Guidelines for Efficient LSA Refreshment in OSPF

draft-ietf-ospf-refresh-guide-01.txt

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress".

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

OSPF, an IGP widely deployed in IP networks today, requires each LSA to be refreshed by the originating router every 30 minutes. This method increases the protocol’s robustness and solves potential problems that may be caused by software bugs, as well as some properties of the protocol itself. Though OSPF expects each LSA to be refreshed independently, ABRs and ASBRs tend to originate Type 3/4/5 LSAs within a short period of time, thus causing periodical network resource exhaustion by LSA refreshments. This document discusses the techniques that can be used to remedy this problem and provide smooth protocol operation. It must be noted that discussed methods can be applied to other link-state routing protocols, such as IS-IS.
1 Introduction

The requirement for each LSA to be refreshed every 30 minutes is really important for OSPF [Ref1]. Not only because OSPF allows LSAs to live no longer than one hour, but also because LSA refreshment provides an automatic method for problem correction.

Possible problems solved by periodical LSA refreshments include those caused by the software bugs and those caused by the features of OSPF itself. This document does not describe the first type of problems, the reader could easily think them out. Examples include lost LSAs, not installed routes, etc. The description of a protocol feature that can potentially cause problems follows.

OSPF uses the Fletcher checksum (see Section 12.1.7 of [Ref1]) to insure that the information transferred and stored in LSAs is not corrupted. But this is not the only function of the LS checksum. It is also used during the LSDB synchronization and LSA flooding processes to understand whether the newly received LSA and an LSA with the same Link State ID and Advertising Router already installed in the LSDB are the same when other parameters (LS Sequence Number and LS Age) do not allow to make the right decision (see Section 13.1 of [Ref1]). Equal values of the LS checksum indicate that LSAs contain identical information, so the received LSA is not processed any further. In the overwhelming majority of cases this really indicates that LSAs are the same and that another copy of already installed LSA is received due to the redundancy of the flooding protocol. However, in some situations, for example when the originating router was reloaded, it is possible that the new LSA (with the same values of the LS ID and Advertising Router fields) will contain different information, but will still have the same checksum as the old one. This is because of the nature of the checksumming process—a small number of bits is used to characterize the contents of a big informational block. The probability of having the same checksum for two different data blocks depends on the mean size of the data blocks, the number of bits used to calculate and store the checksum, and (the last, but definitely not least) the algorithm used to calculate it. Since the probability of this event for OSPF Link State checksum is sufficiently low, the protocol itself relies on the periodical LSA refreshments to solve potential LSDB inconsistency. So, the periodical LSA refreshments insure that the protocol converges within a finite period of time.

The most straightforward approach to refresh LSAs could be to have a single timer firing every 30 minutes and causing reorigination of all self-originated LSAs. This method proved itself as a non-scalable solution causing periodical CPU and link bandwidth exhaustion in the networks. Having a separate timer for each LSA does not mean solving
all problems.

First of all, a router may have a lot of self-originated LSAs in its LSDB, so it may very often be not efficient to keep one timer per each LSA in terms of memory and CPU resources. Second, ABRs tend to originate Type-3 and 4 LSAs within a short period of time. This is caused by the fact that in some implementations Dijkstra is not scheduled every time a change in the LSDB is observed, but only after the LSDB has been quiet for some period of time, preventing excessive CPU load. This may very likely lead to the situation when all intra-area routes are announced in summary-LSAs at once. Even though these originated LSAs have separate timers, these timers fire approximately simultaneously. The same effect can be seen when external routing information is injected into an OSPF domain by an ASBR. If the external routes are available at the moment the ASBR is instructed to advertise them, LSAs will be originated almost at once, causing periodical refreshments to be grouped into big chunks and leading to periodical congestion. Similar logic can be applied to the NSSA ASBRs, NSSA translating ABRs, as well as OSPF routers originating Opaque-LSAs.

The graph show in Figure 1 depicts the number of self-originated LSAs as a function of time. This graph is greatly simplified assuming that the originating router is an ASBR and injects static routing information with stable topology.
At moment t1, the router originates its router-LSA. Then, at t2, the router is instructed to inject external routing information into the OSPF domain. At t3 (t3 - t2 == LSRefreshTime) the router reoriginates the AS-external-LSAs and floods them to its neighbors (reorigination of the router-LSAs is not illustrated). It can be seen that the router introduces periodical traffic bursts into the network, also causing excessive CPU utilization in other OSPF routers.

The customers of such a network can suffer from this behavior, experiencing periodical network congestion and increased response time.

The following sections of this document discuss the methods of implementing OSPF LSA refreshment feature so that it does not disrupt network operation.
2 Consideration of possible approaches

2.1 Dispersing LSA origination

One of the most obvious techniques to avoid LSA refresh grouping could be dispersion of LSA originations. If LSAs were originated with some interval, their refreshes wouldn't group. Though this method seems to be a very easy solution, it has several disadvantages. The most important one is that the time of routing information propagation is increased, i.e., it may take a lot of time (protocol-wise) until the last LSA in the origination queue gets served. However, introducing a minor dispersion in the initial origination process is encouraged.

2.2 Increasing LSA refresh period

Another "easy" way to solve the problem is to increase the refresh interval up to some value near the MaxAge value. This would, on the one hand, decrease the frequency of the updates and, on the other hand, would let the protocol function properly without changes in the aging mechanism. This solution is not ideal, because it does not eliminate periodical LSA refresh bursts, though making them less frequent. This method also increases the time of protocol self-correction in case of problems described in Section 1.

2.3 Avoiding periodical refreshes

Another proposal is to avoid periodic LSA refreshes at all by making all or some subset of LSAs be DoNotAge as in [Ref2]. This approach does solve the refresh burst problem (there are no refreshments at all), but it also decreases the protocol robustness by removing one of the automatic survival features, thus making the protocol less bullet-proof and attractive.

2.4 Further increasing the refresh period using DNA LSAs

The DNA method mentioned above could be used to increase the LSA refresh interval beyond the MaxAge value. This is possible because DNA LSAs are not aged by the routers, so the originating router can refresh its LSAs at whatever rate it chooses. This seems to be a potentially advantageous possibility. However, neither DNA LSAs, nor the refresh interval increase per se can be considered as a sufficient method to prevent the periodical LSA burst problem. More than that, as it was already mentioned, increasing the refresh interval may lead to potentially longer nonoperational periods.

A wide range of methods can be used to implement LSA refreshments efficiently. The following section describes an approach, called
"LSA Refreshment Dispersion", implemented in Zebra routing software, as an example of such a technique.

3 LSA Refreshment Dispersion

Optimization of the LSA refreshment process described here is based on the observation that if the first refreshment interval is randomized rather than set exactly to LSRefreshTime, subsequent LSA refreshes will also be dispersed. If LSAs are refreshed using individual timers, randomization of the first interval for self-originated LSAs prevents LSA refreshment events from grouping and causing network congestion. Individual LSA refreshments are distributed within the LSRefreshTime period evenly.

The described method results in distribution of LSA origination events shown in Figure 2.

```
SelfLSAs()
```

```
  ^
  |
  |
  |
  |
  |
  |
  |
  /
  |
+-------- ---------...-----------|-----> t

LSRefreshTime

|<--------------------------->|

Figure 1. Graph of SelfLSAs(t) when Refreshment Dispersion is used

The effect of the LSA refreshment dispersion is that OSPF does not put high load on the networks, but instead uses network resources evenly, allowing routers to adapt to its traffic and network
administrators to predict necessary resources in more effective manner.

In order to use this technique, the implementation should schedule the LSA refreshment as shown in pseudocode in Figure 3.

```c
if ((LSA_SEQ (lsa) == OSPF_INITIAL_SEQUENCE_NUMBER) &&
    (LSA_AGE (lsa) == 0))
    delay = OSPF_LS_REFRESH_SHIFT +
             (random () % OSPF_LS_REFRESH_TIME);
else {
    delay = OSPF_LS_REFRESH_TIME - LSA_AGE(lsa);
    if (delay < 0) delay = 0;
    /* Add jitter to avoid syncing */
    delay = delay + (random () % OSPF_LS_REFRESH_JITTER) +1;
}
schedule_lsa_refresh (lsa, delay);
```

Figure 3. Pseudocode for refreshment delay calculation

The constants used in the algorithm are defined as follows:

- **OSPF_INITIAL_SEQUENCE_NUMBER** is the InitialSequenceNumber constant as defined in [Ref1]

- **OSPF_LS_REFRESH_TIME** is the LSRefreshTime constant as defined in [Ref1]

- **OSPF_LS_REFRESH_SHIFT** is the length of the silent period, i.e., the minimum time that must elapse after LSA origination before the first LSA refresh.

- **OSPF_LS_REFRESH_JITTER** is the limit for the pseudo-random number added to the LSA refresh interval to avoid synchronization, i.e., to introduce additional minor dispersion.
As it was mentioned before, maintaining a separate timer for each LSA can be costly. A simple way of solving this problem is to group LSAs that must be refreshed approximately at the same time and maintain a single timer for them. This may be implemented by having an LSA accumulation group and a periodic timer. The LSA group is used to list all LSAs originated (or reoriginated) between two consecutive timer expirations (within OSPF_REFRESH_GROUP_TIME seconds). The group is flushed either on the timer expiration or when the size of the group reaches a predefined limit (OSPF_REFRESH_GROUP_LIMIT). When a group is flushed, a refresh event is scheduled as shown in the pseudocode in Figure 3. The event handling routine is supposed to submit the LSAs in the group for reorigination. Note that this is really necessary to limit the maximum number of LSAs in the LSA group and flush the group when this limit is reached in order to achieve acceptable results in the refresh event dispersion.

Even when refresh events are dispersed using the method described above, there can still happen some situations where a big number of LSAs must be refreshed within a short period of time. This may lead to high CPU load in the originating router and protocol traffic bursts in the network. It is possible to avoid these problems if the OSPF implementation maintains an LSA refresh queue, which is served not faster than some predefined number of LSAs per second (OSPF_REFRESH_QUEUE_RATE). Note that the value of the OSPF_REFRESH_QUEUE_RATE should not be very low, because this may lead to the reorigination queue being never served completely and some LSAs remaining unrefreshed for a long period. This may also affect the initial dispersion in the refresh events introduced by randomizing the first refresh interval. The recommended value for the OSPF_REFRESH_QUEUE_RATE constant is several times the OSPF_REFRESH_GROUP_LIMIT per second.

The described algorithm provides desired results in situations when a router receives self-originated LSAs during adjacency establishment after a reload and decides to accept them as correct (and consequently does not originate the new instances). In this case after the received LSAs are identified as self-originated and approved for future use, they are registered in the refreshment module as if they were originated. Such LSAs are also supposed to be bundled into refreshment groups, but in addition to the periodic timer and maximum group size the implementation should care about the ages of the LSAs being grouped. Generally, if the absolute difference between the age of an LSA being added to the group and the first LSA in the group is larger than a certain acceptable value (OSPF_REFRESH_GROUP_AGE_DIF), the group should be unconditionally flushed and the new LSA must start the new group. Note that it is perfectly possible for the received self-originated LSAs to be not
grouped by their age at all when they are transmitted in the database description and link-state update packets. In the worst case each refreshment group will contain a single LSA. However, even in this worst case, LSAs will be grouped by the age again after the first refreshment cycle, so the grouping algorithm presented above is self-optimizing.

Below goes the functional specification of the refreshment algorithm.

1. Each originated (reoriginated) or received self-originated LSA is passed for registration to the LSA refreshment module.

2. The LSA refreshment module bundles the LSAs submitted for registration into refresh groups. The LSAs in one group share the same timer. A group is flushed (see item 3) whenever one of the following three events happens:
   a) the periodical timer expires (every OSPF_REFRESH_GROUP_TIME seconds)
   b) after an LSA has been added to the group, the size of the group reaches OSPF_REFRESH_GROUP_LIMIT
   c) the age of the LSA to be added to the group differs from the age of the first LSA in the group more than OSPF_REFRESH_GROUP_AGE_DIF

3. When a group is flushed, perform the following action:
   a) calculate the delay for the refreshment event as presented in Figure 3
   b) schedule the refreshment event with the delay calculated in the previous step, passing the LSA refresh group as the parameter to the event handling routine
   c) create a new LSA refresh group

4. When a refresh event is processed, put the LSAs in the refresh group associated with the event to the reorigination queue.

5. The reorigination queue is served not faster than OSPF_REFRESH_QUEUE_RATE LSAs per second.

The following table gives recommended values for the constants used in the algorithm description:
Table 1. Recommended values for constants used in algorithm

<table>
<thead>
<tr>
<th>Constant</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSPF_LS_REFRESH_SHIFT</td>
<td>60</td>
</tr>
<tr>
<td>OSPF_LS_REFRESH_JITTER</td>
<td>10</td>
</tr>
<tr>
<td>OSPF_REFRESH_GROUP_TIME</td>
<td>1</td>
</tr>
<tr>
<td>OSPF_REFRESH_GROUP_LIMIT</td>
<td>10</td>
</tr>
<tr>
<td>OSPF_REFRESH_GROUP_AGE_DIF</td>
<td>3</td>
</tr>
<tr>
<td>OSPF_REFRESH_QUEUE_RATE</td>
<td>70</td>
</tr>
</tbody>
</table>

Note that the constants shown above can be implemented as configurable variables. However, the default values should be taken from the above table.

It is also worth noting that presented algorithm can be used along with the extended LSA refresh period described in subsection 2.4 of this document. However, this would require all routers in the network to support DoNotAge LSAs, which may not be the case for some already deployed networks. LSA Refreshment Dispersion per se, in contrast, will possibly require software upgrade only on ASBRs and other routers originating a lot of LSAs.

For more information on the described algorithm, please refer to the source code of Zebra routing software (governed by GNU GPL), available at http://www.zebra.org.

4 Security Considerations

The algorithm specified in this document does not raise any security issues that are not already covered in [Ref1].

5 References


6 Author’s Address

Alex D. Zinin
Cisco Systems
150 West Tasman Dr.
San Jose, CA
95134
E-mail: azinin@cisco.com