Technical Paper

# Mean Time Between Failure in SAS IT Resource Management


sas
THE POWER TO KNOW®

**Release Information** Content Version: 1.0 May 2013.

# Contents

# Introduction

There are many ways to analyze the reliability of IT resources in a given environment including availability percent, queues waiting for free resources, and up/down times. "Mean Time Between Failure" (MTBF) is a metric which originated in hardware engineering circles, and can be used in the broader IT universe to measure whether IT resources are reliably available.

This paper discusses the implementation of Meat Time Between Failure using SAS IT Resource Management. Before trying to calculate an MTBF metric for resources there are decisions to be made and sometimes data issues to overcome. This paper addresses those issues to give you enough background to start thinking about MTBF at your site.

# How to measure success and failure

The answer to "What is a failure?" may be obvious in some cases. For a router we have availability data from SNMP. A router that is unavailable represents a failure.

But the vagaries of data collected from various data sources or adapters often take a little more thought. For measuring MTBF you have to make decisions on how to handle data including the topics of artificially bounded or censored data, missing data, determining what time span of data is appropriate to analyze and what granularity of detail is sufficient.

You may also have to combine data from multiple data sources. If you are estimating MTBF for one type of resource based on one supplied analytic column then the analysis is simpler. In other cases you may want to combine various low-level resources to represent MTBF for a higher-level construct. For example, for a web site to be truly available to the public, the physical server must be up, the server processes and web services must be running, and the network must make the services available to the target audience.

You may want to accommodate expected or scheduled outages in your MTBF calculations and in that case the unavailability during the outage should not count in the same way as failures. For example, if your site had announced a 4-hour outage for upgrading equipment, you may decide that this outage should not count when calculating an MTBF for the same equipment. If there are many of these outages to track and account for, you may need to have a separate outage table.

# Preliminary Checklist

- Data Sources
    - Is availability or failure data already available in the correct form?
    - Is availability or failure data in appropriate granularity (detail) for my purpose?
    - Is the time span represented by the data long enough?
    - Do I have a single existing column that can represent availability, or success versus failure?
    - Do I have to combine metrics from multiple tables?
    - Do I have to recalculate, rescale, or derive new values from existing columns?
- Missing values and other problems
    - Are there observations with missing values? What assumption should be made about them?

    - Are there missing observations representing completely missing time spans? What assumption should be made about them?
    - At the beginning and end of the time span for any resource, there is fragmentary data. What should be done about these fragments?
- Outages, expected and otherwise
    - Should planned outages be treated differently than unexpected failures?
    - Do I need a separate table of outage information to be able to represent these?

## SNMP Interface Example

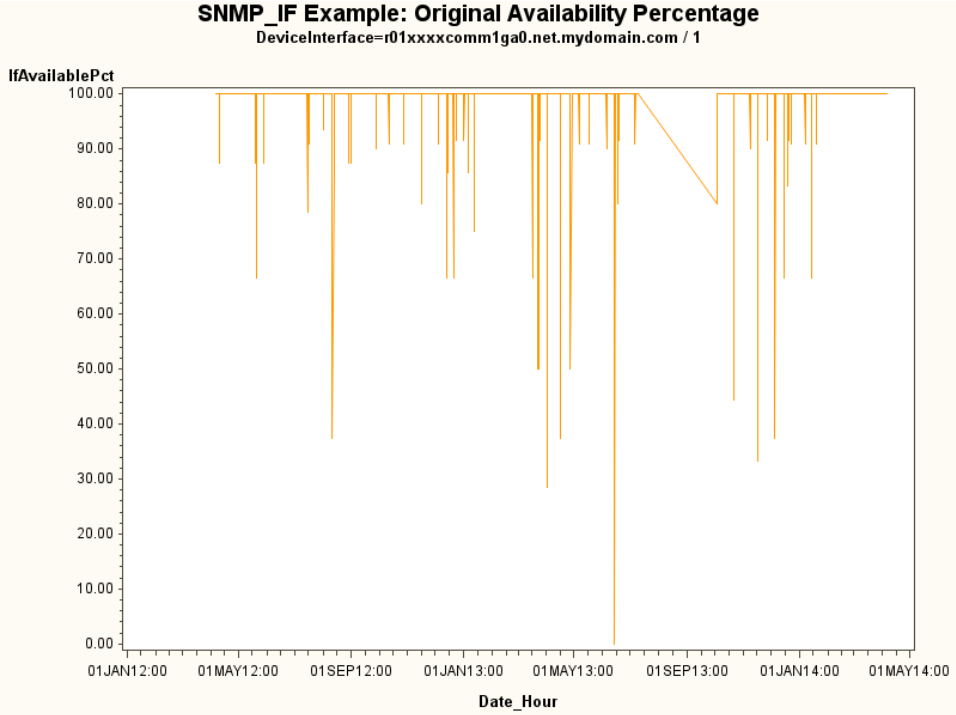
The following example is a fairly simple one to illustrate some potential analytic pitfalls, using the IF interface data in SNMP. In SAS IT Resource Management, this is the SNMP_IF staged table, for which we supply a number of template aggregation summary tables.

In SNMP_IF, up/down status is already being reported as such. For a router, we can take the status information reported (IfStatusChangeToAvails, IfStatusChangeToUnavails, ifAvailableTime) as a definition of whether the router is up or down. However, staged tables are rarely good sources of data to analyze MTBF directly, because they typically don't cover a very long time span.

The supplied aggregation table KeyMetricsIFInterface is a convenient longer-term summary of the SNMP_IF data, which includes a sum for IfAvailableTime and a computed column that translates that into a percentage, IfAvailablePct. IfAvailablePct is calculated using the sum of duration, so that only time periods for which we have data are being used. The default age limit is currently 730 days so it's long enough for meaningful MTBF calculations.)

The KeyMetricsIFInterface table is an accumulated hourly summary of SNMP_IF data, it's probably granular enough for our purposes. However, in other circumstances you may want more granular data, or you may have to use less granular data if you cannot afford to store the more granular data for a long enough time period.

We can plot the availability percentage (ifAvailablePct) over time as a starting point:




Note that the availability percentage rarely reaches zero. That is because this summary table (KeyMetricsIFInterface) is summarized to the hourly level (DayDateHour); only one period of unavailability lasted as long as an hour in this sample data.

## Calculating MTBF from Availability or Failure Rates

Availability, failure rates, and MTBF are all ways to describe the reliability of resources. It is possible, although not recommended, to directly derive MTBF from failure rates by calculating MTBF as the total time space divided by the number of failures observed. This is a rough estimate and not ideal, because it does not take into account the amount of time lost due to failure (only the number of failures). Calculated in this way, the MTBF would be the same for a 24-hour period with two separate 15-second failures as it would be for a 24-hour period with two 6-hour failures.

Estimating MTBF directly as a calculation from availability or failure rates has other potential problems, such as dealing with missing values or partially known time segments, which will be covered in more detail below.

## Translating percentages to up/down periods

If we want to translate an availability percentage to up/down periods – which we'll need to do to calculate MTBF -- then we will want to apportion the hours that have some partial unavailability (between 0 and 100 exclusive). In the following example code, we simply assign the available part versus the unavailable part based on the percentage.

We also presume that there are not multiple failures occurring for the same device within the same summarized hour – if there were, we could not tell at this level of summarization. If this assumption is inappropriate in your circumstances, you may need to find another more granular source of data.

We can add a User-Written Code transformation using the KeyMetricsIFInterface summary table as input, via the IT Resource Management client. This code fragment will create new columns representing the up/down status of the resource:

```
proc sort data=[libref.tablename] out=work.sorted;
   by deviceInterface daydatehour;
data [libref.cumulate];
   set work.sorted;
   by deviceInterface;
   retain uptime downtime 0;

   if first.deviceInterface then do;
      uptime=0;
      downtime=0;
      end;

   /* flag to reset uptime after down occurs during period */
   reset = 0;

   /* If we don't know the availability, do not assume up or down */
   if ifAvailablePct<.z then delete;

   else if ifAvailablePct=100 then do;
      downtime=0;
      uptime=sum(uptime,durationSum);
      end;
   else if ifAvailablePct=0 then do;
      uptime=0;
      downtime=sum(downtime,durationSum);
      end;

   else do;
      /* apportion by presuming downtime is at the end of the current duration */
      uptime=sum(uptime,(durationSum*(ifAvailablePct/100)));
      downtime=sum(downtime,(durationSum*((100-ifAvailablePct)/100)));
      reset=1;  /* reset uptime now that down has occurred */
      end;

   output;

   if reset then uptime=0;
   run;
```

This code calculates a running total for uptime and downtime during the periods that the resource was available or not available, respectively.

Note that in this code fragment we have two assumptions about missing data. One is that if an observation has a missing availability percentage, we do not use it. This is a common way of treating this condition since we do not know the true value. You may decide to be either more pessimistic (by treating this missing percentage as a down time) or optimistic (by treating it as an up time).

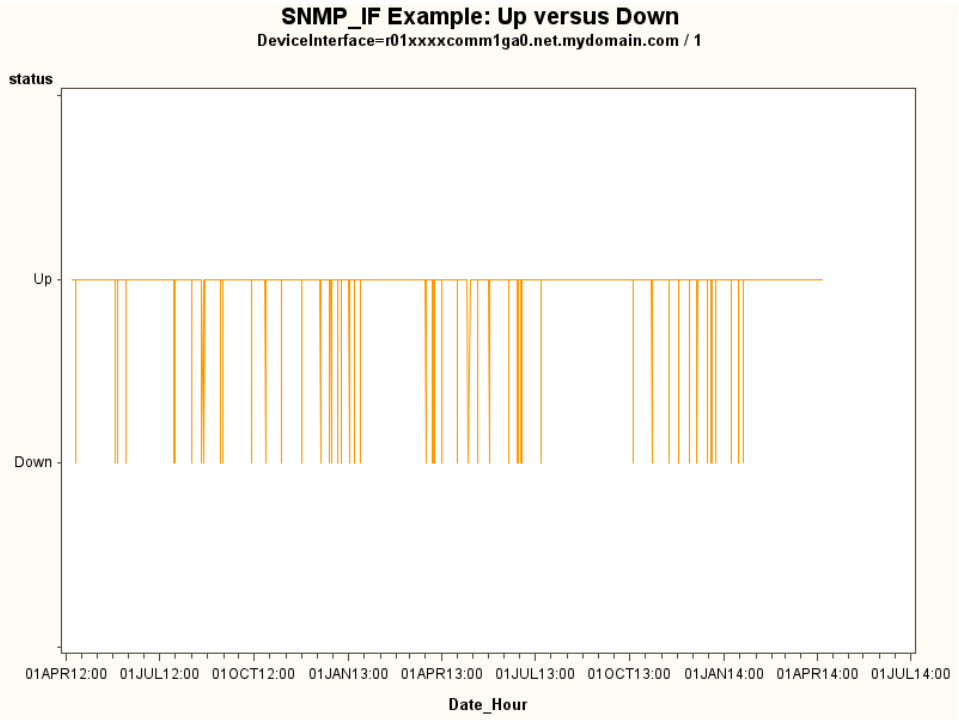
Likewise, the example uses the sum of duration to apportion up and down times. By doing so, we are considering only time periods where we know what happened. For example, if for a given hour there are only 3000 seconds where the status is known, the sum of duration will be 3000, so that the up and down times will also sum to fifty minutes (3000) and not an hour (3600).

In some circumstances, you may decide to either be more or less strict than this. You could attempt to estimate the passage of time periods not represented by any existing observations and declare that either an up or a down period, depending on your situation.

For plotting purposes, we can do a further reworking of the data to be able to plot the binary state over time. (Note that this is only done for this type of plot and not for further numeric analysis.)

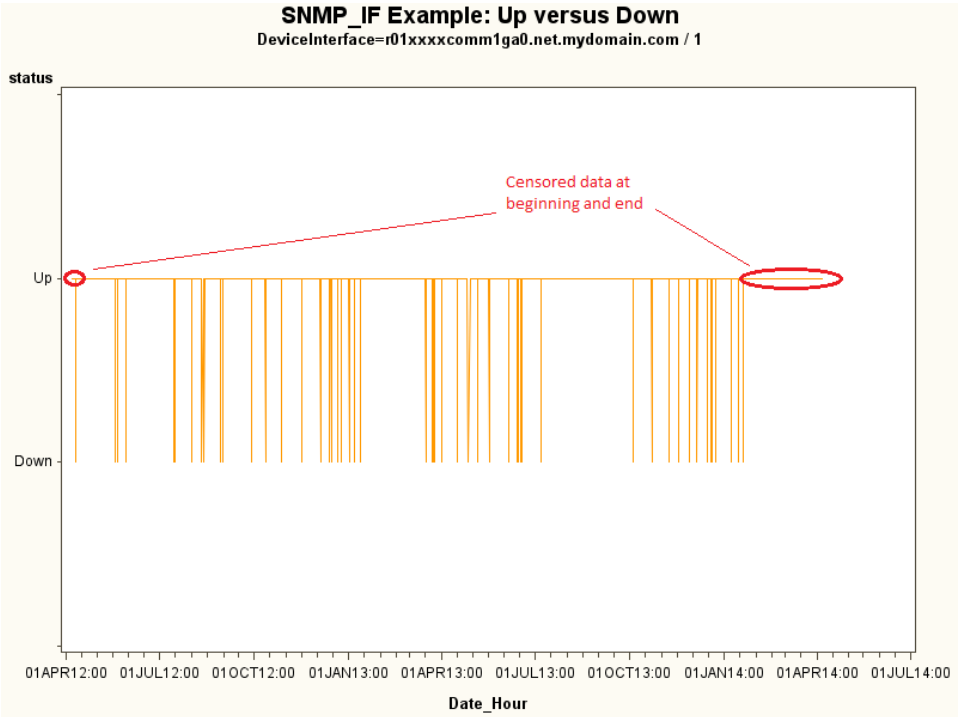
```
proc format;
   value updown
      0="Down"
      1="Up"
      other=" ";

data work.updown;
   set [libref.cumulate];
   length status 3;
   format status updown.;
   if uptime>0 then do;
      status=1;
      output;
      end;
   if downtime>0 then do;
      status=0;
      output;
      end;
   run;
```

If we were to plot this transformed version of the up/down status, the translated up/down periods would then look like this:




## Optionally omit fragments from calculations

There is another potential data problem illustrated here.  At the beginning of the time series, and at the end of the time series, there is only a fragment known (a problem known as censoring).




If we were to take the average MTBF of this data without noticing the censoring, the values at the beginning and end of the time series (particularly the beginning, in this particular case) would bias the MTBF to be calculated as being less than it really is. This is because we know that the period of availability for both the start and end of the sequence is at least this long but almost certainly longer – but we do not know how much longer.

To be more accurate, we could omit any up sequence for which we do not know precisely the start or end of the up sequence – in this case, omitting both the beginning and ending up sequences. If this approach leaves us with no eligible up periods, it is more correct to say that we do not know than to present a spuriously low MTBF.

This, in turn, leads to another interesting special case: what if there were no failures during the time period being analyzed? We cannot know the MTBF for the resource if there was no “F”, or failure. The best that we could say is that the MTBF was “at least” that time period. This is another reason that MTBF analysis should take place over a reasonably long time span – but what is reasonable depends on your knowledge and expectations about the type of resource and your environment.

Taking both of these potential data problems into account is more complex. The sample code fragment from above needs to be revised to omit left- and right-censored time series, which if done correctly will also remove the “no-failures” case. First, we revise the previous code to add sequence information and to write a separate list of observations for possible censoring problems.

We also prorate the fractional availability periods so that we have the lowest number of state changes. We do this by assuming that the hour with partial unavailability is divided into a part that matches the previous state, and a part that does not.

```
proc sort data=[libref.tablename] out=work.sorted;
   by deviceInterface daydatehour;
   run;
data work.uptime(drop=beginUp endUp)
     work.censor(keep=deviceInterface sequence beginUp endUp);
   set work.sorted;
   by deviceInterface;
   retain laststatus .;
   retain uptime downtime 0;
   retain sequence 0;

   if first.deviceInterface then do;
      uptime=0;
      downtime=0;
      sequence=0;
      if ifAvailablePct>0 then beginUp=1;
      else beginUp=0;
      endUp=0;
      laststatus=.;
      end;



  /* If we don't know, do not presume which */
  if ifAvailablePct<.z then delete;

  if ifAvailablePct=100 then do;
     downtime=0;
     uptime=sum(uptime,durationSum);
     status=1;    /* up */
     if laststatus ne status then sequence=sequence+1;
     output work.uptime;
    end;
  else if ifAvailablePct=0 then do;
     uptime=0;
     downtime=sum(downtime,durationSum);
     status=0;    /* down */
     if laststatus ne status then sequence=sequence+1;
     output work.uptime;
    end;



  else do;
     /* apportion by presuming continuation of previous status */
     if laststatus=1 then do;
        status=1;
        downtime=0;
        uptime=sum(uptime,(durationSum*(ifAvailablePct/100)));
        output work.uptime;
        sequence=sequence+1;
        status=0;
        uptime=0;
        downtime=sum(downtime,(durationSum*((100-ifAvailablePct)/100)));
        output work.uptime;
        end;
```

```
    else do;
       status=0;
       uptime=0;
       downtime=sum(downtime,(durationSum*((100-ifAvailablePct)/100)));
       output work.uptime;
       sequence=sequence+1;
       status=1;
       downtime=0;
       uptime=sum(uptime,(durationSum*(ifAvailablePct/100)));
       output work.uptime;
       end;
    end;

 if first.deviceInterface or last.deviceInterface then do;
    if sequence>1 then beginUp=0;
    if last.deviceInterface and uptime>0 then endUp=1;
       else endUp=0;
    output work.censor;
    end;

 laststatus=status;

 run;
```

And then we re-merge the up/down sequences with the possible censoring occurrences to omit beginning and ending sequences that are artificially truncated or censored. At the same time, we can just keep the final observation for each sequence, since that represents the total uptime (for up sequences) and downtime (for down sequences). (Note that this code does that separately in case you want to use the interim form for other analytical purposes.)

```
/* Merge with the censoring information and only keep the final
   observation for each sequence of up or down */
data work.uptime;
   merge work.uptime
      work.censor;
   by deviceInterface sequence;
   if beginUp or endUp then delete;
   if last.sequence;
   run;
```

## Finally the MTBF is calculable

Following all of that data manipulation, the data is now in a form where calculating the MTBF for each device interface is very straightforward. A PROC SUMMARY or MEANS should be sufficient for calculating the MTBF for each device interface over the entire time span of data.

```
 proc summary data=work.uptime;
    by deviceInterface;
    var uptime;
    output out=work.sums mean=MTBF max=Maximum n=Segments;
    run;
 proc print data=work.sums;
    title 'SNMP_IF Example:  Final MTBF';
    footnote "In Hours, Minutes, Seconds";
    id deviceInterface;
    var MTBF Maximum Segments;
    format MTBF time11.2;
    format Maximum time11.2;
 run;
```

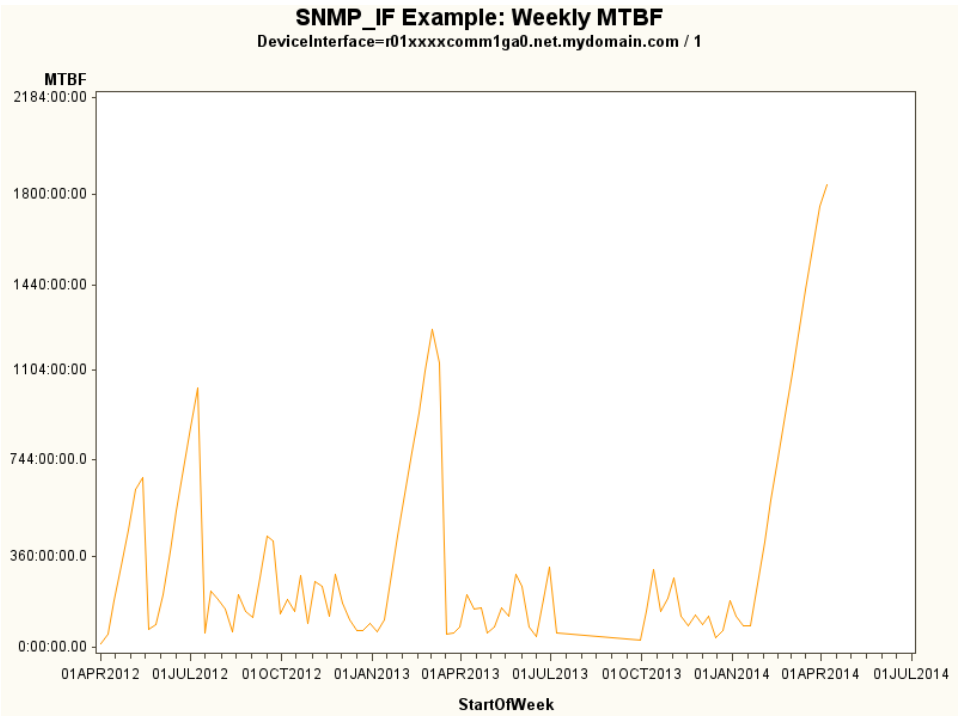
This will give us the overall mean (and maximum) time between failures by device interface, together with the number of segments being analyzed:

SNMP_IF Example: Final MTBF

| deviceInterface | MTBF | Maximum | Segments |
|---|---|---|---|
| r01xxxxcomm1ga0.net.mydomain.com \ 1 | 106:49:58.6 | 1475:26:00 | 121 |

In Hours, Minutes, Seconds

Another common technique is to take the mean separately at a larger granularity (weeks, for example), and plot the weekly MTBF curve.  These data points can also be used for forecasting.




As in several of the previous examples, the units for MTBF are in hours, minutes, and seconds.

## For further information

Other statistical measures closely related to MTBF include:

- Availability
- Mean time to failure (MTTF)
- Mean time to repair (MTTR)
- Mean time between system incidents (MTBSI)
- Failure rate

## References or Resources


Reliability and availability basics
(http://www.eventhelix.com/RealtimeMantra/FaultHandling/reliability_availability_basics.htm)

Yaple, James. "ITIL Availability Management: Beyond the Framework: Part 1", CMG 2006.
(http://www.cmg.org/publications/measureit/2006-2/mit33/measureit-issue-4-07-itil-availability-management-beyond-the-framework-by-james-yaple/)

Heger, Dominique A. and Phil Carinhas. "A Cohesive Framework to Quantify Computer Systems Assurance", CMG 2006.
(http://www.*fortuitous.com/docs/whitepapers/CSA_Methodology_CMG.pdf)*