great.

And eventhough its very tempting to run off and copy-paste some RNN implementation from github I wanted to take a more experimental heuristic first approach by implementing things more or less from scratch, testing my logic and intuition. Another huge reson that I wanted to aproach this as simple as possible was production and scaling. As there are over 12000 distict area operator combinations, in production we need to process many streams and models in parallel.

Improvement

When we have this baseline and analysis it would be interesting to jack in other methods. This was the simplest aproach I could think of and at the same time generate meaningful events.
To implement this in production we could implemented and deploy microservices subscribing a messaging buss for the input

and api towards voice responce and ticket system, just send syslog or route the generated events some where else.

One thing thats outside the scope of this project, but I am eager to try out, is a unsupervised aproach of those diffrent offset patterns shown in the exploratory visualization section. It would be interesting to explore if diffrent kind of clients or devices follows predictable phases and jitter. That is, to predict clusters of device types by their offsetting pattern of generated DHCP renewal. At least analyse if there are correlations.