Improve Statistic Functionality of Kea

Proposal for GSoC 2018

Mentor: Tomek Mrugalski

—

About Me:

Contact Information:

- Name: Souradeep Chakraborty
- University: Birla Institute of Technology and Sciences, Pilani, Goa Campus
- Degree: Undergraduate Student of B.E. (Hons.) Electronics and Electrical Engineering
- Email-id: saura.chak@gmail.com
- Time-Zone: IST (UTC + 5:30)
- Github: https://github.com/sauradefy99

Personal Information:

My name is Souradeep Chakraborty, an undergraduate student at BITS Pilani, Goa Campus, India.

I use Ubuntu Xenial along with Kali Linux as my main workstations. I have been coding in Python (>2 years of experience), C/C++ (>2 years of experience) and Java (>1 year of experience). Apart from mainstream programming languages I have used Matlab, R and

Octave for various projects. Personally if I had to pick, I'd say Python is my favourite coding language as it is much easier to work with.

I had been selected for the prestigious Indian Statistical Institute for its B. Stat course through the ISI 2017 exams. I am not easily daunted by advanced math and love the complexities and nitigrities it presents.

I have also taken fundamental mathematics courses offered by my university like Probability and Statistics, Introduction to Linear Algebra and Calculus I. Apart from these, I have taken various machine learning projects and online courses.

The online courses I have taken include:

- Machine Learning - Stanford University, Professor Andrew Ng
- Deep Learning - deeplearning.ai
- CS50 - Stanford University
- Data Structures and Algorithms - IIT Delhi
- CS231n Convolutional Neural Networks (ongoing) - Stanford University

As part of the courses above, I had to do various coding assignments weekly.

Some projects I have undertaken include a handwriting recognition model that used MNIST datasets to predict handwriting with an accuracy of upto 98% <https://github.com/sauradefy99/Handwriting-digit-recognition-SVM->, an email spam filter based on support vector machines <https://github.com/sauradefy99/Email-spam-filter> and a spam-filter chatbot that is used to filter out reddit spam comments based on deep learning <https://github.com/sauradefy99/reddit_spam_detector_bot>.

I am familiar with Git/GitHub and am thorough with its usage.

Introduction:

Project Idea:

Kea is an open source DHCPv4/DHCPv6 server being developed by Internet Systems Consortium (https://github.com/isc-projects/kea). As such Kea generates many data points, dubbed, statistics that represent various types of data ranging from traffic volume, types of traffic, resource utilization etc.
The idea is to use various statistics generated by Kea, to its fullest extent.
So far not much has been done with the vastitude of data generated while the options are limitless. The goal of this project would be to effectively increase the functionality of statistics for deeper analysis and further use, as for example a predictive regression model that raises an error when a statistic does not perform as it is expected to. The project also tries to tackle the issue of address mismatch when multiple instances of Kea are running.

Importance of the project:

Statistics can be used for getting a deeper understanding and insight of time dependent variables, however none of this is possible unless multiple statistics can be retrieved. The current statistics library only supports singleton statistic retrieval. Successful completion of this project would mean the ability to retrieve a number of statistics based on a given time frame or a given number. Apart from this, the project goes on to propose how we could use simple statistical operations on data points generated by kea for better housekeeping and maintenance. Data points could also be used for error prediction using simple linear regression as a start and could then be developed and further worked upon once the foundations are set.
All of this would then overall provide a much better user experience and a smoother operation of the DHCP servers.

Existing issues: (Rough divisions of the problems this proposal tries to solve)

1. The major drawback of the current statistics library is that there is no way to store multiple statistics. So far only a single statistic can be retrieved and stored. **(Data Storage/Retrieval)**
2. Nothing is being done with the statistics generated. **(Data Usage)**
3. There is no way to visualize the data. **(Data Visualization)**
4. Currently there is no way to predict if the server is going to run into an error based on the statistics generated. **(Prediction Models)**
5. Issue of address mismatch when multiple instances of Kea are running. http://kea.isc.org/ticket/5572#no1 **(Ticket #5572)**

Tackling the issues:

I. Data Storage:

Using the github link https://github.com/isc-projects/kea/blob/master/src/lib/stats/stats_mgr.h as reference:

First and foremost is the retrieval of multiple statistics. This can be done by defining a function that already has a prototype namely,

```
void setMaxSampleAge(const std::string& name, const StatsDuration& duration);
```

What this function is essentially made for is to house a set of statistics generated across a given duration.

Samples that are older than the duration will be discarded. Implementation techniques may include a FIFO data storage unit like Queues.

Another function of prime importance is

```
void setMaxSampleCount(const std::string& name, uint32_t max_samples);
```

With the help of this function we can generate a certain number of statistics.

Once more than one sample is supported updation of the following function in https://github.com/isc-projects/kea/blob/master/src/lib/stats/observation.cc is necessary:

```
void Observation::setValueInternal(SampleType value, StorageType& storage, Type
exp_type)
```

II. Data Usage:

We can effectively calculate a multitude of statistical operations like standard deviation, which can then be stored as a factor to compare later observations for sanity checks.

For example, a simplified example of a function that displays an error when the change in value of a statistic is greater than the standard deviation of the statistic based on previous data:

```
void checkDev(SampleType value_current, SampleTyple value_previous, SampleType stdDev)
{
	if(abs(value_current - value_previous) > stdDev) //assumming stdlib imported for abs
		std::cout << "Warning, change in statistic is greater than standard deviation";
}
```

The retrieval of data for these functions will be done periodically based on time-defined getter functions within the shell script itself. If need be, commands like **netstat** or **ntop** may be used.

These functions can also be extended for creating margin of errors in the linear regression model suggested later on in the proposal. The statistic after being extrapolated through the predictive model would be checked to be within the margin of error and if not, a warning will be raised.

III. Data visualization:

Using open-source data plotting tools like **gnuplot** and/or **koolplot**, we can be able to chart and visualize the statistics newly generated. These can be used from the shell script itself and will thus provide ease of access to users.

Later on, a JavaScript code may also be written which displays these plots in the user's web browser when activated. The specific tool for the task will be chosen later, but Angular is one of the possible tools considered to be chosen.

IV. Prediction Models (Linear Regression):

We use prediction models involving simple linear regression on a number of data points specifically those which are expected to grow (for example, `pkt4-sent/pkt4-received` vs time) so that if the linear model predicts that the statistic will not grow in the future (or act the way it is supposed to), an alarm is raised.

Linear regression is a simple machine learning prediction model where we extrapolate a linear relationship between certain data, so that

```
y(x) = m*x + c
```

In case of Kea, x could be taken as time and y(x) could be taken as the behaviour of the statistic with respect to time. Depending on past values optimisation of m and c could be done such that future values of the statistic could be calculated.

In a linear regression model the major task is to calculate the slope based on previous existing data. This can be done in C++ itself with/without importing any additional external libraries.

Using linear regression, we can also add a margin of error or a confidence interval for future values for example if a statistic is supposed to grow and from its behavior we are extrapolating that in some time the statistic will soon stop growing we can raise a warning. The data for these predictive models could be retrieved from the extended statistic-get functions which would support multiple statistics based on a given timeframe.
A simple implementation idea could be to get the statistic's value at time t, $t+\Delta t$, $t+2\Delta t$... etc and generate a slope and then extrapolate the statistics behaviour at some time $T = t + n\Delta t$.

V. Issue of misplaced statistics:

The problem arrives when multiple servers connect to the same database. For example, two servers connect to a database that has a pool of 100 addresses. When the servers start, 30 addresses are being used. Each of the servers start, determines total-addresses statistic value is 100 and assigned-address statistic value is 30. Now suppose a new client

shows up in the network and gets new address from server 1. This server now bumped assigned-addresses to 31. Other client that already had an address is releasing its address and that is being processed by server 2. That server now decreases its assigned-addresses from 30 to 29.
Now both the servers show an incorrect value of assigned-addresses.

What I propose is the use of a global variable that tracks the progress of all the servers, so that any change in a local server is also reflected in the global variable. One of the pitfalls of this approach is the fact that there are a huge amount of statistics generated each second so the mechanism for delivering to the global counter must be efficient. This needs to be further discussed.

Timeline:

Pre GSoC period (March 27 - April 23)

This period will be used to discuss implementations of the functions to be modified/added along with the best data retrieval techniques. This time will also be used to get an overall better grasp of how Kea works and how other open source libraries can be extended onto Kea. The issue of ticket #5572 will also be discussed in great detail during this period.

Phase I: Implementing functions for supporting multiple-statistic storage (Data Storage and Retrieval)

I. Week 1, 2 (May 15 - May 29)

Finish and test out the implementations of the **`void setMaxSampleCount`**, **`void Observation::setValueInternal`** and **`void setMaxSampleAge`** and keep developing them. Working out the best way to store and retrieve the statistics based on the prior discussions with mentor. Write unit tests and finish the implementations. Final touch-ups to the deployed functions.

II. Week 3 (May 30 - June 7)

Once the storage is extended to support multiple data points, the immediate next step is to update all the getter-setter functions based on statistics. These include the **`statistic-get/reset/remove`** commands.

III. Week 4 (June 7 - June 14)

The next most important thing to cover is to give the user the power to control the limits of the storage, so that the user can decide exactly how much data to store. This will enable users to specify the max sample count and max sample age through commands or through the configuration files.

Phase I evaluation

Phase II: Adding extra usage to the statistic data set (Data Usage and Visualization)

IV. Week 5 (June 15 - June 22)

Start with the addition of functions to generate simple statistical measures like standard deviation. The statistics needed for the calculation would be retrieved through predefined get functions and will be reset after a specific duration of time. After each lapse of the predefined time period, the reset and get functions would be called in succession so that the data for the standard deviation is updated. Thus, the data can be effectively retrieved within the shell script itself. Another possibility might be to run a JavaScript program that retrieves the latest statistics and invokes the stddev function from the web browser itself, or by using commands like netstat or ntop to collect the data through shell itself.

V. Week 6, 7 (June 23 - July 7)

Next, functions will be needed which utilise the various statistical properties like standard deviation and calculate a margin of error for upcoming statistics. Implementations will be discussed with the mentor, agreed on and written down during this period of time. These will be added as C++ functions and extensive unit testing will be done. At the end of these two weeks the JavaScript program running in the web browser would essentially be able to throw an exception if, using the margin of error functions, it feels that in later stages the server might behave anomalously due to unlikely behaviour of statistics. (for example, a statistic that is greater than 2 standard deviations from the older statistic would result in a warning)

VI. Week 8 (July 7 - July 14)

Integrate the open source libraries gnuplot and/or koolplot into the JavaScript program to be developed, or use Angular with d3 framework for more advanced plots (details to be discussed, and agreed upon later). The functions of these libraries can be easily called in shell for the user to get a plot of statistics retrieved and these data visualization techniques will also be helpful for the Kea developers to get a better understanding of how Kea functions (for example by seeing exactly when traffic increases, when it starts saturating, etc).

Phase II evaluations.

Phase III: Further usage of statistics in error prediction (using Linear Regression)

VII. Week 9 (July 15 - July 22)

Start by importing any necessary libraries into Kea for the regression algorithm. The prerequisites for the model will be discussed with the mentor, agreed upon and written down. Modular pieces of code will be written that will take statistics as input and linearly plot them against time. The input will be taken from statistic-get functions which have been extended to support time-based input.

IX. Week 10, 11 (July 23 - August 6)

Deploy the modular pieces of code written in week 9 into the Kea repository and start their integration through unit testing. Also during these weeks, functions written in week 6, 7 which calculate margins of error will be extended to support the linear regression models as input. Such functions will allow Kea to generate a warning if, using the regression line, a certain statistic that is expected to grow, when extrapolated, shows that it will stop growing in the future.

If time permits, this week will also be used to start a comprehensive discussion on the issue of misplaced statistics raised in ticket #5572.

X. Final Week (August 7 - August 14)

Resolve the issue of misplaced statistics raised in ticket #5572. Write up any missing documentation and complete any remaining code. Work on any feedback received along the way.

Final evaluations

Post GSoC

One of the major things to work upon is the implementation of the solution discussed and agreed upon within the GSoC period for the ticket #5572, if not done earlier. Documentation also needs to be extensively updated. Other work may include any fixes needed for existing functions for the faster development and deployment of Kea 1.50.

How do I fit in?

I have a strong statistical background required for this project. I also have the algorithmic/coding background needed because of the online courses I have taken where writing concise modular code was a priority. I can easily devote 50+ hours per week in order to stick to the proposed timeline and hopefully implement other new features apart from the ones suggested in the timeline.