Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-idr-route-damp-00

Curtis Villamizar
ANS
Ravi Chandra
Cisco
Ramesh Govindan
ISI
October 30, 1997

BGP Route Flap Damping

Status of this Memo

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:

- to provide a mechanism capable of reducing router processing load caused by instability
- in doing so prevent sustained routing oscillations
- to do so without sacrificing route convergence time for generally well behaved routes.

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

- pack changes into a small number of updates
- preserve consistent routing
- minimal addition space and computational overhead

Excessive updates to reachability state has been widespread in the Internet. This observation was made in the early 1990s by many people involved in Internet operations and remains to case to date. 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

It is necessary to reduce the amount of routing traffic (the number of update message) generated 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.

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 (NLRI) into a smaller number of updates when routes arrive in separate updates.

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 direct 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 loosely 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 is a transitive function.

$$f(f(\epsilon, t_1), t_2) = f(\epsilon, t_1 + t_2)$$

This transitive property allows the decay for a long period to be computed in a single operation regardless of the current value (ϵ). 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.

$$f(\epsilon, n\Delta t) = f(\epsilon, \Delta t)^n = K^n$$

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.

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^2. To avoid N^2 performance, some form of data structure is needed to group routes with common attributes.

Any implementation should packs 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.

An 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 believed to be 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 (ϵ_{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 (ϵ_{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 (λ_{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 (λ_{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 (Δt)

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

reuse list time granularity (Δt_{reuse})

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

reuse list memory (R_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 ($Size_R$)

This is the number of reuse lists. It may be determined from (R_max) or set explicitly.

A necessary 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

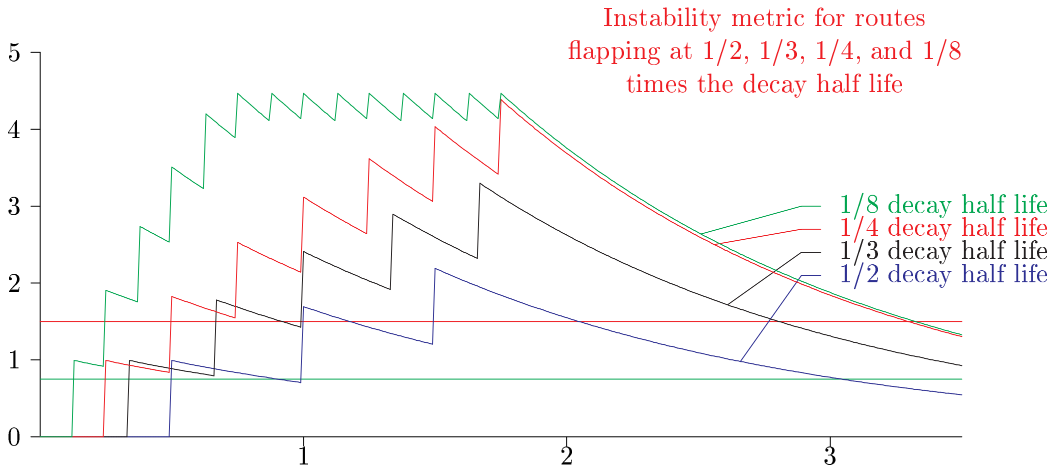

Figure 1: Instability figure of merit for flap at a constant rate

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. 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 ($Size_{R_index}$)

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. If for example, 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. For example, 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 the cutoff and reuse thresholds suggested by the dotted lines, routes would be suppressed after being declared unreachable 2-3 times and be used again after approximately 2 decay half life periods of stability.

From either maximum hold time value ($Tmax_{ok}$ or $Tmax_{ng}$), a ratio of the cutoff to a ceiling can

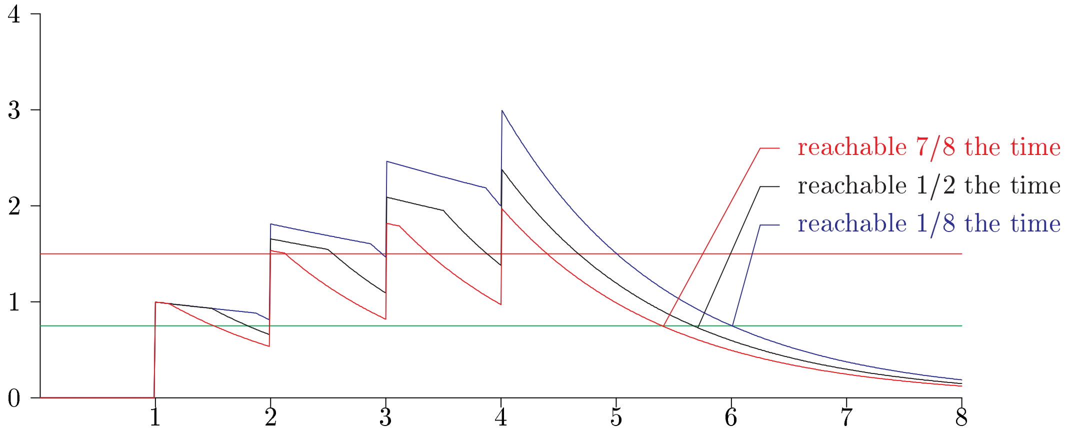

Figure 2: Separate decay constants when unreachable

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.

Figure 2 show 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).

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).

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.

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.

- decay array scale factor ($scale_\delta$)
- cutoff value (ϵ_{cut})
- reuse value (ϵ_{reuse})
- figure of merit ceiling ($\epsilon_{ceiling}$)

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.

- decay rate per tick ($D[1]$)
- decay array size ($Size_D$)

- decay array ($D[\,]$)
- reuse index array size ($Size_{R_{index}}$)
- reuse index array ($R_{index}[\,]$)

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. 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 Data Structures per Route

The following information must be maintained per route. A route here is considered to be a tuple containing at least NLRI prefix, next hop, and AS path. The tuple may also contain other BGP attributes such as MULTI_EXIT_DISCRIMINATOR (MED).

stability figure of merit (ϵ)
: Each route must have a stability figure of merit per applicable parameter set.

last time updated (t_{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.

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.

figure of merit scale factor ($scale_{\epsilon}$)

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 ($D[1]$)

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.

$$D[1] = e^{\left(\frac{1}{\left(\frac{\lambda}{\Delta t}\right)} \log \frac{1}{2} \right)}$$

decay array size ($Size_D$)

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.

$$Size_D = \frac{Tmax}{\Delta t}$$

decay array ($D[\,]$)

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.

$$D[i] = D[1]^i$$

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 ($ratio_{max}$)

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.

$$ratio_{max} = min(\frac{\epsilon_{ceiling}}{\epsilon_{reuse}}, e^{\frac{1}{(\frac{\lambda}{T_{reuse}})}} \log \frac{1}{2})$$

reuse array scale factor ($scale_{reuse}$)

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.

$$scale_{reuse} = \frac{ratio_{max} - 1}{Size_R}$$

reuse index array ($R_{index}[]$)

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.

$$R_{index}[i] = integer \left(\log \frac{\frac{1}{1+((j+1)(ratio_{max}-1))}}{\Delta t_{reuse}} \right)$$

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 array index. If it 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 list.

1. Configuration Parameters
   - $\epsilon_{cut} = 1.25$
   - $\epsilon_{reuse} = 0.5$
   - $T_{hold} = 15min$
   - $\lambda_{ok} = 5min$
   - $\lambda_{ng} = 15min$
   - $T_{max_{ok}}, T_{max_{ng}} = 15, 30min$
2. Implementation Parameters
   - $\Delta t = 1sec$
   - $\Delta t_{reuse} = 15sec$
   - $Size_R = 256$
   - $Size_{reuse\ index\ array} = 1,024$

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 an 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)
   - 900 * integer - decay array (D_{ok})
   - 1,800 * integer - decay array (D_{ng})
   - 120 * pointer - reuse list-heads ($R[\,]$)
   - 2,048 * integer - reuse index arrays ($R_{index}[\,]$)
2. overhead per stable route
   - pointer - containing null entry
3. overhead per unstable route
   - pointer - to a damping structure containing the following
   - integer - figure of merit (ϵ) + bit for state
   - integer - last time updated (t_{last})
   - pointer (optional) to configuration parameter block
   - 2 * pointer - reuse list pointers (prev, next)

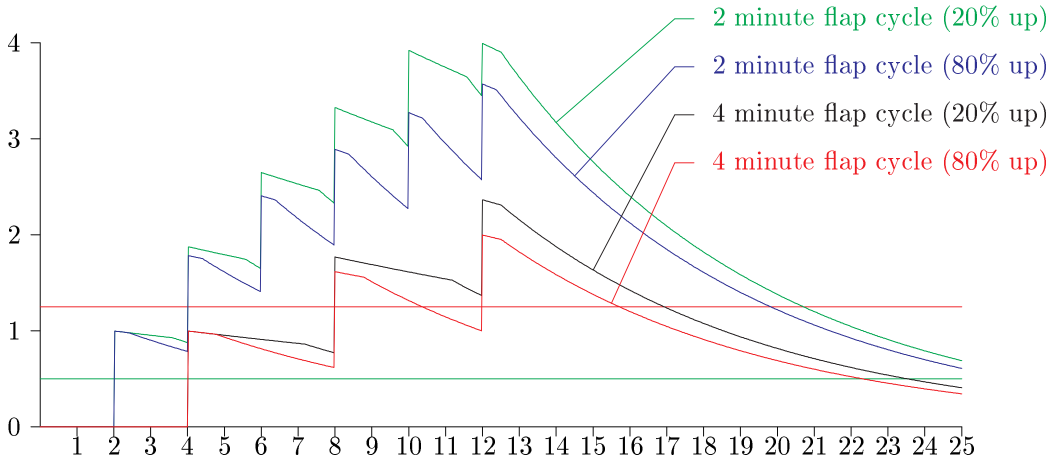

Figure 3: Some fairly long route flap cycles, repeated for 12 minutes, followed by a period of stability.

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)

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. The following two operations are described.

1. Inserting into a reuse list (Section 4.8.6)
2. Reuse list processing every Δt_{reuse} 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 $\epsilon = 1$
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) {
   set $\epsilon = 1$
   } else {
   set $\epsilon = \epsilon * D_{ok}[t_{diff}] + 1$
   if ($\epsilon > ceiling$) {
   set $\epsilon = ceiling$
   }
   }
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 ($D[\ ]$). 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 $\epsilon = 0$
  } else {
    set $\epsilon = \epsilon * D_{ng}[t_{diff}]$
  }
3. if (not suppressed and $\epsilon < \epsilon_{cut}$) {
    use the route
  } else if (suppressed and $\epsilon < \epsilon_{reuse}$) {
    set state to not suppressed
    remove the route from a reuse list if it is on one
    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 ( ε > 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 coute 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 choses 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.

A simple implementation of the circular array of list heads would be an array containing the list heads with an offset. 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 ϵ. The array, scale, and bounds are precomputed to map ϵ to the nearest list head without requiring a logarithm to be computed (see Section 4.5).

1. scale ϵ for the index array lookup producing $index_\epsilon$
2. check $index_\epsilon$ against the array bound
3. if (within the array bound) {
   set $index = index_R[index_\epsilon]$
   } else {
   set $index = Size_R - 1$
   }
4. insert into the list
   $R[modulo_{Size_R}(index + offset)]$

Choosing the correct reuse list involves only a multiply and shift to do the scaling, an integer truncation, then an array lookup. The most common method of implementing a circular array 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 the circular array.

4.8.7 Handling Reuse Timer Events

The granularity of the reuse timer should be more course that 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.

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 Δt_{reuse} seconds.

1. save a pointer to the current zeroth queue head and zero the list head entry
2. set $offset = modulo_{Size_R}(offset + 1)$, thereby rotating the circular queue of list-heads
3. if (the saved list head pointer is non-empty)
   foreach entry {
       set $t_{diff} = t_{now} - t_{updated}$
       set $\epsilon = \epsilon * D_{ok}[t_{diff}]$
       set $t_{updated} = t_{now}$
       if ($\epsilon < \epsilon_{reuse}$)
           reuse the route
       else
           re-insert into another list (see Section 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 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.

Acknowledgements

This work and this document may not have been completed without the advise, comments and encouragement of Yakov Rekhter (Cisco). Dennis Ferguson (MCI) provided a description of the algorithms in the gated BGP implementation and many valuable comments and insights. David Bolen (ANS) and Jordan Becker (ANS) provided valuable comments, particularly regarding early simulations. At the time of this writing two implementations exists. One was led by Ramesh Govindan (ISI) for the NSF Routing Arbiter project. The second was led by Ravi Chandra (Cisco).

Sean Doran (Sprintlink) and Serpil Bayraktar (ANS) were among the early independent testers of the Cisco pre-beta implementation.

References

[1] P. Gross and Y. Rekhter. Application of the border gateway protocol in the internet. Request for Comments (Draft Standard) RFC 1268, Internet Engineering Task Force, October 1991. (Obsoletes RFC1164); (Obsoleted by RFC1655). ftp://ds.internic.net/rfc/rfc1268.txt.

[2] ISO/IEC. Iso/iec 10747 - information technology - telecommunications 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] K. Lougheed and Y. Rekhter. A border gateway protocol 3 (BGP-3). Request for Comments (Draft Standard) RFC 1267, Internet Engineering Task Force, October 1991. (Obsoletes RFC1163). ftp://ds.internic.net/rfc/rfc1267.txt.

[4] Y. Rekhter and P. Gross. Application of the border gateway protocol in the internet. Request for Comments (Draft Standard) RFC 1772, Internet Engineering Task Force, March 1995. (Obsoletes RFC1655). ftp://ds.internic.net/rfc/rfc1772.txt.

[5] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). Request for Comments (Draft Standard) RFC 1771, Internet Engineering Task Force, March 1995. (Obsoletes RFC1654). ftp://ds.internic.net/rfc/rfc1771.txt.

[6] Y. Rekhter and C. Topolcic. Exchanging routing information across provider boundaries in the CIDR environment. Request for Comments (Informational) RFC 1520, Internet Engineering Task Force, September 1993. ftp://ds.internic.net/rfc/rfc1520.txt.

[7] P. Traina. BGP-4 protocol analysis. Request for Comments (Informational) RFC 1774, Internet Engineering Task Force, March 1995. ftp://ds.internic.net/rfc/rfc1774.txt.

[8] P. Traina. Experience with the BGP-4 protocol. Request for Comments (Informational) RFC 1773, Internet Engineering Task Force, March 1995. (Obsoletes RFC1656). ftp://ds.internic.net/rfc/rfc1773.txt.

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.

Author's Addresses

`Curtis Villamizar`

ANS Communications
<curtis@ans.net>

Ravi Chandra
Cisco Systems
<rchandra@cisco.com>

Ramesh Govindan
ISI
<govindan@isi.edu>