Revamping Spacecraft Operational Intelligence

Victor Hwang*

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

A critical aspect of any successful flight project is the ability to correctly interpret raw telemetry data in order to monitor the health of the spacecraft and resolve anomalous behavior. The main obstacle comes from organizing massive volumes of data from both the spacecraft and multiple ground data systems into time-discrete trends and patterns.

The EPOXI flight mission has been testing a new commercial system, Splunk, which employs data mining techniques to organize and present spacecraft telemetry data in a high-level manner. By abstracting away data-source specific details, Splunk unifies arbitrary data formats into one uniform system. This not only reduces the time and effort for retrieving relevant data, but it also increases operational visibility by allowing a spacecraft team to correlate data across many different sources. Splunk's scalable architecture coupled with its graphing modules also provide a solid toolset for generating data visualizations and building real-time applications such as browser-based telemetry displays.

EPOXI's experience using the Splunk system has demonstrated that employing modern searching and indexing techniques improve the overall operational intelligence of the team. Splunk also serves as a solid backend for developing new mission operation tools.

Nomenclature

DI Deep Impact
DSN Deep Space Network
EPOXI Extrasolar Planet Observation and Deep Impact Extended Investigation
JPL Jet Propulsion Laboratory
TCM Trajectory Correction Maneuver

I. Introduction

In the seven years that Jet Propulsion Laboratory (JPL) has been operating the Deep Impact (DI) spacecraft, hundreds of gigabytes of information have been generated. These files, which contain spacecraft telemetry, Deep Space Network (DSN) telemetry, prediction data, testing data, scheduling information, and more, continue to reside in the same organizational structure since its initial development in the early 1980s. Since then, the rise of social media and decreasing cost of computation power has accelerated the focus on uncovering trends in Big Data. Though the amount of raw data produced by DI pales in comparison to the data produced by certain social network data centers, EPOXI has still benefitted from the development of their data analysis tools. Specifically, EPOXI flight project has successfully utilized the popular Big Data tool, Splunk, to provide a new way of looking at its data.

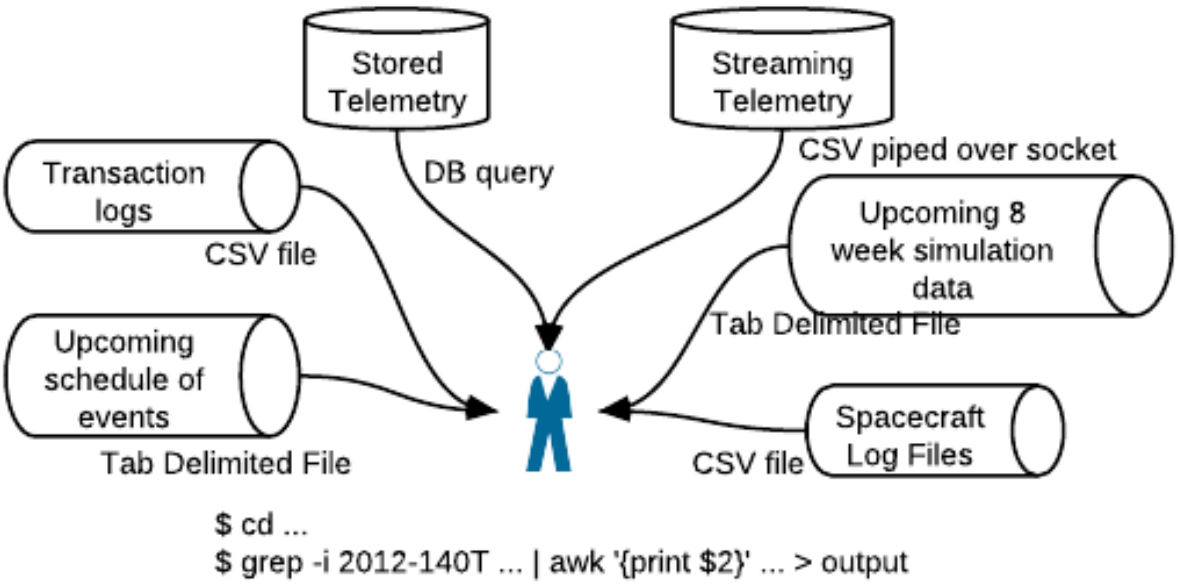
While Splunk is more commonly used for managing large networks of computers, its ability to index arbitrarily formatted timestamped data into events occurring on a timeline makes it ideal for spacecraft operations. Its query language leverages this to search over timespans, rather than managing multiple text files by hand. In addition, Splunk's architectural flexibility allows EPOXI's real-time telemetry stream to be mined and transformed on-the-fly with very little latency. These two features are not new by any means. However, Splunk packages this power into an intuitive web-based interface, which allows the user to begin with a large quantity of telemetry data and systematically define filters until only the desired data is left.

*Staff Engineer, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

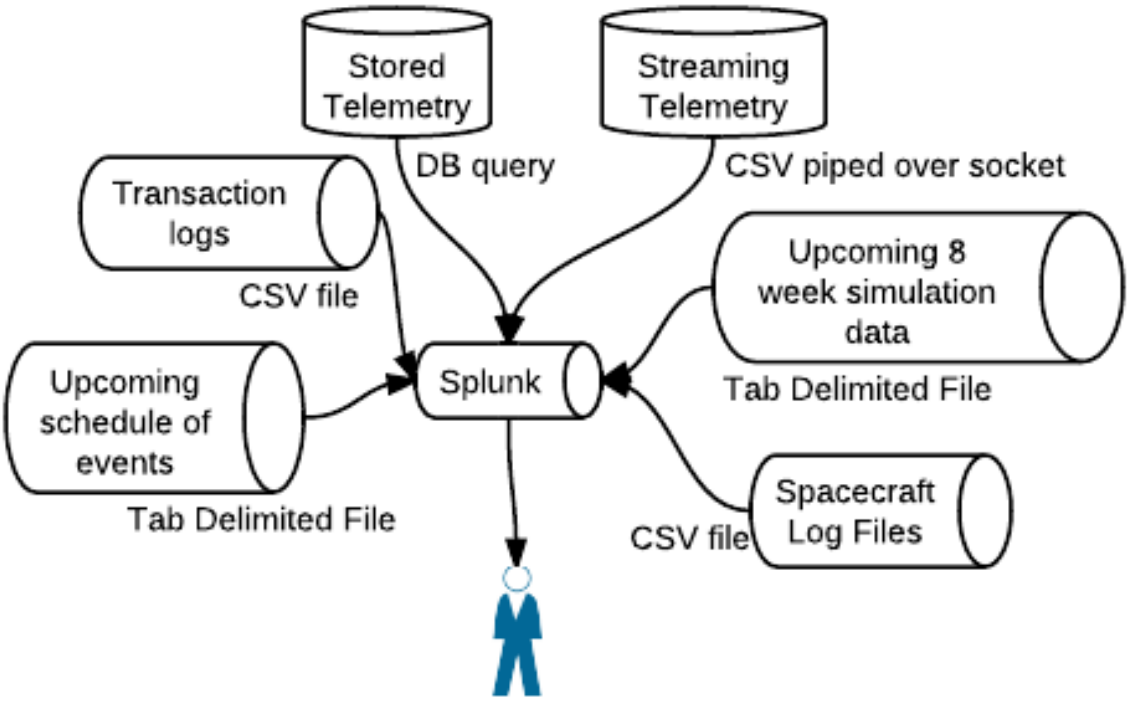
Since implementing Splunk, EPOXI has experimented with real-time browser-based telemetry viewers that allow users to define their own data visualizations, anomaly analysis tools that do not require extensive command line mangling, and advanced reporting and alerting systems. This paper describes how ingesting Deep Impact's data with Splunk resulted in a unified data set and how EPOXI used this to develop a new class of operational tools.

II. Unifying Data

The following image shows a few data types that EPOXI deals with on a frequent basis.



There are various kinds of log files that reside as flat files on various filesystems. The stored telemetry from the spacecraft sits inside databases, while the streaming telemetry come over a socket. While it is possible to access any piece of information from these sources, it requires the user to remember the locations of all this data and have the familiarity to extract data from the feed, whether it be with a scripting language, basic Unix commands, or existing tools. This non-uniformity of data also makes it difficult to view different data together - if one wanted to see the time between when a command was issues on the spacecraft (found in the spacecraft log files) and when it was successfully stored in the ground data system (transaction logs), it would be very cumbersome to interleave the two.



With Splunk, data indexing and access can be abstracted using a few pieces of information: the location of the data, what the data describes, and how the data is formatted. The user defines these parameters and then no longer has to remember file directories, server names, or column headers. Instead, the user retrieves data by specifying a specific category of data over some particular time range. In the figure above, there is no mention of where the data is stored or what format it is in. This has been abstracted away and one can simply filter telemetry data by the attribute `sensor_name` and give a time range specification from "five minutes ago".

III. Defining Data Interfaces

In defining the data interfaces, Splunk decouples two pieces of information: where the data is coming from, and what the data looks like. These are respectively called the **source** and the **sourcetype** and are fundamental to how Splunk organizes data.

A. Defining a Source

Splunk provides modules for interacting with a variety of data sources. It is important to note that defining a data source is completely decoupled from what is contained in the data. This merely establishes the mechanics behind data retrieval.

1. Network Sockets and Forwarders

Splunk can bind to a socket that is streaming UDP or TCP data out. This is useful if another machine has access to data that the indexer does not. With EPOXI, the tool for streaming live telemetry was only compiled for a Solaris machine, and the Splunk indexing machine was running Linux. By piping that data to a TCP socket on the indexer, there was no need to rebuild the tool to work with Linux.

Another alternative to socket binding is to use Splunk's forwarding system. If the data lies on a remote machine other than the indexer, a forwarding program can be installed on remote machine that pipes its data directly to the Splunk instance on the indexer. This alleviates the need to write a script that sends data across a network port.

2. Monitoring files and directories

Many of Deep Impact's logs are single flat files that have additional data appended to them after every DSN track. Splunk monitors these files for changes and indexes new data whenever the file contents change. Expanding this further, Splunk can monitor a directory for new files. This is useful for watching directories that contained downlink transaction logs for every single file downlinked from Deep Impact. Every time a file is downlinked, the ground data system generates a single transaction log for the file and Splunk quickly indexes this.

3. Scripted Inputs

The ability to have Splunk periodically run a script and index its standard output is extremely useful for dealing with particularly unusual data that Splunk is unable to natively parse. In one case, there is a legacy tool that outputs text embedded within tables drawn with ASCII characters.

```
|--------------------------------|
|   2011-100T10:00:00 |  Value 1 |
|--------------------------------|
|   2011-100T12:00:00 |  Value 2 |
|--------------------------------|
```

In this case, it is far easier to write a Python script to generate a simpler, comma-separated output. Splunk then runs this script periodically and indexes the standard output of this Python script.

B. Defining Sourcetypes

In order to parse raw data into something meaningful, the user defines a **sourcetype**. The sourcetype then takes the user definition (generally a regular expression) and generates **events**. The event is essentially Splunk's atomic data type - it consists of an arbitrary set of data that is associated with a timestamp. The following example shows a simple definition of raw channel telemetry:

```
2012-100T00:10:00,temp_1,100
2012-100T00:10:00,temp_2,123
2012-100T00:20:00,temp_1,101
2012-100T00:20:00,temp_2,113
```

The sourcetype definition would consist of the following:

1. The location and time format of the timestamp (here it would be `%Y-%jT%H:%M:%S`)
2. The location and name of the first "field" of data (this would be a regular expression capturing the `temp_1` or `temp_2` string and then naming this field `sensor_name`)
3. The location and name of the second "field" of data (this would be a regular expression capturing the numerical values) and naming this field `digital_value`.

With these definitions, Splunk can index this raw data as four events with two fields of data. This allows the user to search for `temp_1` specific data, or search for `digital_values > 10`. Once the data stream is scaled up to millions of events with thousands of sensor names, this organizational structure becomes very effective.

IV. Architecture Summary

Splunk's distributed architecture allows it to handle very large data streams. Until now, there has been no distinction between where the data gets transformed from the raw data source and where the queries are processed. However, separating these two roles optimizes the process. Splunk defines these two roles as the **indexer** and the **search head**. The indexer takes in the raw data feed and stores it in its own internal

format, while the search head does the more CPU intensive function of parsing the query and retrieving exactly the data that is needed from the indexer.

This setup lends itself to solving one particular security issue that EPOXI ran into. Access to the spacecraft data is confined to one particular network behind a firewall. Because of user access rules, having both the search and indexing located on one machine would have required all Splunk users to have access to this particular network, which was not an option.

Instead, the structure was split into one indexer and one search head. The indexer, which held sensitive spacecraft data, remained behind the firewall in order to follow security regulations. The search head machine was placed outside of this network, and a firewall exception was made between these two machines. Because the search head only caches the sensitive data long enough to return a query result, there was no breach of security regulations and users did not need access to the limited network.

V. Data Searches and Visualizations

Once the interfaces to data have been properly defined, searching for anomalous telemetry data becomes an exercise in writing queries that filter out extraneous data. Below are a few example of the Unix-style search language that Splunk employs.

`sensor_name=temp_sensor_1`	query temp_sensor_1 data
`sensor_name=temp_sensor_*`	query all temp_sensor data
`sensor_name=temp_sensor_* temp>10`	query all temp_sensor data greater than 10

`sensor_name=temp_sensor_*`
`earliest=10/19/2009:0:0:0`
`latest=10/27/2009:0:0:0`
	query all temp_sensor data between the time range

Table 1.

For the EPOXI project, any of the previous queries would still return too much data to work with. The following queries show how one might begin filtering out irrelevant data with Unix-style pipes.

`sensor_name=temp_sensor_1 | sort temp | head 5`	returns 5 lowest temps
`sensor_name=temp_sensor_1 | sort -temp | head 5`	returns 5 highest temps

`sensor_name=temp_sensor_1 |`
`delta temp as chng_temp |`
`where chng_temp > 5`
	returns events where the temp changes by 5

Splunk's search language provides hundreds of functions for manipulating data. While the queries can occasionally get complex and lengthy, the fast performance of the queries makes trial-and-error a painless task.

A. Deep Space Network Performance Analysis

Just as Unix command line tools have their limits, Splunk's search language falls short for particularly complex analyses. In these cases, Splunk's external API allows procedural languages to make data queries. This section describes an analysis of the Deep Space Network (DSN) performance using Python and Splunk to manipulate the data.

There are many aspects of the DSN process that influence how the spacecraft team plans their activities. One in particular is the time it takes the DSN to lock on the spacecraft's telemetry. Because time on the DSN is so valuable, this wait time is expected to be extremely consistent. Otherwise, time is wasted during the track waiting for the DSN to establish a solid connection with the spacecraft.

With Splunk, EPOXI was able to aggregate the past few years of DSN data to generate performance statistics for each DSN station. The table below shows what performance statistics were generated (with altered data values for export).

DSN Station	Avg Predicted Tlm Lock Time	Avg Actual Tlm Lock Time	Std Dev in Lock Time
11	200 sec	202 sec	8
12	200 sec	209 sec	5
22	200 sec	204 sec	4
34	200 sec	201 sec	2

DSN Station	Avg Dropped Tlm Lock per Track	# Times No Telemetry Lock At All
11	1.2	5
12	1.11	3
22	1.4	4
34	1.3	8

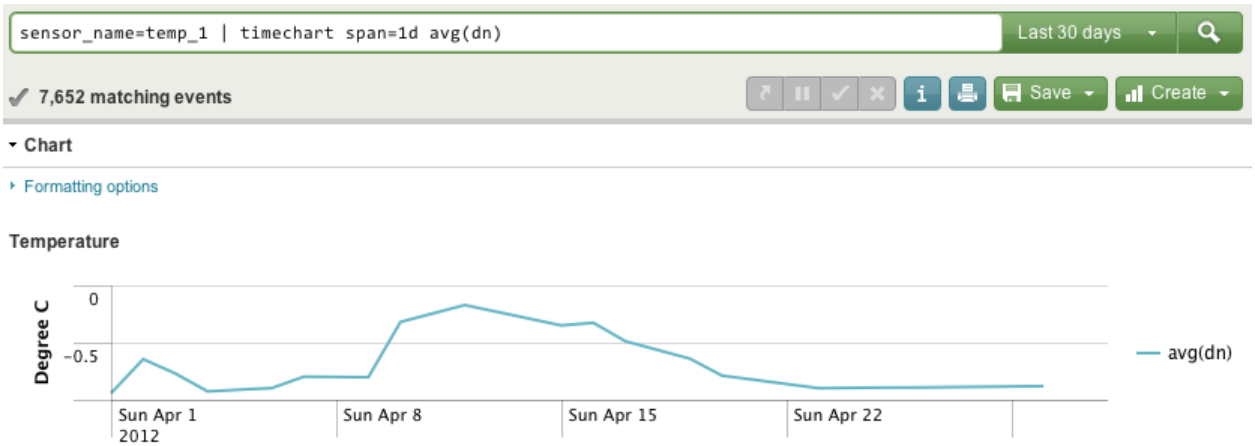
The source code to generate these statistics did not exceed 100 lines. First, a Splunk query was written that would return data from both the telemetry sourcetype and the prediction sourcetype. Then, using transactions[a], the relevent events were grouped together based on a grouping definition. Specifically, the boundaries of the DSN transaction object were defined by a "beginning of track" event and an "end of track" event. Everything in between was grouped into the returned data object. Once Python ran this query, the returned object contained the relevant time events for every single DSN track, which looked similar to the following:

```
{
  dsn_station: 11,
  beginning_of_track: 2011-100T01:00:00,
  predicted_lock_time: 2011-100T01:30:00,
  tlm_lock_times: [ 2011-100T01:33:00, 2011-100T04:00:00, 2011T05:00:00]
}
```

The majority of development time was spent building a Splunk query to group these events properly. At first, the transactions returned would miss the end of a track event and group together multiple predicted lock times and telemetry lock times together. By adding more aggressive filtering, the transactions were pruned and it became trivial to calculate the relevant time statistics with Python.

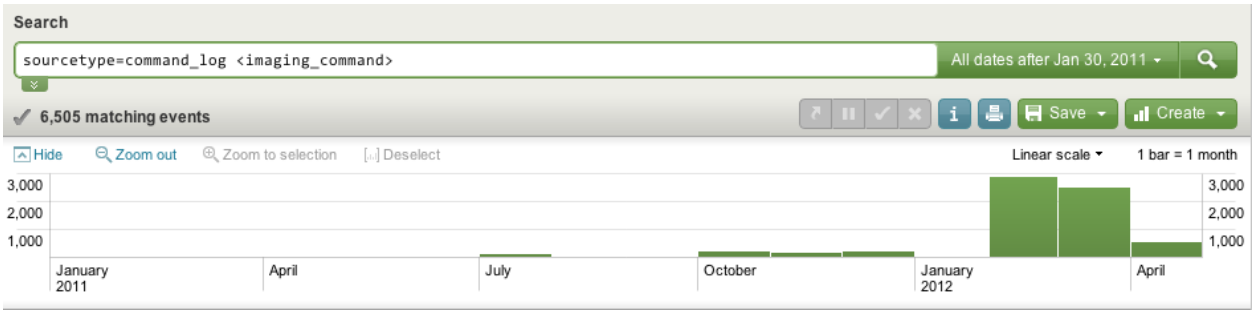
B. Visualizing Data

Splunk's web interface has useful data visualization modules that build on top of the search language. Tables can be built with one-line commands and graphs can have many signal filters applied to them. The following screenshot shows the command to apply a rolling 1-day average filter over a set of data.



[a]More about Splunk's `transaction` command can be found at http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/transaction

Splunk also has quickviews to show event counts over time, which are helpful for showing a summary of activities. For example,the following image shows a histogram of the number of images taken since January of 2011, which quickly shows when the EPOXI project was actively observing.



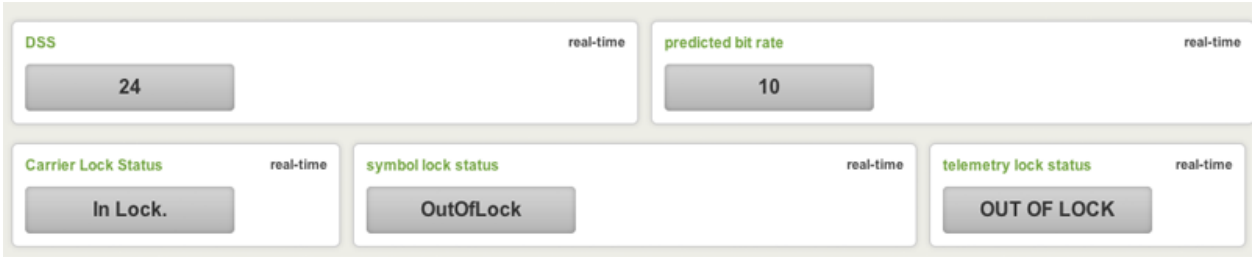
One particularly useful feature in this quickview is the ability to zoom in on a particular time range. This allows for an interactive environment where navigating through data only requires mouse clicks. For instance, one could zoom in on the activity period during January and April, which would establish a new time range. Then the query could be changed to `sourcetype=downlink_transactions` to see exactly which image files from that period were downlinked.

VI. Real-Time Searches

Splunk provides a real-time search system that already exists within the search language. By specifying a time range with the `rt` keywoard, Splunk begins running a continuous search. For example, the query `sourcetype=temp_1 earliest=rt-5` informs Splunk to continuously search for data that has been received within the past five minutes. Combining this feature with the data visualization modules gives the ability to build real-time telemetry displays that are web-based, user-customizable, and very responsive.

A. Real-Time Telemetry Displays

EPOXI's existing telemetry display system is built on a text-display system that can be difficult to gather information from. It is difficult to customize and can only be accessed through window forwarding systems. With Splunk already attached to the data streams, it was very simple to build dashboard views that mimicked the existing system. The following image shows a simple DSN status page that is the result of aggregating multiple data channels.



Each of the single value boxes continuously updates the value inside with the most recent data from its corresponding channel.

This next image shows a page that lists completed downlinks from the spacecraft in real-time. The ground data system stores an XML file that contains details about the transactions. Most of the data is not pertinent, so this display shows the filename, downlink time, and the completion status (which is extracted from the XML file). The downlink time is calculated on the fly by using the previously mentioned `delta` command.

	_time	Filename	FileState	DLTime
1	4/26/12 3:53:27.000 AM	IINHIUBXF_4000127_017_017.bin-DI_FBB-2A14-2012-117T03.53.25.dtl	COMPLETE	121
2	4/26/12 3:51:26.000 AM	IINHIUBXF_4000127_016_017.bin-DI_FBB-2A13-2012-117T03.51.24.dtl	COMPLETE	107
3	4/26/12 3:49:39.000 AM	IINHIUBXF_4000127_015_017.bin-DI_FBB-2A12-2012-117T03.49.37.dtl	COMPLETE	118
4	4/26/12 3:47:41.000 AM	IINHIUBXF_4000127_014_017.bin-DI_FBB-2A11-2012-117T03.47.38.dtl	COMPLETE	117
5	4/26/12 3:45:44.000 AM	IINHIUBXF_4000127_013_017.bin-DI_FBB-2A10-2012-117T03.45.41.dtl	COMPLETE	117
6	4/26/12 3:43:47.000 AM	IINHIUBXF_4000127_012_017.bin-DI_FBB-2A0F-2012-117T03.43.44.dtl	COMPLETE	112
7	4/26/12 3:41:55.000 AM	IINHIUBXF_4000127_011_017.bin-DI_FBB-2A0E-2012-117T03.41.53.dtl	COMPLETE	114
8	4/26/12 3:40:01.000 AM	IINHIUBXF_4000127_010_017.bin-DI_FBB-2A0D-2012-117T03.39.58.dtl	COMPLETE	113
9	4/26/12 3:38:08.000 AM	IINHIUBXF_4000127_009_017.bin-DI_FBB-2A0C-2012-117T03.38.05.dtl	COMPLETE	123
10	4/26/12 3:36:05.000 AM	IINHIUBXF_4000127_008_017.bin-DI_FBB-2A0B-2012-117T03.36.03.dtl	COMPLETE	305
11	4/26/12 3:31:00.000 AM	IINHIUBXF_4000127_007_017.bin-DI_FBB-2A09-2012-117T03.30.57.dtl	COMPLETE	110
12	4/26/12 3:29:10.000 AM	IINHIUBXF_4000127_006_017.bin-DI_FBB-2A08-2012-117T03.29.07.dtl	COMPLETE	117

While these dashboards do not show novel information, both of these dashboards were built in under ten minutes using the simple searches described in Table 1. No programming (aside from the queries) was required.

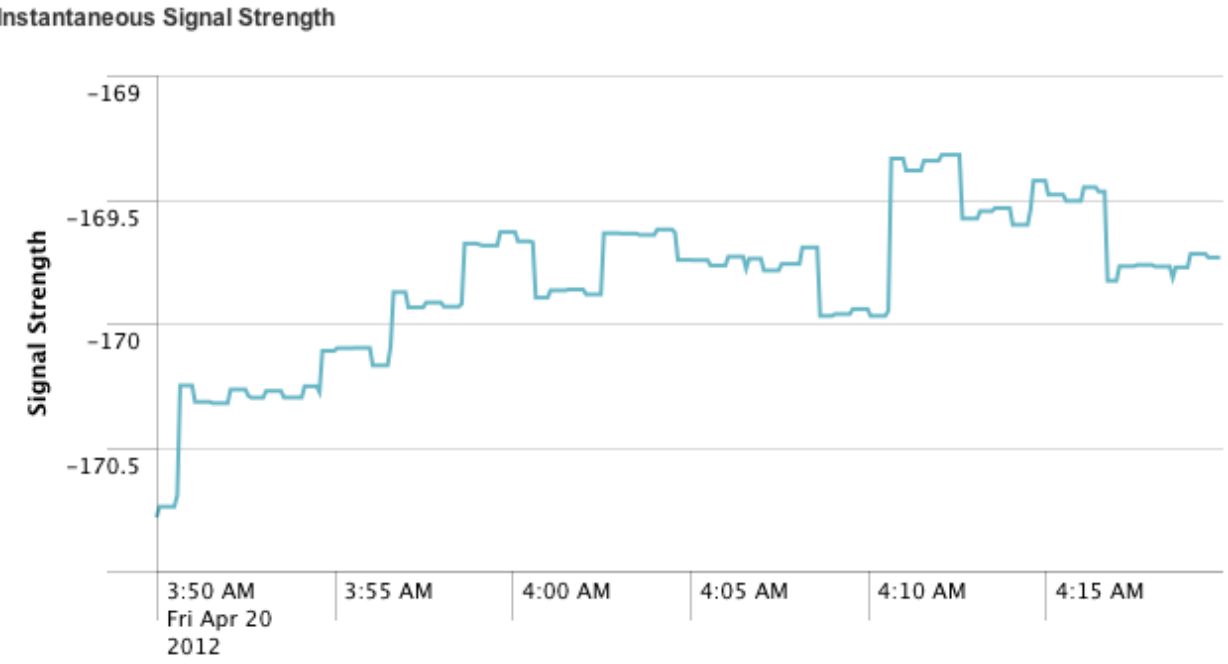
B. Safing Event April 2012

In April of 2012, the Deep Impact spacecraft went into safing. While the technical details of safing are not described here, the importance of data visualization became very apparent during the recovery of the spacecraft.

Due to the geometry of the spacecraft and its sequenced safing behavior, solid downlink and uplink connections were only available at periodic intervals. Despite calculating signal strength predicts, the motion of the spacecraft was extremely inconsistent. As a result, the decision to radiate commands was based solely on real-time signal strength data.

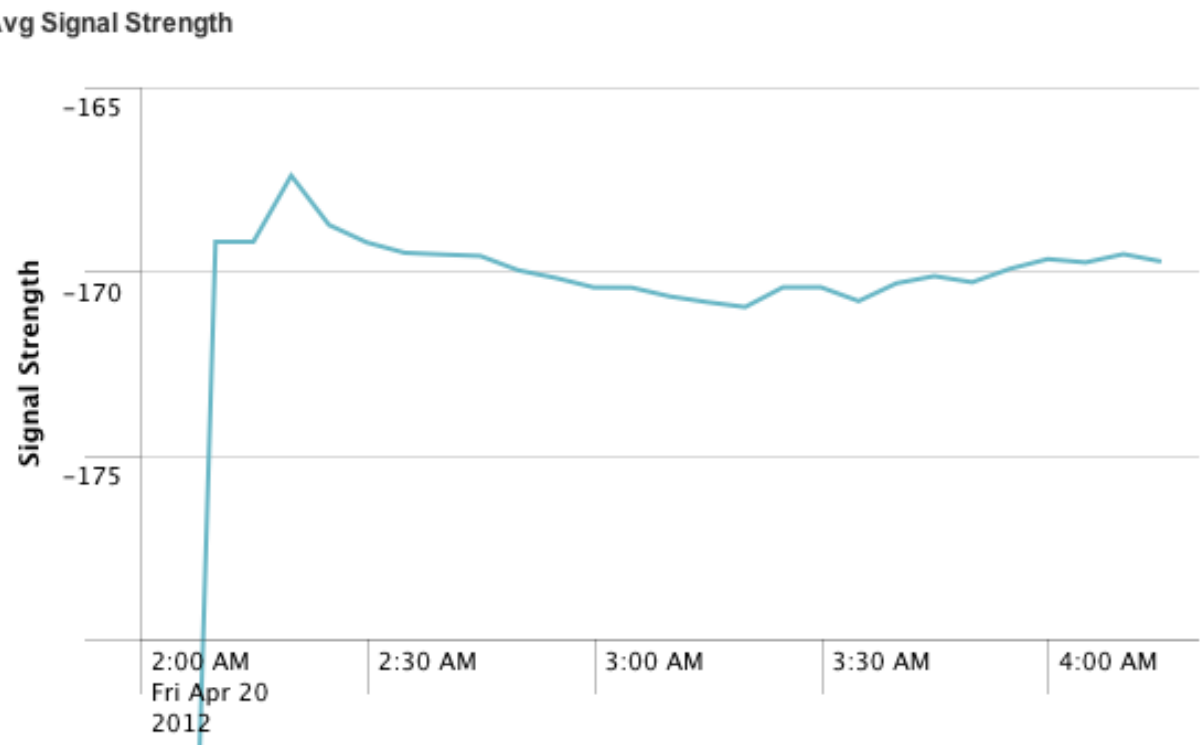
The below image shows the real-time instantaneous signal strength over the last 40 minutes, which was the initial graph built before the safing event. The query tells Splunk to plot the data by quantizing the data into 5 second buckets (which was the subscription rate of the data) and taking the first value. By itself, this visualization was not helpful because of the noise of the signal and the short time range.

```
sourcetype=DSN_telemetry channel_name=signal_strength earliest=-40m | timechart
span=5s first(digital_value)
```



A simple change in the query syntax created a graph of far more value. The syntax now quantizes the data from the past 2 hours into 10 minute buckets and takes the median value. This allowed us to pick out the periodic trend much more easily and to have a better sense of when to radiate commands.

```
sourcetype=DSN_telemetry channel_name=signal_strength earliest=-2h | timechart
span=10m median(digital_value)
```



VII. Future Implementations

EPOXI still has yet to take full advantage of Splunk's technology. The following section describes theoretically feasible ideas that have yet to be implemented.

A. Predicted vs. Actual Data Analysis

Splunk has no issues indexing prediction data from the future. One idea considered for real-time telemetry displays was to have plots that back-fill predicted data into the real-time graphs. Combining this with alarm limits on channel values could provide a useful interface when monitoring a critical spacecraft event. For example, during a trajectory correction maneuver (TCM), the predicted attitude rates could be plotted for the entire duration of the maneuver. The real-time telemetry data could then be plotted against the same plot, making any major discrepancies instantly visible.

This could be further expanded upon for post-DSN tracks to analyze telemetry data for anomalies. For example, if any channel deviates from its prediction for greater than some set amount of time, an alert could be thrown. Also, it could be used to constantly monitor the accuracy of the prediction models every time data is received.

B. Alert System

Splunk has an alerting system that hooks into real-time searches. One can set alerts if the result of a user-defined query returns a particular result. For example, one useful alert would have Splunk send a text message to the spacecraft team if there was no telemetry lock within the first hour of the start of a DSN track (or alternatively, if no spacecraft data is received within the first hour of the DSN track).

This could be combined with the feature described in the previous section to send alerts if a particular telemetry value begins deviating from the prediction files. This would be less relevant to the EPOXI project due to the infrequency of DSN tracks. However, this could be very useful to a project that had constant visibility to the spacecraft.

VIII. Conclusion

The integration of Splunk with EPOXI's data has helped unify the growing data set from the Deep Impact spacecraft. Initially, EPOXI's numerous data sources were stored in a variety of databases, flat files and directories. The differing formats of the data was an obstacle whenever there was a need to extract information - one would have to remember the location of the data and the corresponding tool needed to unpack the data. Splunk provides a data access layer that abstracts away these low-level details so that anomaly analysis can be done without battling these obstacles. Its built-in search language also makes correlating data across many different sources (such as prediction data and spacecraft event logs) simple. Splunk's distributed architecture also provides a responsive backend that supports highly customizable telemetry display systems, which has already assisted the team in making operational decisions in real-time.

Splunk's ability to turn Deep Impact's unstructured data into a complete picture of spacecraft's state has the potential to revamp numerous existing tools at JPL. As the collection of data continues to grow, Splunk proves that modern searching and indexing tools will only become more valuable as time goes on.

IX. Acknowledgements

The author would like to extend their sincerest thanks to all of the EPOXI team: Steve Wissler (JPL), Tim Larson (JPL), Greg La Borde (JPL), Rich Rieber (JPL), Terry Himes (JPL), Bob Wing (JPL), Yee Lee (JPL), Jeff Pinner (Twitter) and Hal Uffelman (JPL).

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration