Generic networking statistics for netlink users
==========

Statistic counters are grouped into structs:

Struct	TLV type	Description
gnet_stats_basic	TCA_STATS_BASIC	Basic statistics
gnet_stats_rate_est	TCA_STATS_RATE_EST	Rate estimator
gnet_stats_queue	TCA_STATS_QUEUE	Queue statistics
none	TCA_STATS_APP	Application specific

Collecting:

Declare the statistic structs you need:

```
struct mystruct {
        struct gnet_stats_basic bstats;
        struct gnet_stats_queue qstats;
        ...
};
```

Update statistics:

```
mystruct->tstats.packet++;
mystruct->qstats.backlog += skb->pkt_len;
```

Export to userspace (Dump):

```
my_dumping_routine(struct sk_buff *skb, ...)
{
        struct gnet_dump dump;

        if (gnet_stats_start_copy(skb, TCA_STATS2, &mystruct->lock, &dump) < 0)
                goto rtattr_failure;

        if (gnet_stats_copy_basic(&dump, &mystruct->bstats) < 0 ||
            gnet_stats_copy_queue(&dump, &mystruct->qstats) < 0 ||
                gnet_stats_copy_app(&dump, &xstats, sizeof(xstats)) < 0)
                goto rtattr_failure;

        if (gnet_stats_finish_copy(&dump) < 0)
                goto rtattr_failure;
        ...
}
```

TCA_STATS/TCA_XSTATS backward compatibility:

Prior users of struct tc_stats and xstats can maintain backward
compatibility by calling the compat wrappers to keep providing the
existing TLV types.

```
my_dumping_routine(struct sk_buff *skb, ...)
```

```
{
    if (gnet_stats_start_copy_compat(skb, TCA_STATS2, TCA_STATS,
                TCA_XSTATS, &mystruct->lock, &dump) < 0)
                goto rtattr_failure;
        ...
}
```

A struct tc_stats will be filled out during gnet_stats_copy_* calls
and appended to the skb. TCA_XSTATS is provided if gnet_stats_copy_app
was called.

Locking:

Locks are taken before writing and released once all statistics have
been written. Locks are always released in case of an error. You
are responsible for making sure that the lock is initialized.

Rate Estimator:
----

0) Prepare an estimator attribute. Most likely this would be in user
   space. The value of this TLV should contain a tc_estimator structure.
   As usual, such a TLV needs to be 32 bit aligned and therefore the
   length needs to be appropriately set, etc. The estimator interval
   and ewma log need to be converted to the appropriate values.
   tc_estimator.c::tc_setup_estimator() is advisable to be used as the
   conversion routine. It does a few clever things. It takes a time
   interval in microsecs, a time constant also in microsecs and a struct
   tc_estimator to be populated. The returned tc_estimator can be
   transported to the kernel. Transfer such a structure in a TLV of type
   TCA_RATE to your code in the kernel.

In the kernel when setting up:
1) make sure you have basic stats and rate stats setup first.
2) make sure you have initialized stats lock that is used to setup such
   stats.
3) Now initialize a new estimator:

```
   int ret = gen_new_estimator(my_basicstats,my_rate_est_stats,
       mystats_lock, attr_with_tcestimator_struct);

   if ret == 0
       success
   else
       failed
```

From now on, every time you dump my_rate_est_stats it will contain
up-to-date info.

Once you are done, call gen_kill_estimator(my_basicstats,
my_rate_est_stats) Make sure that my_basicstats and my_rate_est_stats
are still valid (i.e still exist) at the time of making this call.

Authors:

Thomas Graf <tgraf@suug.ch>
Jamal Hadi Salim <hadi@cyberus.ca>