Resilient Next-hop Groups

Resilient groups are a type of next-hop group that is aimed at minimizing disruption in flow routing across changes to the group composition and weights of constituent next hops.

The idea behind resilient hashing groups is best explained in contrast to the legacy multipath next-hop group, which uses the hash-threshold algorithm, described in RFC 2992.

To select a next hop, hash-threshold algorithm first assigns a range of hashes to each next hop in the group, and then selects the next hop by comparing the SKB hash with the individual ranges. When a next hop is removed from the group, the ranges are recomputed, which leads to reassignment of parts of hash space from one next hop to another. RFC 2992 illustrates it thus:

```
+-------+-------+-------+-------+-------+
|   1   |   2   |   3   |   4   |   5   |
+-------+-+-----+---+---+-----+-+-------+
|    1    |    2    |    4    |    5    |
+---------+---------+---------+---------+

 Before and after deletion of next hop 3
 under the hash-threshold algorithm.
```

Note how next hop 2 gave up part of the hash space in favor of next hop 1, and 4 in favor of 5. While there will usually be some overlap between the previous and the new distribution, some traffic flows change the next hop that they resolve to.

If a multipath group is used for load-balancing between multiple servers, this hash space reassignment causes an issue that packets from a single flow suddenly end up arriving at a server that does not expect them. This can result in TCP connections being reset.

If a multipath group is used for load-balancing among available paths to the same server, the issue is that different latencies and reordering along the way causes the packets to arrive in the wrong order, resulting in degraded application performance.

To mitigate the above-mentioned flow redirection, resilient next-hop groups insert another layer of indirection between the hash space and its constituent next hops: a hash table. The selection algorithm uses SKB hash to choose a hash table bucket, then reads the next hop that this bucket contains, and forwards traffic there.

This indirection brings an important feature. In the hash-threshold algorithm, the range of hashes associated with a next hop must be continuous. With a hash table, mapping between the hash table buckets and the individual next hops is arbitrary. Therefore when a next hop is deleted the buckets that held it are simply reassigned to other next hops:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|1|1|1|2|2|2|2|3|3|3|3|4|4|4|4|5|5|5|5|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                v v v v
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|1|1|1|2|2|2|2|1|2|4|5|4|4|4|4|5|5|5|5|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Before and after deletion of next hop 3
under the resilient hashing algorithm.
```

When weights of next hops in a group are altered, it may be possible to choose a subset of buckets that are currently not used for forwarding traffic, and use those to satisfy the new next-hop distribution demands, keeping the "busy" buckets intact. This way, established flows are ideally kept being forwarded to the same endpoints through the same paths as before the next-hop group change.

Algorithm

In a nutshell, the algorithm works as follows. Each next hop deserves a certain number of buckets, according to its weight and the number of buckets in the hash table. In accordance with the source code, we will call this number a "wants count" of a next hop. In case of an event that might cause bucket allocation change, the wants counts for individual next hops are updated.

Next hops that have fewer buckets than their wants count, are called "underweight". Those that have more are "overweight". If there are no overweight (and therefore no underweight) next hops in the group, it is said to be "balanced".

Each bucket maintains a last-used timer. Every time a packet is forwarded through a bucket, this timer is updated to current jiffies value. One attribute of a resilient group is then the "idle timer", which is the amount of time that a bucket must not be hit by traffic in order for it to be considered "idle". Buckets that are not idle are busy.

After assigning wants counts to next hops, an "upkeep" algorithm runs. For buckets:

1. that have no assigned next hop, or
2. whose next hop has been removed, or
3. that are idle and their next hop is overweight,

upkeep changes the next hop that the bucket references to one of the underweight next hops. If, after considering all buckets in this manner, there are still underweight next hops, another upkeep run is scheduled to a future time.

There may not be enough "idle" buckets to satisfy the updated wants counts of all next hops. Another attribute of a resilient group is the "unbalanced timer". This timer can be set to 0, in which case the table will stay out of balance until idle buckets do appear, possibly never. If set to a non-zero value, the value represents the period of time that the table is permitted to stay out of balance.

With this in mind, we update the above list of conditions with one more item. Thus buckets:

4. whose next hop is overweight, and the amount of time that the table has been out of balance exceeds the unbalanced timer, if that is non-zero,

... are migrated as well.

Offloading & Driver Feedback

When offloading resilient groups, the algorithm that distributes buckets among next hops is still the one in SW. Drivers are notified of updates to next hop groups in the following three ways:

- Full group notification with the type `NH_NOTIFIER_INFO_TYPE_RES_TABLE`. This is used just after the group is created and buckets populated for the first time.
- Single-bucket notifications of the type `NH_NOTIFIER_INFO_TYPE_RES_BUCKET`, which is used for notifications of individual migrations within an already-established group.
- Pre-replace notification, `NEXTHOP_EVENT_RES_TABLE_PRE_REPLACE`. This is sent before the group is replaced, and is a way for the driver to veto the group before committing anything to the HW.

Some single-bucket notifications are forced, as indicated by the "force" flag in the notification. Those are used for the cases where e.g. the next hop associated with the bucket was removed, and the bucket really must be migrated.

Non-forced notifications can be overridden by the driver by returning an error code. The use case for this is that the driver notifies the HW that a bucket should be migrated, but the HW discovers that the bucket has in fact been hit by traffic.

A second way for the HW to report that a bucket is busy is through the `nexthop_res_grp_activity_update()` API. The buckets identified this way as busy are treated as if traffic hit them.

Offloaded buckets should be flagged as either "offload" or "trap". This is done through the `nexthop_bucket_set_hw_flags()` API.

Netlink UAPI

Resilient Group Replacement

Resilient groups are configured using the `RTM_NEWNEXTHOP` message in the same manner as other multipath groups. The following changes apply to the attributes passed in the netlink message:

`NHA_GROUP_TYPE`	Should be `NEXTHOP_GRP_TYPE_RES` for resilient group.
`NHA_RES_GROUP`	A nest that contains attributes specific to resilient groups.

`NHA_RES_GROUP` payload:

`NHA_RES_GROUP_BUCKETS`	Number of buckets in the hash table.
`NHA_RES_GROUP_IDLE_TIMER`	Idle timer in units of clock_t.
`NHA_RES_GROUP_UNBALANCED_TIMER`	Unbalanced timer in units of clock_t.

Next Hop Get

Requests to get resilient next-hop groups use the `RTM_GETNEXTHOP` message in exactly the same way as other next hop get requests. The response attributes match the replacement attributes cited above, except `NHA_RES_GROUP` payload will include the following attribute:

`NHA_RES_GROUP_UNBALANCED_TIME`	How long has the resilient group been out of balance, in units of clock_t.

Bucket Get

The message `RTM_GETNEXTHOPBUCKET` without the `NLM_F_DUMP` flag is used to request a single bucket. The attributes recognized at get requests are:

`NHA_ID`	ID of the next-hop group that the bucket belongs to.
`NHA_RES_BUCKET`	A nest that contains attributes specific to bucket.

`NHA_RES_BUCKET` payload:

`NHA_RES_BUCKET_INDEX`	Index of bucket in the resilient table.

Bucket Dumps

The message `RTM_GETNEXTHOPBUCKET` with the `NLM_F_DUMP` flag is used to request a dump of matching buckets. The attributes recognized at dump requests are:

`NHA_ID`	If specified, limits the dump to just the next-hop group with this ID.
`NHA_OIF`	If specified, limits the dump to buckets that contain next hops that use the device with this ifindex.
`NHA_MASTER`	If specified, limits the dump to buckets that contain next hops that use a device in the VRF with this ifindex.
`NHA_RES_BUCKET`	A nest that contains attributes specific to bucket.

`NHA_RES_BUCKET` payload:

`NHA_RES_BUCKET_NH_ID`	If specified, limits the dump to just the buckets that contain the next hop with this ID.

Usage

To illustrate the usage, consider the following commands:

```
# ip nexthop add id 1 via 192.0.2.2 dev eth0
# ip nexthop add id 2 via 192.0.2.3 dev eth0
# ip nexthop add id 10 group 1/2 type resilient \
        buckets 8 idle_timer 60 unbalanced_timer 300
```

The last command creates a resilient next-hop group. It will have 8 buckets (which is unusually low number, and used here for demonstration purposes only), each bucket will be considered idle when no traffic hits it for at least 60 seconds, and if the table remains out of balance for 300 seconds, it will be forcefully brought into balance.

Changing next-hop weights leads to change in bucket allocation:

```
# ip nexthop replace id 10 group 1,3/2 type resilient
```

This can be confirmed by looking at individual buckets:

```
# ip nexthop bucket show id 10
id 10 index 0 idle_time 5.59 nhid 1
id 10 index 1 idle_time 5.59 nhid 1
id 10 index 2 idle_time 8.74 nhid 2
id 10 index 3 idle_time 8.74 nhid 2
id 10 index 4 idle_time 8.74 nhid 1
id 10 index 5 idle_time 8.74 nhid 1
id 10 index 6 idle_time 8.74 nhid 1
id 10 index 7 idle_time 8.74 nhid 1
```

Note the two buckets that have a shorter idle time. Those are the ones that were migrated after the next-hop replace command to satisfy the new demand that next hop 1 be given 6 buckets instead of 4.

Netdevsim

The netdevsim driver implements a mock offload of resilient groups, and exposes debugfs interface that allows marking individual buckets as busy. For example, the following will mark bucket 23 in next-hop group 10 as active:

```
# echo 10 23 > /sys/kernel/debug/netdevsim/netdevsim10/fib/nexthop_bucket_activity
```

In addition, another debugfs interface can be used to configure that the next attempt to migrate a bucket should fail:

```
# echo 1 > /sys/kernel/debug/netdevsim/netdevsim10/fib/fail_nexthop_bucket_replace
```

Besides serving as an example, the interfaces that netdevsim exposes are useful in automated testing, and `tools/testing/selftests/drivers/net/netdevsim/nexthop.sh` makes use of them to test the algorithm.