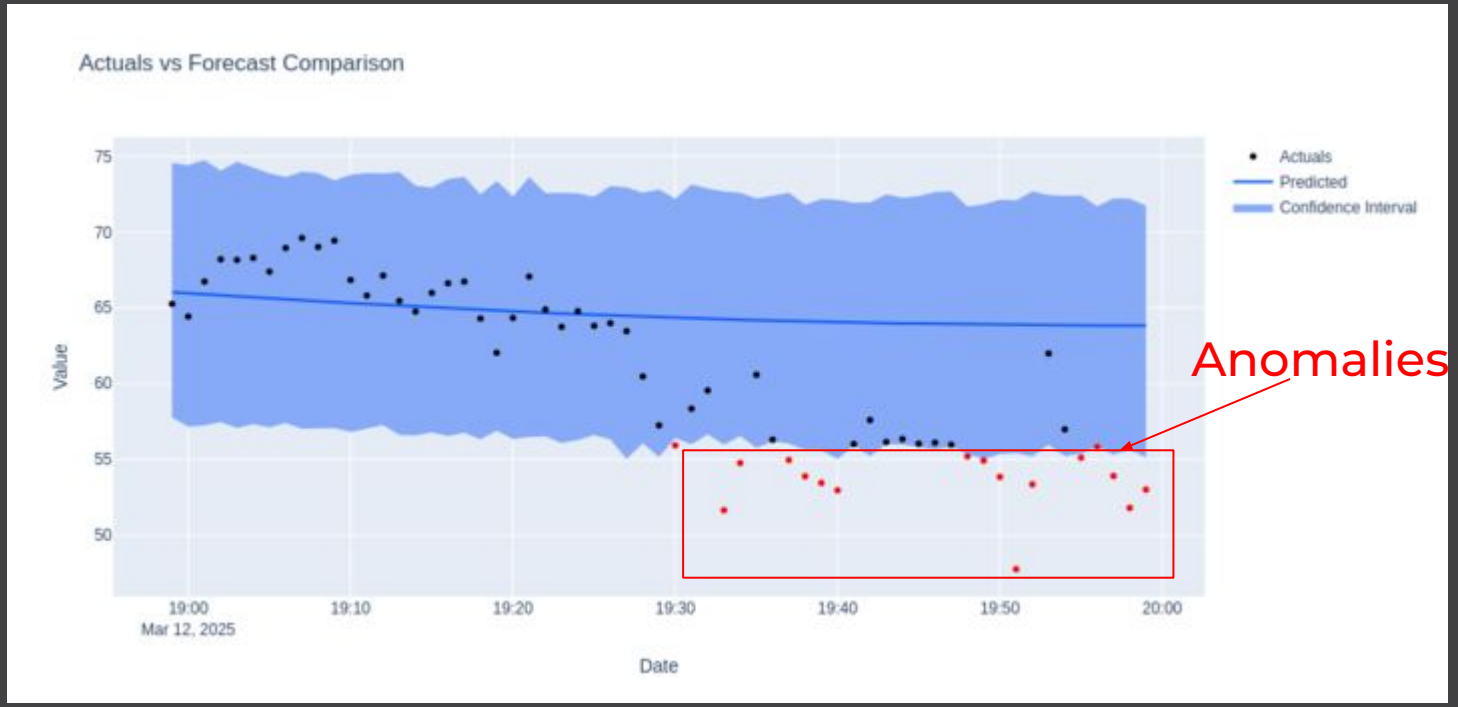
SNMP Data Forecast and Anomaly detection - Utilizing Meta's Prophet

A.J. Ragusa - GlobalNOC @ Indiana University

GlobalNOC
at Indiana University

The Goal/Results!



GlobalNOC
at Indiana University

We've been exploring forecasting for a little while

Things we have tried:

- Holt-Winters
- Line Fit
- Custom code with TensorFlow
- Throwing data at ChatGPT/Grok

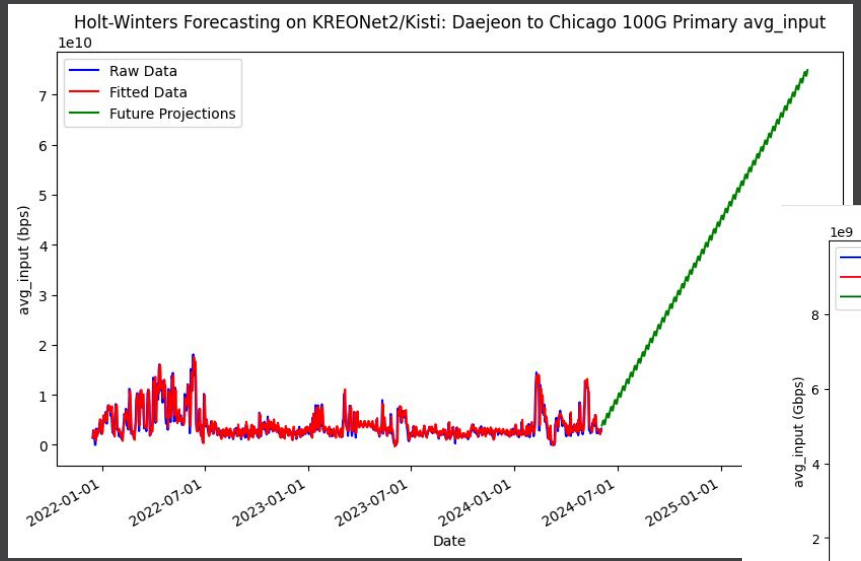
Most of these worked... sort of. Nothing really worked well enough for us to really use.

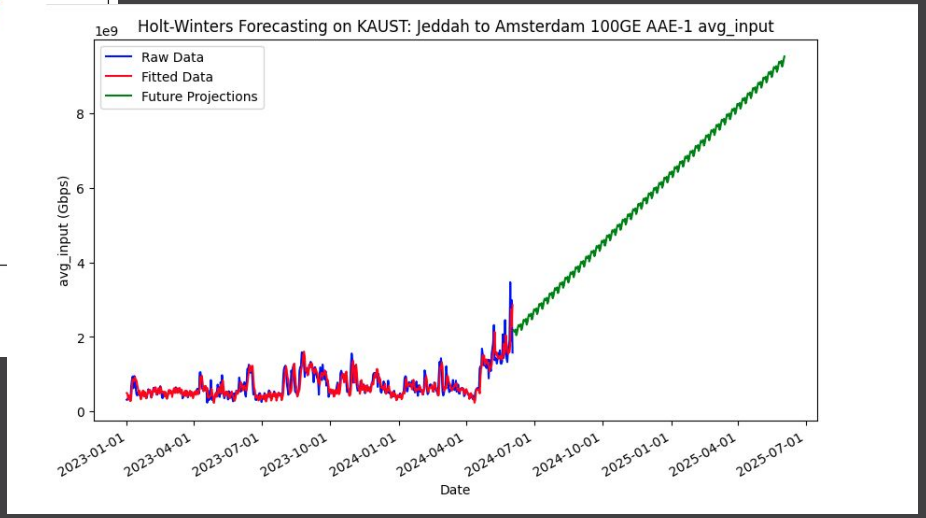
James D. at Internet2 suggested Prophet - so we took a look!

GlobalNOC
at Indiana University

Failed forecast images

Grok and ChatGTP both failed to forecast.... anything





GlobalNOC
at Indiana University

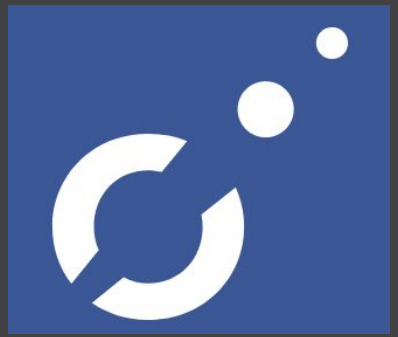
Prophet from Meta/Facebook

Prophet is designed to do forecasting of time-series data using yearly, weekly, and daily seasonality and holidays.

Its written in both R and Python making it really easy for us to work with

Its highly tunable, and does not require a GPU.

https://facebook.github.io/prophet/



One of the Primary Drivers

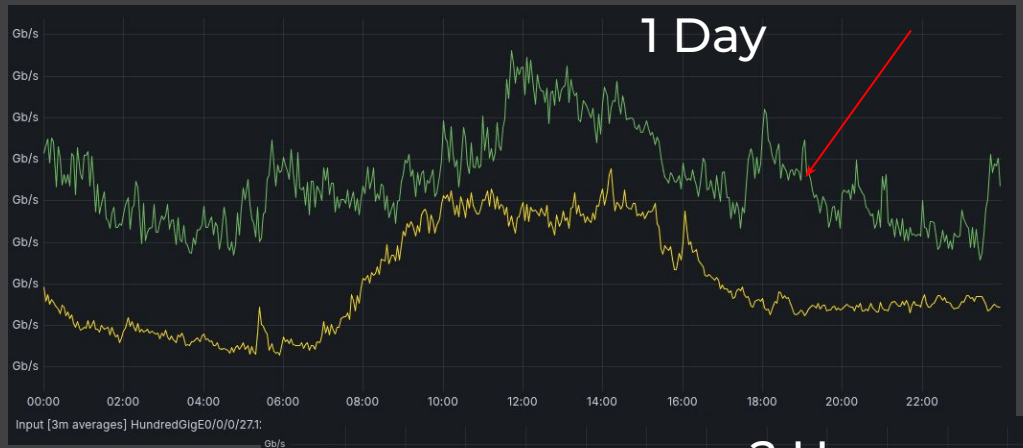
Internet2 asked us what we could do to help detect anomalies when a participant behind a connector has a misconfiguration that perhaps drops their traffic going through Internet2

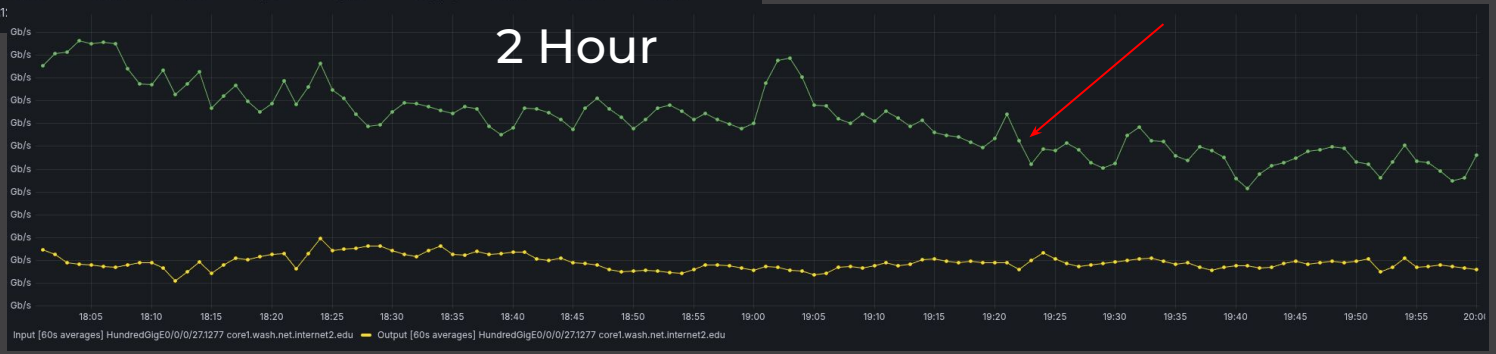
There was a specific event that led them to ask for this capability

The question: Can we automatically detect these situations and create alerts for network engineers to investigate in near-real time? Not days or weeks later but within a few hours?

GlobalNOC
at Indiana University

Here is the Data for the connector





Anyone see the anomaly?

GlobalNOC
at Indiana University

First we need data!

Good news we have SNAPP/TSDS!

The GlobalNOC SNAPP Service backed by TSDS stores time-series data about many different aspects about the network

- Interface Utilization
- Optical data (txpower, rxpower, etc..)
- Temperature
- CPU
- Memory
- BGP Prefixes
- etc..

GlobalNOC
at Indiana University

SNAPP/TSDS API

The SNAPP/TSDS API has a programmatic API allowing us to fetch the data we need. Specifically it implements a query language very similar to an SQL like statement:

“get aggregate(values.input, 3600, average) between(now-1d, now) by node,intf where node = “<node>” and interface = “<interface>”

The results are a JSON structure with 24 values of the hourly average for the Input of the specified interface

GlobalNOC
at Indiana University

TSDS is Perl based so we can't use Prophet

Instead we are going to write a wrapper Web-service that will fetch the TSDS query and then run Prophet based on the passed parameters and return the actual and predicted data

First fetches the data from TSDS

Convert into pandas dataframe (pandas is a library designed for data manipulation and analysis and Prophet expects it as an input)

Initialize Prophet

Make Future Dataframe

GlobalNOC
at Indiana University

Code Example

```python
@api.get("/get_forecast")
def prophet(query: str, target: str, prediction_end: int, capacity: int, bin_size: int, seasonality_mode: str, floor: int | None = 0):
    #--- the way this works is by running whatever tsds query that they send us.
    #--- then we look in the results for the 'target' that they sent us. So if the target isnt there then that will be an issue.
    client = wsc.WSC()
    client._strict_content_type = False
    client.url = os.getenv("TSDS_URL")

    results = client.query(query=query)
    result = results["results"][0][target]

    #--- so we have the data they want to run prophet on. Now we need to put it into some pandas dataframes and do some cleanup on it.
    df = pd.DataFrame(result, columns=['ds','y'])
    df['ds'] = pd.to_datetime(df['ds'], unit='s')

    #--- its possible for null values to come back from tsds. prophet doesnt like nulls. So we need to fill them in some how.
    #--- in this case we will foreward fill which pulls the last good value foreward in time until the next good value which
    #--- closes the hole in the data.
    #df = df.fillna(method='ffill')
    df.ffill()

    #--- sometimes TSDS will return future dates to us as a NAN. In the previous step we foreward filled
    #--- NANs with the last good data point. But since there is no future data the last data point can
    #--- be carried foreward into the future. So we need to cut out any dates from the future so that
    #--- prophet is the only thing making future predictions.
    now = datetime.now()
    filtered_df = df[df['ds'] <= now]
```

Talk to TSDS to fetch data

Convert to Pandas

More Code

```
#--- we generate future times for prophet to make predictions for. However I am uncertain that this is correct since we're generating
#--- futures days but the data coming back from TSDS can be in forms other than days (IE hours). So this may not be the correct way to do
#--- this. We should investigate more.
future_date = datetime.fromtimestamp(prediction_end)
delta = future_date - now
periods = delta.days

m = Prophet(seasonality_mode=seasonality_mode, growth='logistic')
filtered_df['cap'] = capacity
m.fit(filtered_df)

if(bin_size == 3600):
    periods = periods * 24
    future = m.make_future_dataframe(periods=periods, freq = 'h')
else:
    future = m.make_future_dataframe(periods=periods)

future['floor'] = floor
future['cap'] = capacity
forecast = m.predict(future)

#--- prophet keeps the data in two places. When you look at how prophet calls the graphing functions internally you see that the prediction data is
#--- stored in our forecast object. But this doesn't contain the original data that we fed in. So we need to grab both the original and the forecast
#--- data and stick it into a single object that we can return.

merged_df = pd.merge(forecast, filtered_df, on='ds', how='left')
data = merged_df.to_dict(orient='records')

return {
    "tsds_query": query,
    "data": data
}
```

Initialize Prophet

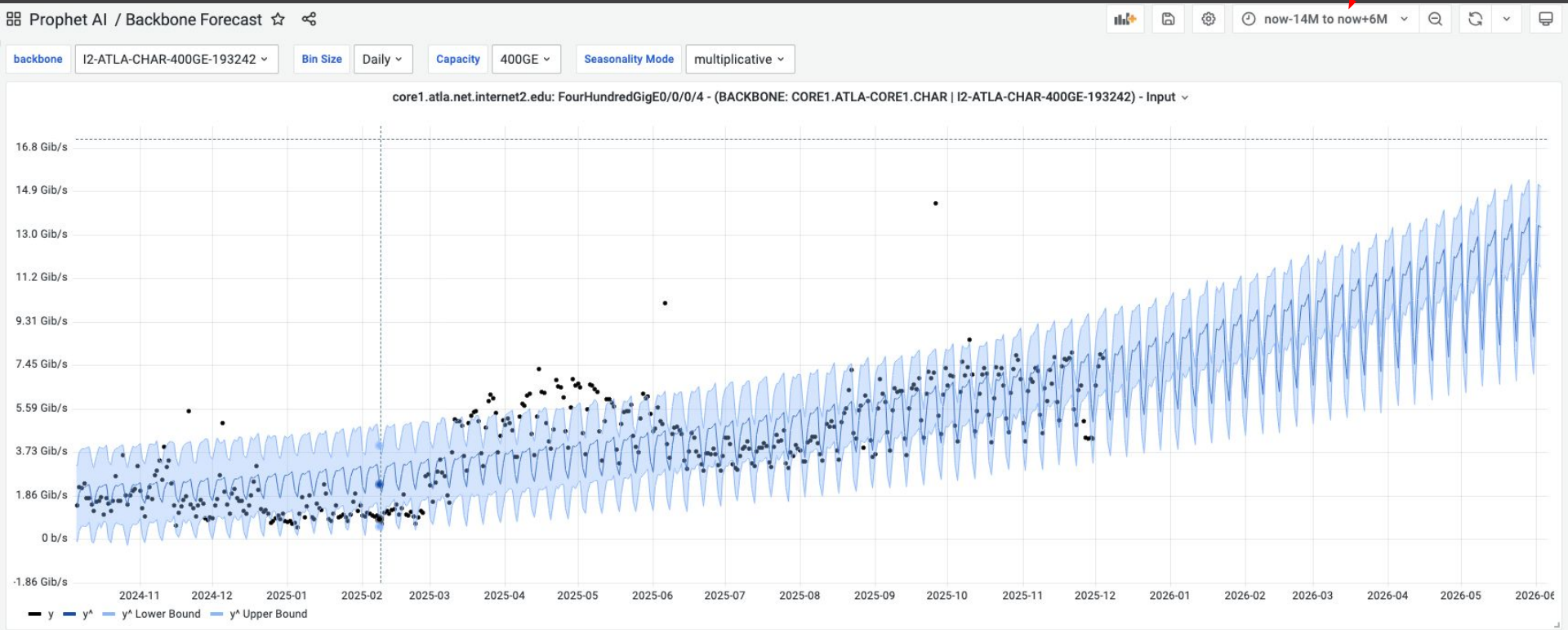
Make prediction

Return data

GlobalNOC
at Indiana University

Running our existing data through Prophet!

Grafana can do future timestamps!



Tuning prophet!

Prophet really does not take a lot of additional parameters to tune:

- Bin-Size... what size time chunks are you sending (second, minute, hourly daily)
- Seasonality Mode:
  - Multiplicative
  - Additive
- Capacity: Maximum capacity can reach (keeping things in-bounds)
- Floor: Lowest things can go (in our case 0)

This is only a proof of concept demo

This does work and is available to Internet2 Engineering (not publicly available at the moment)

We still have a lot of work on forecasting specifically around seasonality!

One of the very interesting things that Prophet will let you do is add custom seasonality like US holidays

We can also attempt to add things like Spring break, Winter break and summer break into the seasonality models

This work will become more critical as we work on Anomaly detection

GlobalNOC
at Indiana University

How do we find anomalies with this?

Prophet does not say anything in its documentation about Anomalies and we got a little stuck at this point (*note we are software engineers not Data Scientists)

All we have is a prediction how can we use this to detect anomalies?

(Luckily... James D. really likes us or hates us because he keeps pushing us to detect anomalies)

GlobalNOC
at Indiana University

What if...

Finally our idea becomes kind of obvious

Generate a prediction for the last hour for an interface and then overlay the actual last hour of data on top of it. Anything outside of our “confidence interval” is then considered an anomaly

Then we can configure a % of anomalous data as an alarm for engineers to investigate!

Is it possible? What do we need to do to try this out? We need an event to attempt to see.... Of course we have the exact example that spurred on this event!

GlobalNOC
at Indiana University

Proof of concept code (Jupyter Notebook)

```
now = int(time.time())
hour = 3600
day = hour * 24
week = day * 7
month = day * 30
aggregate_size = hour / 60

days_back = 3 #--- how many previous days worth of data should we use to train the model

end = 1741809600
#end = now
start = end - days_back * day

[redacted]

confidence_interval = 80 #--- in percent
prediction_length = 1 #--- in hours
```

We can tune these to try and improve results

GlobalNOC
at Indiana University

Those important values

- Confidence Interval: The Confidence Interval is the “range” in which a % of the points should be guaranteed inside the bounds. Essentially if we set it to 100% the “range” of our confidence interval is going to extremely large to account for any datapoint
    - The lower the confidence interval the more likely we will have data points fall outside of the expected range
- Days back: this is allowing us to select how much data to train the model on in this example we did 3 days so we could get quicker answers from TSDS using high res. If we go much further we would need to return a lower resolution, and that would provide lower resolution result. Ideally we would include ~ 1 year worth of data but 1 year of high-res data is relatively large and difficult to process (were working on it)

More code examples (I'm gluing cells together)

```
m = Prophet(interval_width=confidence_interval / 100)
m.fit(df_train)
future = m.make_future_dataframe(periods=prediction_length * 60 + 1, freq='min')
future
```

```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

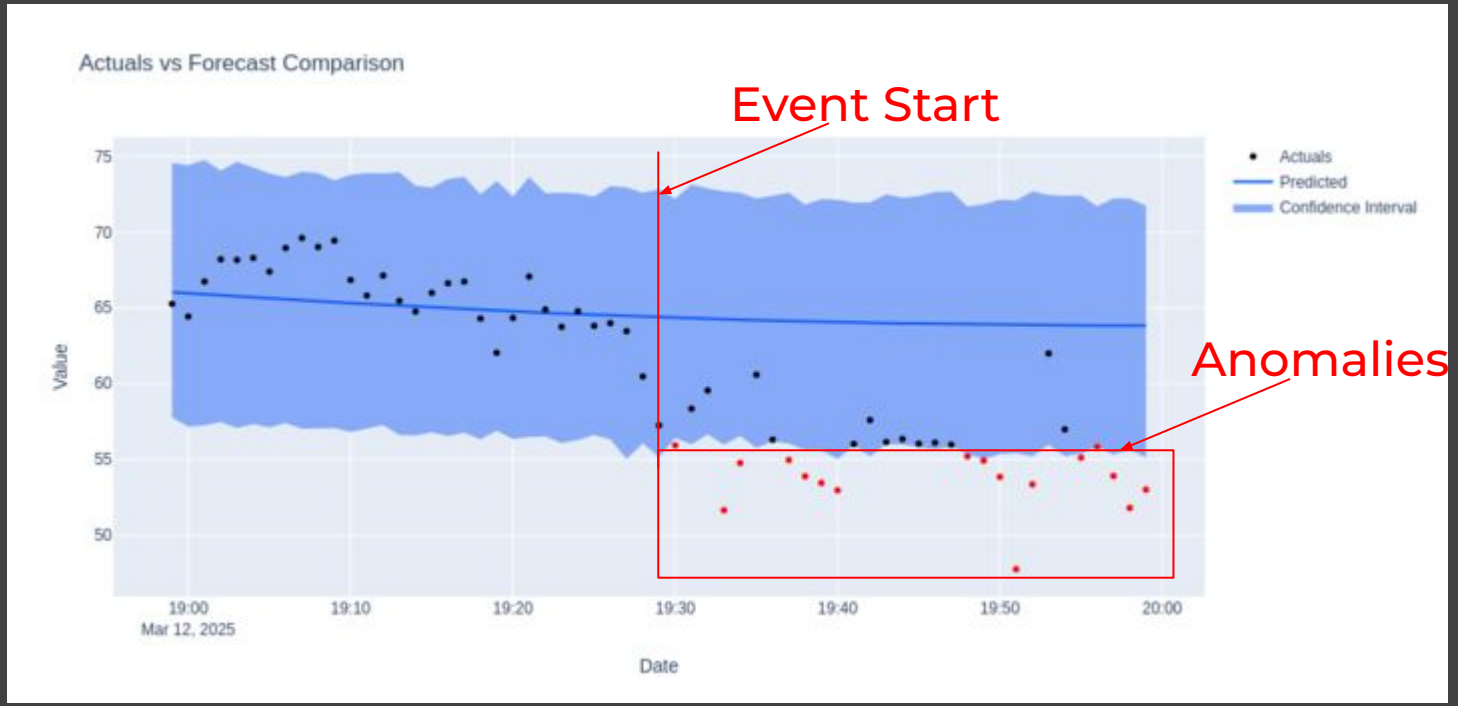
Do prediction (*note confidence interval)

```
# Get the portion of the forecast corresponding to the prediction period
#forecast_future_only = forecast.iloc[-(12*60):]
forecast_future_only = forecast[forecast['ds'] >= split_timestamp]

# Combine forecast with actuals from df_predict
# Make sure both dataframes cover the exact same dates for a direct merge
comparison_df = pd.merge(df_predict, forecast_future_only[['ds', 'yhat', 'yhat_lower', 'yhat_upper']], on='ds')

# Actuals
# Determine if points are inside or outside the CI
comparison_df['within_ci'] = (comparison_df['y'] >= comparison_df['yhat_lower']) & \
                             (comparison_df['y'] <= comparison_df['yhat_upper'])
```

Compare actual value with the upper and lower confidence levels

GlobalNOC
at Indiana University

The Result - The actual incident I2 brought to us



GlobalNOC
at Indiana University

What are we doing next?

We have recently released TSDS2 and are integrating the Prophet forecasting directly into the query language (no need for our Add-On web service).

We'll be able to run the forecast as a native query via the API

We'll run the forecast for the last hour and overlay the last hours worth of data and count how many anomalous data points exist and then create an alarm based on it.

GlobalNOC
at Indiana University

Profiles

One aspect as we ran this through different interfaces across the network is that we will most likely need different interface profiles based on the “role” the interface plays in the network

Adjusting values like the Confidence interval as well as days back, and seasonality we hope to be able to be able to enable Anomaly detection on many of our customer networks with a relatively low false-alarm threshold for anomalous traffic patterns.

Any data scientists out there?

If you are doing work in this space or have any thoughts (especially on this approach) we would love to hear from you :)

Questions?

aragusa@globalnoc.iu.edu

GlobalNOC
at Indiana University