BGP Route Flap Damping

Status of this Memo

Copyright Notice

Abstract

   A usage of the BGP routing protocol is described which is capable of
   reducing the routing traffic passed on to routing peers and therefore
   the load on these peers without adversely affecting route convergence
   time for relatively stable routes. This technique has been
   implemented in commercial products supporting BGP. The technique is
   also applicable to IDRP.

   The overall goals are:

   o to provide a mechanism capable of reducing router processing load
      caused by instability

   o in doing so prevent sustained routing oscillations

   o to do so without sacrificing route convergence time for generally
      well behaved routes.

   This must be accomplished keeping other goals of BGP in mind:

   o pack changes into a small number of updates

   o preserve consistent routing

o   minimal addition space and computational overhead

An excessive rate of update to the advertised reachability of a
subset of Internet prefixes has been widespread in the Internet.
This observation was made in the early 1990s by many people involved
in Internet operations and remains the case. These excessive updates
are not necessarily periodic so route oscillation would be a
misleading term. The informal term used to describe this effect is
"route flap". The techniques described here are now widely deployed
and are commonly referred to as "route flap damping".

1 Overview

To maintain scalability of a routed internet, it is necessary to
reduce the amount of change in routing state propagated by BGP in
order to limit processing requirements. The primary contributors of
processing load resulting from BGP updates are the BGP decision
process and adding and removing forwarding entries.

Consider the following example. A widely deployed BGP implementation
may tend to fail due to high routing update volume. For example, it
may be unable to maintain it's BGP or IGP sessions if sufficiently
loaded. The failure of one router can further contribute to the load
on other routers. This additional load may cause failures in other
instances of the same implementation or other implementations with a
similar weakness. In the worst case, a stable oscillation could
result. Such worse cases have already been observed in practice.

A BGP implementation must be prepared for a large volume of routing
traffic. A BGP implementation cannot rely upon the sender to
sufficiently shield it from route instabilities. The guidelines here
are designed to prevent sustained oscillations, but do not eliminate
the need for robust and efficient implementations. The mechanisms
described here allow routing instability to be contained at an AS
border router bordering the instability.

Even where BGP implementations are highly robust, the performance of
the routing process is limited. Limiting the propagation of
unnecessary change then becomes an issue of maintaining reasonable
route change convergence time as a routing topology grows.

2 Methods of Limiting Route Advertisement

Two methods of controlling the frequency of route advertisement are
described here. The first involves fixed timers. The fixed timer
technique has no space overhead per route but has the disadvantage of
slowing route convergence for the normal case where a route does not
have a history of instability. The second method overcomes this

limitation at the expense of maintaining some additional space
overhead. The additional overhead includes a small amount of state
per route and a very small processing overhead.

It is possible and desirable to combine both techniques. In
practice, fixed timers have been set to very short time intervals and
have proven useful to pack routes into a smaller number of updates
when routes arrive in separate updates. The BGP protocol refers to
this as packing Network Layer Reachability Information (NLRI) [5].

Seldom are fixed timers set to the tens of minutes to hours that
would be necessary to actually damp route flap. To do so would
produce the undesirable effect of severely limiting routing
convergence.

2.1 Existing Fixed Timer Recommendations

BGP-3 does not make specific recommendations in this area [1]. The
short section entitled "Frequency of Route Selection" simply
recommends that something be done and makes broad statements
regarding certain properties that are desirable or undesirable.

BGP4 retains the "Frequency of Route Advertisement" section and adds
a "Frequency of Route Origination" section. BGP-4 describes a method
of limiting route advertisement involving a fixed (configurable)
MinRouteAdvertisementInterval timer and fixed
MinASOriginationInterval timer [5]. The recommended timer values of
MinRouteAdvertisementInterval is 30 seconds and
MinASOriginationInterval is 15 seconds.

2.2 Desirable Properties of Damping Algorithms

Before describing damping algorithms the objectives need to be
clearly defined. Some key properties are examined to clarify the
design rationale.

The overall objective is to reduce the route update load without
limiting convergence time for well behaved routes. To accomplish
this, criteria must be defined for well behaved and poorly behaved
routes. An algorithm must be defined which allows poorly behaved
routes to be identified. Ideally, this measure would be a prediction
of the future stability of a route.

Any delay in propagation of well behaved routes should be minimal.
Some delay is tolerable to support better packing of updates. Delay
of poorly behave routes should, if possible, be proportional to a
measure of the expected future instability of the route. Delay in
propagating an unstable route should cause the unstable route to be

suppressed until there is some degree of confidence that the route
has stabilized.

If a large number of route changes are received in separate updates
over some very short period of time and these updates have the
potential to be combined into a single update then these should be
packed as efficiently as possible before propagating further. Some
small delay in propagating well behaved routes is tolerable and is
necessary to allow better packing of updates.

Where routes are unstable, use and announcement of the routes should
be suppressed rather than suppressing their removal. Where one route
to a destination is stable, and another route to the same destination
is somewhat unstable, if possible, the unstable route should be
suppressed more aggressively than if there were no alternate path.

Routing consistency within an AS is very important. Only very
minimal delay of internal BGP (IBGP) should be done. Routing
consistency across AS boundaries is also very important. It is
highly undesirable to advertise a route that is different from the
route that is being used, except for a very minimal time. It is more
desirable to suppress the acceptance of a route (and therefore the
use of that route in the IGP) rather than suppress only the
redistribution.

It is clearly not possible to accurately predict the future stability
of a route. The recent history of stability is generally regarded as
a good basis for estimating the likelihood of future stability. The
criteria that is used to distinguish well behaved from poorly behaved
routes is therefore based on the recent history of stability of the
route. There is no simple quantitative expression of recent
stability so a figure of merit must be defined. Some desirable
characteristics of this figure of merit would be that the farther in
the past that instability occurred, the less it's affect on the
figure of merit and that the instability measure would be cumulative
rather than reflecting only the most recent event.

The algorithms should behave such that for routes which have a
history of stability but make a few transitions, those transitions
should be made quickly. If transitions continue, advertisement of
the route should be suppressed. There should be some memory of prior
instability. The degree to which prior instability is considered
should be gradually reduced as long as the route remains announced
and stable.

2.3 Design Choices

After routes have been accepted their readvertisement will be briefly suppressed to improve packing of updates. There may be a lengthy suppression of the acceptance of an external route. How long a route will be suppressed is based on a figure of merit that is expected to be correlated to the probability of future instability of a route. Routes with high figure of merit values will be suppressed. An exponential decay algorithm was chosen as the basis for reducing the figure of merit over time. These choices should be viewed as suggestions for implementation.

An exponential decay function has the property that previous instability can be remembered for a fairly long time. The rate at which the instability figure of merit decays slows as time goes on. Exponential decay has the following property.

$$f(f(figure\text{-}of\text{-}merit, t1), t2) = f(figure\text{-}of\text{-}merit, t1+t2)$$

This property allows the decay for a long period to be computed in a single operation regardless of the current value (figure-of-merit). As a performance optimization, the decay can be applied in fixed time increments. Given a desired decay half life, the decay for a single time increment can be computed ahead of time. The decay for multiple time increments is expressed below.

$$f(figure\text{-}of\text{-}merit, n*t0) = f(figure\text{-}of\text{-}merit, t0)**n = K**n$$

The values of K ** n can be precomputed for a reasonable number of "n" and stored in an array. The value of "K" is always less than one. The array size can be bounded since the value quickly approaches zero. This makes the decay easy to compute using an array bound check, an array lookup and a single multiply regardless as to how much time has elapsed.

3 Limiting Route Advertisements using Fixed Timers

This method of limiting route advertisements involves the use of fixed timers applied to the process of sending routes. It's primary purpose is to improve the packing of routes in BGP update messages. The delay in advertising a stable route should be bounded and minimal. The delay in advertising an unreachable need not be zero, but should also be bounded and should probably have a separate bound set less than or equal to the bound for a reachable advertisement.

The BGP protocol defines the use of a Routing Information Base (RIB). Routes that need to be readvertised can be marked in the RIB or an external set of structures maintained, which references the RIB.

Periodically, a subset of the marked routes can be flushed. This is
fairly straightforward and accomplishes the objectives. Computation
for too simple an implementation may be order N squared. To avoid N
squared performance, some form of data structure is needed to group
routes with common attributes.

An implementation should pack updates efficiently, provide a minimum
readvertisement delay, provide a bounds on the maximum
readvertisement delay that would be experienced solely as a result of
the algorithm used to provide a minimum delay, and must be
computationally efficient in the presence of a very large number of
candidates for readvertisement.

4 Stability Sensitive Suppression of Route Advertisement

This method of limiting route advertisements uses a measure of route
stability applied on a per route basis. This technique is applied
when receiving updates from external peers only (EBGP). Applying this
technique to IBGP learned routes or to advertisement to IBGP or EBGP
peers after making a route selection can result in routing loops.

A figure of merit based on a measure of instability is maintained on
a per route basis. This figure of merit is used in the decision to
suppress the use of the route. Routes with high figure of merit are
suppressed. Each time a route is withdrawn, the figure of merit is
incremented. While the route is not changing the figure of merit
value is decayed exponentially with separate decay rates depending on
whether the route is stable and reachable or has been stable and
unreachable. The decay rate may be slower when the route is
unreachable, or the stability figure of merit could remain fixed (not
decay at all) while the route remains unreachable. Whether to decay
unreachable routes at the same rate, a slower rate, or not at all is
an implementation choice. Decaying at a slower rate is recommended.

A very efficient implementation is suggested in the following
sections. The implementation only requires computation for the
routes contained in an update, when an update is received or
withdrawn (as opposed to the simplistic approach of periodically
decaying each route). The suggested implementation involves only a
small number of simple operations, and can be implemented using
scaled integers.

The behavior of unstable routes is fairly predictable. Severely
flapping routes will often be advertised and withdrawn at regular
time intervals corresponding to the timers of a particular protocol
(the IGP or exterior protocol in use where the problem exists).
Marginal circuits or mild congestion can result in a long term
pattern of occasional brief route withdrawal or occasional brief

connectivity.

4.1 Single vs. Multiple Configuration Parameter Sets

The behavior of the algorithm is modified by a number of configurable
parameters. It is possible to configure separate sets of parameters
designed to handle short term severe route flap and chronic milder
route flap (a pattern of occasional drops over a long time period).
The former would require a fast decay and low threshold (allowing a
small number of consecutive flaps to cause a route to be suppressed,
but allowing it to be reused after a relatively short period of
stability). The latter would require a very slow decay and a higher
threshold and might be appropriate for routes for which there was an
alternate path of similar bandwidth.

It may also be desirable to configure different thresholds for routes
with roughly equivalent alternate paths than for routes where the
alternate paths have a lower bandwidth or tend to be congested. This
can be solved by associating a different set of parameters with
different ranges of preference values. Parameter selection could be
based on BGP LOCAL_PREF.

Parameter selection could also be based on whether an alternate route
was known. A route would be considered if, for any applicable
parameter set, an alternate route with the specified preference value
existed and the figure of merit associated with the parameter set did
not indicate a need to suppress the route. A less aggressive
suppression would be applied to the case where no alternate route at
all existed. In the simplest case, a more aggressive suppression
would be applied if any alternate route existed. Only the highest
preference (most preferred) value needs to be specified, since the
ranges may overlap.

It might also be desirable to configure a different set of thresholds
for routes which rely on switched services and may disconnect at
times to reduce connect charges. Such routes might be expected to
change state somewhat more often, but should be suppressed if
continuous state changes indicate instability.

While not essential, it might be desirable to be able to configure
multiple sets of configuration parameters per route. It may also be
desirable to be able to configure sets of parameters that only
correspond to a set of routes (identified by AS path, peer router,
specific destinations or other means). Experience may dictate how
much flexibility is needed and how to best to set the parameters.
Whether to allow different damping parameter sets for different
routes, and whether to allow multiple figures of merit per route is
an implementation choice.

Parameter selection can also be based on prefix length. The
rationale is that longer prefixes tend to reach less end systems and
are less important and these less important prefixes can be damped
more aggressively. This technique is in fairly widespread use.
Small sites or those with dense address allocation who are multihomed
are often reachable by long prefixes which are not easily aggregated.
These sites tend to dispute the choice of prefix length for parameter
selection. Advocates of the technique point out that it encourages
better aggregation.

4.2 Configuration Parameters

At configuration time, a number of parameters may be specified by the
user. The configuration parameters are expressed in units meaningful
to the user. These differ from the parameters used at run time which
are in unit convenient for computation. The run time parameters are
derived from the configuration parameters. Suggested configuration
parameters are listed below.

   cutoff threshold (cut)

      This value is expressed as a number of route withdrawals. It is
      the value above which a route advertisement will be suppressed.

   reuse threshold (reuse)

      This value is expressed as a number of route withdrawals. It is
      the value below which a suppressed route will now be used again.

   maximum hold down time (T-hold)

      This value is the maximum time a route can be suppressed no
      matter how unstable it has been prior to this period of
      stability.

   decay half life while reachable (decay-ok)

      This value is the time duration in minutes or seconds during
      which the accumulated stability figure of merit will be reduced
      by half if the route if considered reachable (whether suppressed
      or not).

   decay half life while unreachable (decay-ng)

      This value is the time duration in minutes or seconds during
      which the accumulated stability figure of merit will be reduced
      by half if the route if considered unreachable. If not
      specified or set to zero, no decay will occur while a route

        remains unreachable.

    decay memory limit (Tmax-ok or Tmax-ng)

        This is the maximum time that any memory of previous instability
        will be retained given that the route's state remains unchanged,
        whether reachable or unreachable. This parameter is generally
        used to determine array sizes.

There may be multiple sets of the parameters above as described in
Section 4.1. The configuration parameters listed below would be
applied system wide. These include the time granularity of all
computations, and the parameters used to control reevaluation of
routes that have previously been suppressed.

    time granularity (delta-t)

        This is the time granularity in seconds used to perform all
        decay computations.

    reuse list time granularity (delta-reuse)

        This is the time interval between evaluations of the reuse
        lists. Each reuse lists corresponds to an additional time
        increment.

    reuse list memory reuse-list-max

        This is the time value corresponding to the last reuse list.
        This may be the maximum value of T-hold for all parameter sets
        of may be configured.

    number of reuse lists (reuse-list-size)

        This is the number of reuse lists. It may be determined from
        reuse-list-max or set explicitly.

A recommended optimization is described in Section 4.8.6 that
involves an array referred to as the "reuse index array". A reuse
index array is needed for each decay rate in use. The reuse index
array is used to estimate which reuse list to place a route when it
is suppressed. Proper placement avoids the need to periodically
evaluate decay to determine if a route can be reused or when storage
can be recovered. Using the reuse index array avoids the need to
compute a logarithm to determine placement. One additional system
wide parameter can be introduced.

    reuse index array size (reuse-index-array-size)

        This is the size of reuse index arrays. This size determines
        the accuracy with which suppressed routes can be placed within
        the set of reuse lists when suppressed for a long time.

4.3 Guidelines for Setting Parameters

    The decay half life should be set to a time considerably longer than
    the period of the route flap it is intended to address. For example,
    if the decay is set to ten minutes and a route is withdrawn and
    readvertised exactly every ten minutes, the route would continue to
    flap if the cutoff was set to a value of 2 or above.

    The stability figure of merit itself is an accumulated time decayed
    total. This must be kept in mind in setting the decay time, cutoff
    values and reuse values. The figure of merit is increased each time
    a route transitions from reachable to unreachable. The figure of
    merit is decayed at a rate proportional to its current value.
    Increasing the rate of route flap therefore increments the figure of
    merit more often and reaches a given threshhold in a shorter amount
    of time. When the response to a constant rate route flap is plotted
    this looks like a sawtooth with an abrupt rising edge and a decaying
    falling edge. Since the absolute decay amount is proportional to the
    figure of merit, at a continuous constant flap rate the baseline of
    the sawtooth will tend to stop rising and converge if not clipped by
    a ceiling value.

    If clipped by a ceiling value, the sawtooth baseline will simply
    reach the ceiling faster at a higher rate of route flap. For
    example, if flapping at four times the decay rate the following
    progression occurs. When the route becomes unreachable the first
    time the value becomes 1. When the next flap occurs, one is added to
    the previous value, which has been decreased by the fourth root of 2
    (the amount of decay that would occur in 1/4 of the half life time if
    decay is exponential). The sequence is 1, 1.84, 2.55, 3.14, 3.64,
    4.06, 4.42, 4.71, 4.96, 5.17, ..., converging at about 6.285. If a
    route flaps at four times the decay rate, it will reach 3 in 4
    cycles, 4 in 6 cycles, 5 in 10 cycles, and will converge at about
    6.3. At twice the decay time, it will reach 3 in 7 cycles, and
    converge at a value of less than 3.5.

    Figure 1 shows the stability figure of merit for route flap at a
    constant rate. The time axis is labeled in multiples of the decay
    half life. The plots represent route flap with a period of 1/2, 1/3,
    1/4, and 1/8 times the decay half life. A ceiling of 4.5 was set,
    which can be seen to affect three of the plots, effectively limiting
    the time it takes to readvertise the route regardless of the prior

history. With cutoff and reuse thresholds of 1.5 and 0.75, routes
would be suppressed after being declared unreachable 2-3 times and be
used again after approximately 2 decay half life periods of
stability.

This function can be expressed formally. Reachability of a route can
be represented by a variable "R" with possible values of 0 and 1
representing unreachable and reachable. At a discrete time R can
only have one value. The figure of merit is increased by 1 at each
transition from R=1 to R=0 and clipped to a ceiling value. The decay
in figure of merit can then be expressed over a set of discrete times
as follows.

    figure-of-merit(t) = K * figure-of-merit(t - delta-t)
    K = K1 for R=0 K=K2 for R=1

The four plots are presented vertically. Due to space limitations,
only a limited set of points along the time axis are shown. The
value of the figure of merit is given. Along side each value is a
very low resolution strip chart made up of ASCII dots. This is just
intended to give a rough feel for the rise and fall of the values.
The strip charts are not displayed on an overlapping set of axes
because the sawtooth waveforms cross each other quite frequently. At
the very low resolution of these plots, the rise and fall of the
baseline is evident, but the sawtooth nature is only observed in the
printed value.

From the maximum hold time value (T-hold), a ratio of the reuse value
to a ceiling can be determined. An integer value for the ceiling can
then be chosen such that overflow will not be a problem and all other
values can be scaled accordingly. If both cutoffs are specified or
if multiple parameter sets are used the highest ceiling will be used.

   time   figure-of-merit as a function of time (in minutes)

```
0.00    0.000 .          0.000 .            0.000 .            0.000 .
0.08    0.000 .          0.000 .            0.000 .            0.000 .
0.16    0.000 .          0.000 .            0.000 .            0.973  .
0.24    0.000 .          0.000 .            0.000 .            0.920  .
0.32    0.000 .          0.000 .            0.946  .           1.817    .
0.40    0.000 .          0.953  .           0.895  .           2.698      .
0.48    0.000 .          0.901  .           0.847  .           2.552      .
0.56    0.953  .         0.853  .           1.754    .         3.367        .
0.64    0.901  .         0.807  .           1.659    .         4.172          .
0.72    0.853  .         1.722    .         1.570    .         3.947          .
0.80    0.807  .         1.629    .         2.444      .       4.317          .
0.88    0.763  .         1.542    .         2.312      .       4.469          .
0.96    0.722  .         1.458    .         2.188      .       4.228          .
1.04    1.649    .       2.346      .       3.036        .     4.347          .
1.12    1.560    .       2.219      .       2.872        .     4.112          .
1.20    1.476    .       2.099      .       2.717        .     4.257          .
1.28    1.396    .       1.986      .       3.543          .   4.377          .
1.36    1.321    .       2.858        .     3.352          .   4.141          .
1.44    1.250    .       2.704        .     3.171          .   4.287          .
1.52    2.162      .     2.558        .     3.979            . 4.407          .
1.60    2.045      .     2.420        .     3.765            . 4.170          .
1.68    1.935      .     3.276          .   3.562            . 4.317          .
1.76    1.830      .     3.099          .   4.356            . 4.438          .
1.84    1.732      .     2.932          .   4.121            . 4.199          .
1.92    1.638      .     2.774          .   3.899            . 3.972          .
2.00    1.550      .     2.624          .   3.688            . 3.758            .
2.08    1.466      .     2.483          .   3.489            . 3.555            .
2.16    1.387      .     2.349          .   3.301          .   3.363            .
2.24    1.312      .     2.222          .   3.123          .   3.182            .
2.32    1.242      .     2.102          .   2.955          .   3.010            .
2.40    1.175      .     1.989          .   2.795          .   2.848            .
2.48    1.111    .       1.882        .     2.644          .   2.694          .
2.56    1.051    .       1.780        .     2.502          .   2.549          .
2.64    0.995    .       1.684        .     2.367          .   2.411          .
2.72    0.941    .       1.593        .     2.239          .   2.281          .
2.80    0.890    .       1.507        .     2.118          .   2.158        .
2.88    0.842    .       1.426        .     2.004          .   2.042        .
2.96    0.797    .       1.349        .     1.896          .   1.932        .
3.04    0.754    .       1.276        .     1.794          .   1.828        .
3.12    0.713    .       1.207        .     1.697          .   1.729        .
3.20    0.675    .       1.142        .     1.605        .     1.636      .
3.28    0.638    .       1.081        .     1.519        .     1.547      .
3.36    0.604    .       1.022        .     1.437        .     1.464      .
3.44    0.571    .       0.967        .     1.359        .     1.385      .
```

   Figure 1: Instability figure of merit for flap at a constant rate

   time   figure-of-merit as a function of time (in minutes)

   0.00   0.000 .   0.000 .   0.000 .
   0.20   0.000 .   0.000 .   0.000 .
   0.40   0.000 .   0.000 .   0.000 .
   0.60   0.000 .   0.000 .   0.000 .
   0.80   0.000 .   0.000 .   0.000 .
   1.00   0.999 .   0.999 .   0.999 .
   1.20   0.971 .   0.971 .   0.929 .
   1.40   0.945 .   0.945 .   0.809 .
   1.60   0.919 .   0.865 .   0.704 .
   1.80   0.894 .   0.753 .   0.613 .
   2.00   1.812 .   1.657 .   1.535 .
   2.20   1.762 .   1.612 .   1.428 .
   2.40   1.714 .   1.568 .   1.244 .
   2.60   1.667 .   1.443 .   1.083 .
   2.80   1.622 .   1.256 .   0.942 .
   3.00   1.468 .   1.094 .   0.820 .
   3.20   2.400   .   2.036   .   1.694   .
   3.40   2.335   .   1.981   .   1.475   .
   3.60   2.271   .   1.823   .   1.284   .
   3.80   2.209   .   1.587 .   1.118 .
   4.00   1.999   .   1.381 .   0.973 .
   4.20   2.625   .   2.084   .   1.727   .
   4.40   2.285   .   1.815   .   1.503   .
   4.60   1.990   .   1.580   .   1.309   .
   4.80   1.732   .   1.375   .   1.139   .
   5.00   1.508   .   1.197   .   0.992 .
   5.20   1.313   .   1.042 .   0.864 .
   5.40   1.143   .   0.907 .   0.752 .
   5.60   0.995 .   0.790 .   0.654 .
   5.80   0.866 .   0.688 .   0.570 .
   6.00   0.754 .   0.599 .   0.496 .
   6.20   0.656 .   0.521 .   0.432 .
   6.40   0.571 .   0.454 .   0.376 .
   6.60   0.497 .   0.395 .   0.327 .
   6.80   0.433 .   0.344 .   0.285 .
   7.00   0.377 .   0.299 .   0.248 .
   7.20   0.328 .   0.261 .   0.216 .
   7.40   0.286 .   0.227 .   0.188 .
   7.60   0.249 .   0.197 .   0.164 .
   7.80   0.216 .   0.172 .   0.142 .
   8.00   0.188 .   0.150 .   0.124 .

        Figure 2: Separate decay constants when unreachable

Figure 2 shows the effect of configuring separate decay rates to be
used when the route is reachable or unreachable. The decay rate is 5
times slower when the route is unreachable. In the three case shown,
the period of the route flap is equal to the decay half life but the
route is reachable 1/8 of the time in one, reachable 1/2 the time in
one, and reachable 7/8 of the time in the other. In the last case
the route is not suppressed until after the third unreachable (when
it is above the top threshold after becoming reachable again).

The main point of Figure 2 is to show the effect of changing the duty
cycle of the square wave in the variable "R" for a fixed frequency of
the square wave. If the decay constants are chosen such that decay
is slower when R=0 (the route is unreachable), then the figure of
merit rises more slowly (more accurately, the baseline of the
sawtooth waveform rises more slowly) if the route is reachable a
larger percentage of the time. The effect when the route becomes
persistently reachable again can be fairly negligible if the sawtooth
is clipped by a ceiling value, but is more significant if a slow
route flap rate or short interval of route flapping is such that the
sawtooth does not reach the ceiling value. In Figure 2 the interval
in which the routes are unstable is short enough that the ceiling
value is not reached, therefore, the routes that are reachable for a
greater percentage of the route flap cycle are reused (placed in the
RIB and advertised to peers) sooner than others after the route
becomes stable again ("R" becomes 1, indicating the announced state
goes to reachable and remains there).

In both Figure 1 and Figure 2, routes would be suppressed. Routes
flapping at the decay half life or less would be withdrawn two or
three times and then remain withdrawn until they had remained stably
announced and stable for on the order of 1 1/2 to 2 1/2 times the
decay half life (given the ceiling in the example).

The purpose of damping BGP route flap is to reduce the processor
burden at the immediate router and the processor burden to downstream
routers (BGP peer routers and peers of peers that will see the route
announcements advertised by the immediate router). Computing a
figure of merit at each discrete time interval using figure-of-
merit(t) = K * figure-of-merit(t - delta-t) would be very inefficient
and defeat the purpose. This problem is addressed by defering
computation as long as possible and doing a single simple computation
to compensate for the decay during the time that has elapsed since
the figure of merit was last updated. The use of decay arrays
provides the single simple calculation. The use of reuse lists
(described later) provide a means to defer calculations. A route
becomes usable if there was not further change for a period of time
and the route is unreachable. The data structure storage is
recovered if the route's state has not changed for a period of time

and it has been unreachable. The reuse arrays provide a means to
estimate how long a computation can be deferred if there is no
further change.

A larger time granularity will keep table storage down. The time
granularity should be less than a minimal reasonable time between
expected worse case route flaps. It might be reasonable to fix this
parameter at compile time or set a default and strongly recommend
that the user leave it alone. With an exponential decay, array size
can be greatly reduced by setting a period of complete stability
after which the decayed total will be considered zero rather than
retaining a tiny quantity. Alternately, very long decays can be
implemented by multiplying more than once if array bounds are
exceeded.

The reuse lists hold suppressed routes grouped according to how long
it will be before the routes are eligible for reuse. Periodically
each list will be advanced by one position and one list removed as
described in Section 4.8.7. All of the suppressed routes in the
removed list will be reevaluated and either used or placed in another
list according to how much additional time must elapse before the
route can be reused. The last list will always contain all the
routes which will not be advertised for more time than is appropriate
for the remaining list heads. When the last list advances to the
front, some of the routes will not be ready to be used and will have
to be requeued. The time interval for reconsidering suppressed
routes and number of list heads should be configurable. Reasonable
defaults might be 30 seconds and 64 list heads. A route suppressed
for a long time would need to be reevaluated every 32 minutes.

4.4 Run Time Data Structures

A fixed small amount of per system storage will be required. Where
sets of multiple configuration parameters are used, storage will be
required per set of parameters. A small amount of per route storage
is required. A set of list heads is needed. These list heads are
used to arrange suppressed routes according to the time remaining
until they can be reused.

A separate reuse list can be used to hold unreachable routes for the
purpose of later recovering storage if they remain unreachable too
long. This might be more accurately described as a recycling list.
The advantage this would provide is making free data structures
available as soon as possible. Alternately, the data structures can
simply be placed on a queue and the storage recovered when the route
hits the front of the queue and if storage is needed. The latter is
less optimal but simple.

If multiple sets of configuration parameters are allowed per route,
there is a need for some means of associating more than one figure of
merit and set of parameters with each route. Building a linked list
of these objects seems like one of a number of reasonable
implementations. Similarly, a means of associating a route to a
reuse list is required. A small overhead will be required for the
pointers needed to implement whatever data structure is chosen for
the reuse lists. The suggested implementation uses a double linked
lists and so requires two pointers per figure of merit.

Each set of configuration parameters can reference decay arrays and
reuse arrays. These arrays should be shared among multiple sets of
parameters since their storage requirement is not negligible. There
will be only one set of reuse list heads for the entire router.

4.4.1 Data Structures for Configuration Parameter Sets

Based on the configuration parameters described in the previous
section, the following values can be computed as scaled integers
directly from the corresponding configuration parameters.

o decay array scale factor (decay-array-scale-factor)

o cutoff value (cut)

o reuse value (reuse)

o figure of merit ceiling (ceiling)

Each configuration parameter set will reference one or two decay
arrays and one or two reuse arrays. Only one array will be needed if
the decay rate is the same while a route is unreachable as while it
is reachable, or if the stability figure of merit does not decay
while a route is unreachable.

4.4.2 Data Structures per Decay Array and Reuse Index Array

The following are also computed from the configuration parameters
though not as directly. The computation is described in Section 4.5.

o decay rate per tick (decay-delta-t)

o decay array size (decay-array-size)

o decay array (decay[])

o reuse index array size (reuse-index-array-size)

   o   reuse index array (reuse-index-array[])

   For each decay rate specified, an array will be used to store the
   value of a computed parameter raised to the power of the index of
   each array element. This is to speed computations. The decay rate
   per tick is an intermediate value expressed as a real number and used
   to compute the values stored in the decay arrays. The array size is
   computed from the decay memory limit configuration parameter
   expressed as an array size or as a maximum hold time.

   The decay array size must be of sufficient size to accommodate the
   specified decay memory given the time granularity, or sufficient to
   hold the number of array elements until integer rounding produces a
   zero result if that value is smaller, or a implementation imposed
   reasonable size to prevent configurations which use excessive memory.
   Implementations may chose to make the array size shorter and multiply
   more than once when decaying a long time interval to reduce storage.

   The reuse index arrays serve a similar purpose to the decay arrays.
   In BGP, a route is said to be "used" if it is considered the best
   route. In this context, if the route is "used" it is placed in the
   RIB and is eligible for advertisement to BGP peers. If a route is
   withdrawn (a BGP announcement is made by a peer indicating that it is
   no longer reachable), then it is no longer eligible for "use". When
   a route becomes reachable it may not be "used" immediately if the
   figure of merit indicates that a recent instability has occurred.
   After the route remains stable and the figure of merit decays below
   the "reuse" threshhold, the route is said to be eligible to be
   "reused" (treated as truly reachable, placed in the RIB and
   advertised to peers). The amount of time until a route can be reused
   can be determined using a array lookup. The array can be built given
   the decay rate. The array is indexed using a scaled integer
   proportional to the ratio between a current stability figure of merit
   value and the value needed for the route to be reused.

4.4.3 Per Route State

   Information must be maintained per some tuple representing a route.
   At the very minimum, the NLRI (BGP prefix and length) must be
   contained in the tuple. Different BGP attributes may be included or
   excluded depending on the specific situation. The AS path should
   also be contained in the tuple by default. The tuple may also
   optionally contain other BGP attributes such as
   MULTI_EXIT_DISCRIMINATOR (MED).

   The tuple representing a route for the purpose of route flap damping
   is:

```
        tuple entry           default      options
        -----------------------------------------
        NLRI
          prefix              required
          length              required
        AS path               included     option to exclude
        last AS set in path   excluded     option to include
        next hop              excluded     option to include
        MED                   excluded     option to include
                                           in comparisons only
```

The AS path is generally included in order to identify downstream instability which is not being damped or not being sufficiently damped and is alternating between a stable and an unstable path. Under rare circumstances it may be desirable to exclude AS path for all or a subset of prefixes. If an AS path ends in an AS set, in practice the path is always for an aggregate. Changes to the trailing AS set should be ignored. Ideally the AS path comparison should insure that at least one AS has remained constant in the old and new AS set, but completely ignoring the contents of a trailing AS set is also acceptable.

Including next hop and MED changes can help suppress the use of an AS which is internally unstable or avoid a next hop which is closer to an unstable IGP path in the adjacent AS. If a large number of MED values are used, the increase in the amount of state may become a problem. For this reason MED is disabled by default and enabled only as part of the tuple comparison, using a single state entry regardless of MED value. Including MED will suppress the use of the adjacent AS even though the change need not be propagated further. Using MED is only a safe practice if a path is known to exist through another AS or where there are enough peering sites with the adjacent AS such that routes heard at only a subset of the peering sites will be suppressed.

4.4.4 Data Structures per Route

The following information must be maintained per route. A route here is considered to be a tuple usually containing NLRI, next hop, and AS path as defined in Section 4.4.3.

   stability figure of merit (figure-of-merit)

      Each route must have a stability figure of merit per applicable parameter set.

   last time updated (time-update)

The exact last time updated must be maintained to allow
exponential decay of the accumulated figure of merit to be
deferred until the route might reasonable be considered eligible
for a change in status (having gone from unreachable to
reachable or advancing within the reuse lists).

config block pointer

Any implementation that supports multiple parameter sets must
provide a means of quickly identifying which set of parameters
corresponds to the route currently being considered. For
implementations supporting only parameter sets where all routes
must be treated the same, this pointer is not required.

reuse list traversal pointers

If doubly linked lists are used to implement reuse lists, then
two pointers will be needed, previous and next. Generally there
is a double linked list which is unused when a route is
suppressed from use that can be used for reuse list traversal
eliminating the need for additional pointer storage.

4.5 Processing Configuration Parameters

From the configuration parameters, it is possible to precompute
a number of values that will be used repeatedly and retain these
to speed later computations that will be required frequently.

Scaling is usually dependent on the highest value that figure-
of-merit can attain, referred to here as the ceiling. The real
number value of the ceiling will typically be determined by the
following equation. The ceiling can also be configured to a
specific value, which in turn dictates T-hold.

$$ceiling = reuse * (exp(T\text{-}hold/decay\text{-}half\text{-}life) * log(2))$$

In the above equation, reuse is the reuse threshhold described
in Section 4.2.

The methods of scaled integer arithmetic are not described in
detail here. The methods of determining the real values are
given. Translation into scaled integer values and the details
of scaled integer arithmetic are left up to the individual
implementations.

The ceiling value can be set to be the largest integer that can fit
in half the bits available for an unsigned integer. This will
allow the scaled integers to be multiplied by the scaled decay

value and then shifted down. Implementations may prefer to use
real numbers or may use any integer scaling deemed appropriate for
their architecture.

penalty value and thresholds (as proportional scaled integers)

   The figure of merit penalty for one route withdrawal and the
   cutoff values must be scaled according to the above scaling
   factor.

decay rate per tick (decay[1])

   The decay value per increment of time as defined by the time
   granularity must be determined (at least initially as a floating
   point number). The per tick decay is a number slightly less
   than one. It is the Nth root of the one half where N is the
   half life divided by the time granularity.

      decay[1] = exp ((1 / (decay-half-life/delta-t)) * log (1/2))

decay array size (decay-array-size)

   The decay array size is the decay memory divided by the time
   granularity. If integer truncation brings the value of an array
   element to zero, the array can be made smaller. An
   implementation should also impose a maximum reasonable array
   size or allow more than one multiplication.

                     decay-array-size = (Tmax/delta-t)

decay array (decay[])

   Each i-th element of the decay array is the per tick delay
   raised to the i-th power. This might be best done by successive
   floating point multiplies followed by scaling and integer
   rounding or truncation. The array itself need only be computed
   at startup.

                        decay[i] = decay[1] ** i

4.6 Building the Reuse Index Arrays

   The reuse lists may be accessed quite frequently if a lot of routes
   are flapping sufficiently to be suppressed. A method of speeding the
   determination of which reuse list to use for a given route is
   suggested. This method is introduced in Section 4.2, its
   configuration described in Section 4.4.2 and the algorithms described
   in Section 4.8.6 and Section 4.8.7. This section describes building

the reuse list index arrays.

A ratio of the figure of merit of the route under consideration to
the cutoff value is used as the basis for an array lookup. The ratio
is scaled and truncated to an integer and used to index the array.
The array entry is an integer used to determine which reuse list to
use.

  reuse array maximum ratio (max-ratio)

    This is the maximum ratio between the current value of the
    stability figure of merit and the target reuse value that can be
    indexed by the reuse array. It may be limited by the ceiling
    imposed by the maximum hold time or by the amount of time that
    the reuse lists cover.

      max-ratio = min(ceiling/reuse, exp((1 / (half-life/reuse-
    array-time)) * log(2)))

  reuse array scale factor (scale-factor)

    Since the reuse array is an estimator, the reuse array scale
    factor has to be computed such that the full size of the reuse
    array is used.

        scale-factor = reuse-index-array-size / (max-ratio - 1)

  reuse index array (reuse-index-array[])

    Each reuse index array entry should contain an index into the
    reuse list array pointing to one of the list heads. This index
    should corresponding to the reuse list that will be evaluated
    just after a route would be eligible for reuse given the ratio
    of current value of the stability figure of merit to target
    reuse value corresponding the the reuse array entry.

      reuse-index-array[j] = integer((decay-half-life / reuse-
    time-granularity) * log(1/(reuse * (1 + (j / scale-factor)))) /
    log(1/2))

To determine which reuse queue to place a route which is being
suppressed, the following procedure is used. Divide the current
figure of merit by the cutoff. Subtract one. Multiply by the scale
factor. This is the index into the reuse index array (reuse-index-
array[]). The value fetched from the reuse index array (reuse-
index-array[]) is an index into the array of reuse lists (reuse-
array[]). If this index is off the end of the array use the last
queue otherwise look in the array and pick the number of the queue

from the array at that index. This is quite fast and well worth the
setup and storage required.

4.7 A Sample Configuration

A simple example is presented here in which the space overhead is
estimated for a set of configuration parameters. The design here
assumes:

1. there is a single parameter set used for all routes,

2. decay time for unreachable routes is slower than for reachable
    routes

3. the arrays must be full size, rather than allow more than one
    multiply per decay operation to reduce the array size.

This example is used in later sections. The use of multiple
parameter sets complicates the examples somewhat. Where multiple
parameter sets are allowed for a single route, the decay portion of
the algorithm is repeated for each parameter set. If different
routes are allowed to have different parameter sets, the routes must
have pointers to the parameter sets to keep the time to locate to a
minimum, but the algorithms are otherwise unchanged.

A sample set of configuration parameters and a sample set of
implementation parameters are provided in in the two following lists.

   1. Configuration Parameters

       o cut = 1.25

       o reuse = 0.5

       o T-hold = 15 mins

       o decay-ok = 5 min

       o decay-ng = 15 min

       o Tmax-ok, Tmax-ng = 15, 30 mins

   2. Implementation Parameters

       o delta-t = 1 sec

       o delta-reuse = 15 sec

   o reuse-list-size = 256

   o reuse-index-array-size = 1,024

Using these configuration and implementation parameters and the
equations in Section 4.5, the space overhead can be computed. There
is a fixed space overhead that is independent of the number of
routes. There is a space requirement associated with a stable route.
There is a larger space requirement associated with an unstable
route. The space requirements for the parameters above are provide
in the lists below.

   1. fixed overhead (using parameters from previous example)

   o 900 * integer - decay array

   o 1,800 * integer - decay array

   o 120 * pointer - reuse list-heads

   o 2,048 * integer - reuse index arrays

   2. overhead per stable route

   o pointer - containing null entry

   3. overhead per unstable route

   o pointer - to a damping structure containing the following

   o integer - figure of merit + bit for state

   o integer - last time updated

   o 2 * pointer - reuse list pointers (prev, next)

The decay arrays are sized acording to delta-t and Tmax-ok or Tmax-
ng. The number of reuse list-heads is based on delta-reuse and the
greater of Tmax-ok or Tmax-ng. There are two reuse index arrays
whose size is a configured parameter.

Figure 3 shows the behavior of the algorithm with the parameters
given above. Four cases are given in this example. In all four,
there is a twelve minute period of route oscillations. Two periods
of oscillation are used, 2 minutes and 4 minutes. Two duty cycles
are used, one in which the route is reachable during 20% of the cycle
and the other where the route is reachable during 80% of the cycle.
In all four cases, the route becomes suppressed after it becomes

unreachable the second time. Once suppressed, it remains suppressed
until some period after becoming stable. The routes which oscillate
over a 4 minute period are no longer suppressed within 9-11 minutes
after becoming stable. The routes with a 2 minute period of
oscillation are suppressed for nearly the maximum 15 minute period
after becoming stable.

4.8 Processing Routing Protocol Activity

The prior sections concentrate on configuration parameters and their
relationship to the parameters and arrays used at run time and
provide the algorithms for initializing run time storage. This
section provides the steps taken in processing routing events and
timer events when running.

The routing events are:

  1. A BGP peer or new route comes up for the first time (or after
      an extended down time) (Section 4.8.1)

  2. A route becomes unreachable (Section 4.8.2)

  3. A route becomes reachable again (Section 4.8.3)

  4. A route changes (Section 4.8.4)

  5. A peer goes down (Section 4.8.5)

```
    time       figure-of-merit as a function of time (in minutes)

    0.00   0.000 .           0.000 .          0.000 .           0.000 .
    0.62   0.000 .           0.000 .          0.000 .           0.000 .
    1.25   0.000 .           0.000 .          0.000 .           0.000 .
    1.88   0.000 .           0.000 .          0.000 .           0.000 .
    2.50   0.977 .           0.968 .          0.000 .           0.000 .
    3.12   0.949 .           0.888 .          0.000 .           0.000 .
    3.75   0.910 .           0.814 .          0.000 .           0.000 .
    4.37   1.846  .          1.756  .         0.983 .           0.983 .
    5.00   1.794  .          1.614  .         0.955 .           0.935 .
    5.63   1.735  .          1.480  .         0.928 .           0.858 .
    6.25   2.619   .         2.379   .        0.901 .           0.786 .
    6.88   2.544   .         2.207   .        0.876 .           0.721 .
    7.50   2.472   .         2.024   .        0.825 .           0.661 .
    8.13   3.308    .        2.875    .       1.761  .          1.608   .
    8.75   3.213    .        2.698    .       1.711  .          1.562   .
    9.38   3.122    .        2.474    .       1.662  .          1.436  .
   10.00   3.922     .       3.273     .      1.615  .          1.317  .
   10.63   3.810     .       3.107     .      1.569  .          1.207  .
   11.25   3.702     .       2.849     .      1.513  .          1.107  .
   11.88   3.498     .       2.613     .      1.388  .          1.015  .
   12.50   3.904     .       3.451      .     2.312   .         1.953   .
   13.13   3.580     .       3.164      .     2.120   .         1.791  .
   13.75   3.283     .       2.902     .      1.944  .          1.643  .
   14.38   3.010     .       2.661     .      1.783  .          1.506  .
   15.00   2.761    .        2.440     .      1.635  .          1.381  .
   15.63   2.532    .        2.238     .      1.499  .          1.267  .
   16.25   2.321    .        2.052     .      1.375  .          1.161  .
   16.88   2.129    .        1.882     .      1.261  .          1.065  .
   17.50   1.952    .        1.725     .      1.156  .          0.977  .
   18.12   1.790    .        1.582     .      1.060  .          0.896  .
   18.75   1.641    .        1.451    .       0.972 .           0.821  .
   19.38   1.505    .        1.331    .       0.891 .           0.753  .
   20.00   1.380   .         1.220    .       0.817 .           0.691  .
   20.62   1.266   .         1.119    .       0.750 .           0.633  .
   21.25   1.161   .         1.026    .       0.687 .           0.581  .
   21.87   1.064   .         0.941   .        0.630 .           0.533  .
   22.50   0.976  .          0.863   .        0.578 .           0.488 .
   23.12   0.895  .          0.791   .        0.530 .           0.448 .
   23.75   0.821  .          0.725   .        0.486 .           0.411 .
   24.37   0.753  .          0.665   .        0.446 .           0.377 .
   25.00   0.690  .          0.610   .        0.409 .           0.345 .
```

Figure 3: Some fairly long route flap cycles, repeated for 12 minutes,
          followed by a period of stability.

The reuse list is used to provide a means of fast evaluation of route
that had been suppressed, but had been stable long enough to be
reused again or had been suppressed long enough that it can be
treated as a new route. The following two operations are described.

1. Inserting into a reuse list (Section 4.8.6)

2. Reuse list processing every delta-t seconds (Section 4.8.7)

4.8.1 Processing a New Peer or New Routes

When a peer comes up, no action is required if the routes had no
previous history of instability, for example if this is the first
time the peer is coming up and announcing these routes. For each
route, the pointer to the damping structure would be zeroed and route
used. The same action is taken for a new route or a route that has
been down long enough that the figure of merit reached zero and the
damping structure was deleted.

4.8.2 Processing Unreachable Messages

When a route is withdrawn or changed (Section 4.8.4 describes how a
change is handled), the following procedure is used.

If there is no previous stability history (the damping structure
pointer is zero), then:

1. allocate a damping structure

2. set figure-of-merit = 1

3. withdraw the route

Otherwise, if there is an existing damping structure, then:

1. set t-diff = t-now - t-updated

2. if (t-diff puts you off the end of the array) {

setfigure-of-merit =1

}else {

setfigure-of-merit =figure-of-merit *decay-array-ok [t-diff]+ 1

if(figure-of-merit >ceiling) {

setfigure-of-merit =ceiling

```
     }

    }
```

   3. remove the route from a reuse list if it is on one

   4. withdraw the route unless it is already suppressed

   In either case then:

   1. set t-updated = t-now

   2. insert into a reuse list (see Section 4.8.6)

   If there was a stability history, the previous value of the stability
   figure of merit is decayed. This is done using the decay array
   (decay-array). The index is determined by subtracting the current
   time and the last time updated, then dividing by the time
   granularity. If the index is zero, the figure of merit is unchanged
   (no decay). If it is greater than the array size, it is zeroed.
   Otherwise use the index to fetch a decay array element and multiply
   the figure of merit by the array element. If using the suggested
   scaled integer method, shift down half an integer. Add the scaled
   penalty for one more unreachable (shown above as 1). If the result
   is above the ceiling replace it with the ceiling value. Now update
   the last time updated field (preferably taking into account how much
   time was truncated before doing the decay calculation).

   When a route becomes unreachable, alternate paths must be considered.
   This process is complicated slightly if different configuration
   parameters are used in the presence or absence of viable alternate
   paths. If all of these alternate paths have been suppressed because
   there had previously been an alternate route and the new route
   withdrawal changes that condition, the suppressed alternate paths
   must be reevaluated. They should be reevaluated in order of normal
   route preference. When one of these alternate routes is encountered
   that had been suppressed but is now usable since there is no
   alternate route, no further routes need to be reevaluated. This only
   applies if routes are given two different reuse thresholds, one for
   use when there is an alternate path and a higher threshold to use
   when suppressing the route would result in making the destination
   completely unreachable.

4.8.3 Processing Route Advertisements

   When a route is readvertised if there is no damping structure, then
   the procedure is the same as in Section 4.8.1.

   1. don't create a new damping structure

   2. use the route

If an damping structure exists, the figure of merit is decayed and
the figure of merit and last time updated fields are updated. A
decision is now made as to whether the route can be used immediately
or needs to be suppressed for some period of time.

   1. set t-diff = t-now - t-updated

   2. if (t-diff puts you off the end of the array) {

          set figure-of-merit =0

       }else {

          set figure-of-merit= figure-of-merit* decay-array-ng[t-diff]

       }

   3. if (not suppressed and figure-of-merit < cut) {

          use the route

       }else if(suppressed and figure-of-merit< reuse) {

         set state tonot suppressed

         remove the route from a reuse list

         use the route

       }else {

         set state to suppressed

         don't use the route

         insert into a reuse list (see Section 4.8.6)

       }

   4. if (figure-of-merit > 0) {

          set t-updated= t-now

       }else {

```
            recover memory for damping struct

            zero pointer to damping struct

        }
```

If the route is deemed usable, a search for the current best route
must be made. The newly reachable route is then evaluated according
to the BGP protocol rules for route selection.

If the new route is usable, the previous best route is examined.
Prior to route comparisons, the current best route may have to be
reevaluated if separate parameter sets are used depending on the
presence or absence of an alternate route. If there had been no
alternate the previous best route may be suppressed.

If the new route is to be suppressed it is placed on a reuse list
only if it would have been preferred to the current best route had
the new route been accepted as stable. There is no reason to queue a
route on a reuse list if after the route becomes usable it would not
be used anyway due to the existence of a more preferred route. Such
a route would not have to be reevaluated unless the preferred route
became unreachable. As specified here, the less preferred route
would be reevaluated and potentially used or potentially added to a
reuse list when processing the withdrawal of a more preferred best
route.

4.8.4 Processing Route Changes

If a route is replaced by a peer router by supplying a new path, the
route that is being replaced should be treated as if an unreachable
were received (see Section 4.8.2). This will occur when a peer
somewhere back in the AS path is continuously switching between two
AS paths and that peer is not damping route flap (or applying less
damping). There is no way to determine if one AS path is stable and
the other is flapping, or if they are both flapping. If the cycle is
sufficiently short compared to convergence times neither route
through that peer will deliver packets very reliably. Since there is
no way to affect the peer such that it chooses the stable of the two
AS paths, the only viable option is to penalize both routes by
considering each change as an unreachable followed by a route
advertisement.

4.8.5 Processing A Peer Router Loss

When a peer routing session is broken, either all individual routes
advertised by that peer may be marked as unstable, or the peering
session itself may be marked as unstable. Marking the peer will save

considerable memory. Since the individual routes are advertised as
unreachable to routers beyond the immediate problem, per route state
will be incurred beyond the peer immediately adjacent to the BGP
session that went down. If the instability continues, the
immediately adjacent router need only keep track of the peer
stability history. The routers beyond that point will receive no
further advertisements or withdrawal of routes and will dispose of
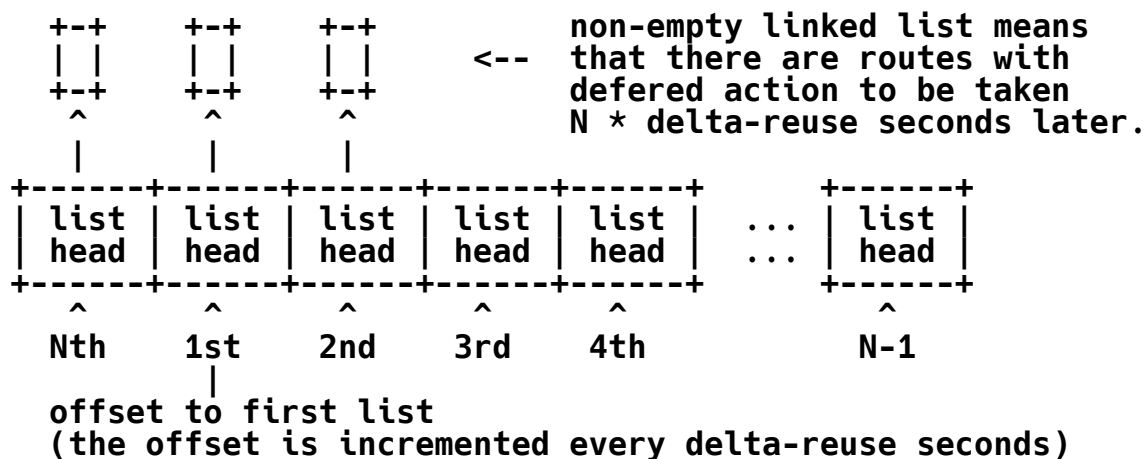the damping structure over time.

BGP notification through an optional transitive attribute that
damping will already be applied may be considered in the future to
reduce the number of routers that incur damping structure storage
overhead.

4.8.6 Inserting into the Reuse Timer List

The reuse lists are used to provide a means of fast evaluation of
route that had been suppressed, but had been stable long enough to be
reused again. The data structure consists of a series of list heads.
Each list contains a set of routes that are scheduled for
reevaluation at approximately the same time. The set of reuse list
heads are treated as a circular array. Refer to Figure 4.

A simple implementation of the circular array of list heads would be
an array containing the list heads. An offset is used when accessing
the array. The offset would identify the first list. The Nth list
would be at the index corresponding to N plus the offset modulo the
number of list heads. This design will be assumed in the examples
that follow.

A key requirement is to be able to insert an entry in the most
appropriate queue with a minimum of computation. The computation is
given only the current value of figure-of-merit. Instead of a
computation which would involve a logarithm, the reuse array (reuse-
array[]) described in Section 4.6 is used. The array, scale, and
bounds are precomputed to map figure-of-merit to the nearest list
head without requiring a logarithm to be computed (see Section 4.5).

```
  +-+      +-+      +-+              non-empty linked list means
  | |      | |      | |       <--   that there are routes with
  +-+      +-+      +-+              defered action to be taken
   ^        ^        ^              N * delta-reuse seconds later.
   |        |        |
+------+------+------+------+------+       +------+
| list | list | list | list | list |  ... | list |
| head | head | head | head | head |  ... | head |
+------+------+------+------+------+       +------+
   ^       ^       ^       ^       ^          ^
  Nth     1st     2nd     3rd     4th        N-1
           |
  offset to first list
  (the offset is incremented every delta-reuse seconds)
```

                Figure 4: Reuse List Data Structures

Note that in the following sections the operator prefix notation
"modulo a b" means "b % a" in C language algebraic operator notation.
For example, "modulo 16 1023" would be 15.

  1. scale figure-of-merit for the index array lookup producing
      index

  2. check index against the array bound

  3. if (within the array bound) {

         set index =reuse-array [index]

      }else {

         set index =reuse-list-size -1

      }

  4. insert into the list

         reuse-list[moduloreuse-list-size (index +offset)]

Choosing the correct reuse list involves only a multiply and shift to
do the scaling, an integer truncation, then an array lookup in the
reuse array (reuse-array[]). The value retrieved from the reuse
array is used to select a reuse list. The reuse list is a circular
list. The most common method of implementing a circular list is to
use an array and apply an offset and modulo operation to pick the
correct array entry. The offset is incremented to rotate the
circular list.

4.8.7 Handling Reuse Timer Events

   The granularity of the reuse timer should be more coarse than that of
   the decay timer. As a result, when the reuse timer fires, suppressed
   routes should be decayed by multiple increments of decay time. Some
   computation can be avoided by always inserting into the reuse list
   corresponding to one time increment past reuse eligibility. In cases
   where the reuse lists have a longer "memory" than the "decay memory"
   (described above), all of the routes in the first queue will be
   available for immediate reuse if reachable or the history entry could
   be disposed of if unreachable.

   When it is time to advance the lists, the first queue on the reuse
   list must be processed and the circular queue must be rotated. Using
   an array and an offset as a circular array (as described in Section
   4.8.6), the algorithm below is repeated every delta-reuse seconds.

      1.   save a pointer to the current zeroth queue head and zero the
           list head entry

      2.   set offset = modulo reuse-list-size (offset + 1), thereby
           rotating the circular queue of list-heads

      3.   if (the saved list head pointer is non-empty)

           for each entry {

             sett-diff =t-now -t-updated

             set figure-of-merit =figure-of-merit *decay-array-ok [t-diff]

             sett-updated =t-now

             if(figure-of-merit< reuse)

               reuse the route

             else

               re-insert into another list (seeSection 4.8.6)

           }

   The value of the zeroth list head would be saved and the array entry
   itself zeroed. The list heads would then be advanced by incrementing
   the offset. Starting with the saved head of the old zeroth list,
   each route would be reevaluated and used, disposed of entirely or
   requeued if it were not ready for reuse. If a route is used, it must

be treated as if it were a new route advertisement as described in
Section 4.8.3.

5 Implementation Experience

The first implementations of "route flap damping" were the route
server daemon (rsd) coding by Ramesh Govindan (ISI) and the Cisco IOS
implementation by Ravi Chandra. Both implementations first became
available in 1995 and have been used extensively. The rsd
implementation has been in use in route servers at the NSF funded
Network Access Points (NAPs) and at other major Internet
interconnects. The Cisco IOS version has been in use by Internet
Service Providers worldwide. The rsd implementation has been
integrated in releases of gated (see http://www.gated.org) and is
available in commercial routers using gated.

There are now more than 2 years of BGP route damping deployment
experience. Some problems have occurred in deployment. So far these
are solvable by careful implementation of the algorithm and by
careful deployment. In some topologies coordinated deployment can be
helpful and in all cases disclosure of the use of route damping and
the parameters used is highly beneficial in debugging connectivity
problems.

Some of the problems have occurred due to subtle implementation
errors. Route damping should never be applied on IBGP learned
routes. To do so can open the possibility for persistent route
loops. When IBGP routes within an AS are inconsistent, route loops
can easily form. Suppressing IBGP learned routes causes such
inconsistencies. Implementations should disallow configuration of
route damping on IBGP peers.

Penalties for instability should only be applied when a route is
removed or replaced and not when a route is added. If damping
parameters are applied consistently, this implementation constraint
will result in a stable secondary path being preferred over an
unstable primary path due to damping of the primary path near the
source.

In topologies where multiple AS paths to a given destination exist
flapping of the primary path can result in suppression of the
secondary path. This can occur if no damping is being done near the
cause of the route flap or if damping is being applied more
aggressively by a distant AS. This problem can be solved in one of
two ways. Damping can be done near the source of the route flap and
the damping parameters can be made consistent. Alternately, a
distant AS which insists on more aggressive damping parameters can
disable penalizing routes on AS path change, penalizing routes only

if they are withdrawn completely. In order to do so, the
implementation must support this option (as described in Section
4.4.3).

Route flap should be damped near the source. Single homed
destinations can be covered by static routes. Aggregation provides
another means of damping. Providers should damp their own internal
problems, however damping on IGP link state origination is not yet
implemented by router vendors. Providers which use multiple AS
within their own topology should damp between their own AS. Providers
should damp adjacent providers AS.

Damping provides a means to limit propagation excessive route change
when connectivity is highly intermittent. Once a problem is
corrected, damping state corresponding to the prefixes known to be
damped due to the problem just fixed can be manually cleared. In
order to determine where damping may have occurred after connectivity
problems, providers should publish their damping parameters.
Providers should be willing to manually clear damping on specific
prefixes or AS paths at the request of other providers when the
request is accompanied by credible assurance that the problem has
truly been addressed.

By damping their own routing information, providers can reduce their
own need to make requests of other providers to clear damping state
after correcting a problem. Providers should be pro-active and
monitor what prefixes and paths are suppressed in addition to
monitoring link states and BGP session state.

Acknowledgements

Thanks also to Rob Coltun (Fore Systems), Sanjay Wadhwa (Fore), John
Scudder (IENG), Eric Bennet (IENG) and Jayesh Bhatt (Bay Networks)
for pointing out errors in the math uncovered during coding of more
recent implementations. These errors appeared in the details of the
implementation suggestion sections written after the first two
implementations were completed. Thanks also to Vern Paxson for a
very thorough review resulting in numerous clarifications to the
document.

References

   [1] Gross, P., and Y. Rekhter, "Application of the border gateway
       protocol in the internet", RFC 1268, October 1991.

   [2] ISO/IEC. Iso/iec 10747 - information technology - telecommuni-
       cations and information exchange between systems - protocol for
       exchange of inter-domain routeing information among intermediate
       systems to support forwarding of iso 8473 pdus. Technical
       report, International Organization for Standardization, August
       1994. ftp://merit.edu/pub/iso/idrp.ps.gz.

   [3] Lougheed, K., and Y. Rekhter, "A border gateway protocol 3 (BGP-
       3)", RFC 1267, October 1991.

   [4] Rekhter, Y., and P. Gross, "Application of the border gateway
       protocol in the internet", RFC 1772, March 1995.

   [5] Rekhter, Y., and T. Li, "A border gateway protocol 4 (BGP-4)",
       RFC 1771, March 1995.

   [6] Rekhter, Y., and C. Topolcic,"Exchanging routing information
       across provider boundaries in the CIDR environment", RFC 1520,
       September 1993.

   [7] Traina, P., "BGP-4 protocol analysis", RFC 1774, March 1995.

   [8] Traina, P., "Experience with the BGP-4 protocol", RFC 1773, March
       1995.

Security Considerations

   The practices outlined in this document do not further weaken the
   security of the routing protocols. Denial of service is possible in
   an already insecure routing environment but these practices only
   contribute to the persistence of such attacks and do not impact the
   methods of prevention and the methods of determining the source.

Authors' Addresses

     Curtis Villamizar
     ANS

     EMail: curtis@ans.net

     Ravi Chandra
     Cisco Systems

     EMail: rchandra@cisco.com

     Ramesh Govindan
     ISI

     EMail: govindan@isi.edu

Full Copyright Statement