NAME

ALTQ - kernel interfaces for manipulating output queues on network interfaces

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_var.h>
```

Enqueue macros

IFQ_ENQUEUE(*struct ifaltq *ifq, struct mbuf *m, int error*);

IFQ_HANDOFF(*struct ifnet *ifp, struct mbuf *m, int error*);

IFQ_HANDOFF_ADJ(*struct ifnet *ifp, struct mbuf *m, int adjust, int error*);

Dequeue macros

IFQ_DEQUEUE(*struct ifaltq *ifq, struct mbuf *m*);

IFQ_POLL_NOLOCK(*struct ifaltq *ifq, struct mbuf *m*);

IFQ_PURGE(*struct ifaltq *ifq*);

IFQ_IS_EMPTY(*struct ifaltq *ifq*);

Driver managed dequeue macros

IFQ_DRV_DEQUEUE(*struct ifaltq *ifq, struct mbuf *m*);

IFQ_DRV_PREPEND(*struct ifaltq *ifq, struct mbuf *m*);

IFQ_DRV_PURGE(*struct ifaltq *ifq*);

IFQ_DRV_IS_EMPTY(*struct ifaltq *ifq*);

General setup macros

IFQ_SET_MAXLEN(*struct ifaltq *ifq, int len*);

IFQ_INC_LEN(*struct ifaltq *ifq*);

IFQ_DEC_LEN(*struct ifaltq *ifq*);

IFQ_INC_DROPS(*struct ifaltq *ifq*);

IFQ_SET_READY(*struct ifaltq *ifq*);

DESCRIPTION

The **ALTQ** system is a framework to manage queuing disciplines on network interfaces. **ALTQ** introduces new macros to manipulate output queues. The output queue macros are used to abstract queue operations and not to touch the internal fields of the output queue structure. The macros are independent from the **ALTQ** implementation, and compatible with the traditional *ifqueue* macros for ease of transition.

IFQ_ENQUEUE(), **IFQ_HANDOFF**() and **IFQ_HANDOFF_ADJ**() enqueue a packet *m* to the queue *ifq*. The underlying queuing discipline may discard the packet. The *error* argument is set to 0 on success, or ENOBUFS if the packet is discarded. The packet pointed to by *m* will be freed by the device driver on success, or by the queuing discipline on failure, so the caller should not touch *m* after enqueuing. **IFQ_HANDOFF**() and **IFQ_HANDOFF_ADJ**() combine the enqueue operation with statistic generation and call **if_start**() upon successful enqueue to initiate the actual send.

IFQ_DEQUEUE() dequeues a packet from the queue. The dequeued packet is returned in *m*, or *m* is set to NULL if no packet is dequeued. The caller must always check *m* since a non-empty queue could return NULL under rate-limiting.

IFQ_POLL_NOLOCK() returns the next packet without removing it from the queue. The caller must hold the queue mutex when calling **IFQ_POLL_NOLOCK**() in order to guarantee that a subsequent call to **IFQ_DEQUEUE_NOLOCK**() dequeues the same packet.

IFQ_*_NOLOCK() variants (if available) always assume that the caller holds the queue mutex. They can be grabbed with **IFQ_LOCK**() and released with **IFQ_UNLOCK**().

IFQ_PURGE() discards all the packets in the queue. The purge operation is needed since a non-work conserving queue cannot be emptied by a dequeue loop.

IFQ_IS_EMPTY() can be used to check if the queue is empty. Note that **IFQ_DEQUEUE**() could still return NULL if the queuing discipline is non-work conserving.

IFQ_DRV_DEQUEUE() moves up to *ifq->ifq_drv_maxlen* packets from the queue to the "driver managed" queue and returns the first one via *m*. As for **IFQ_DEQUEUE**(), *m* can be NULL even for a non-empty queue. Subsequent calls to **IFQ_DRV_DEQUEUE**() pass the packets from the "driver managed" queue without obtaining the queue mutex. It is the responsibility of the caller to protect against concurrent access. Enabling **ALTQ** for a given queue sets *ifq_drv_maxlen* to 0 as the "bulk

dequeue" performed by **IFQ_DRV_DEQUEUE**() for higher values of *ifq_drv_maxlen* is adverse to **ALTQ**'s internal timing. Note that a driver must not mix **IFQ_DRV_*** () macros with the default dequeue macros as the default macros do not look at the "driver managed" queue which might lead to an mbuf leak.

IFQ_DRV_PREPEND() prepends *m* to the "driver managed" queue from where it will be obtained with the next call to **IFQ_DRV_DEQUEUE**().

IFQ_DRV_PURGE() flushes all packets in the "driver managed" queue and calls to **IFQ_PURGE**() afterwards.

IFQ_DRV_IS_EMPTY() checks for packets in the "driver managed" part of the queue. If it is empty, it forwards to **IFQ_IS_EMPTY**().

IFQ_SET_MAXLEN() sets the queue length limit to the default FIFO queue. The *ifq_drv_maxlen* member of the *ifaltq* structure controls the length limit of the "driver managed" queue.

IFQ_INC_LEN() and **IFQ_DEC_LEN**() increment or decrement the current queue length in packets. This is mostly for internal purposes.

IFQ_INC_DROPS() increments the drop counter and is identical to **IF_DROP**(). It is defined for naming consistency only.

IFQ_SET_READY() sets a flag to indicate that a driver was converted to use the new macros. **ALTQ** can be enabled only on interfaces with this flag.

COMPATIBILITY

ifaltq structure

In order to keep compatibility with the existing code, the new output queue structure *ifaltq* has the same fields. The traditional **IF_*** () macros and the code directly referencing the fields within *if_snd* still work with *ifaltq*.

```
        ##old-style##                    ##new-style##
                                |
 struct ifqueue {               | struct ifaltq {
   struct mbuf *ifq_head;       |   struct mbuf *ifq_head;
   struct mbuf *ifq_tail;       |   struct mbuf *ifq_tail;
   int         ifq_len;         |   int         ifq_len;
   int         ifq_maxlen;      |   int         ifq_maxlen;
 };                             |   /* driver queue fields */
```

```
                        |   ......
                        |   /* altq related fields */
                        |   ......
                        | };
                        |
```

The new structure replaces *struct ifqueue* in *struct ifnet*.

```
        ##old-style##                   ##new-style##
                        |
 struct ifnet {                 | struct ifnet {
    ....                        |    ....
                                |
    struct ifqueue if_snd;      |    struct ifaltq if_snd;
                                |
    ....                        |    ....
 };                             | };
                                |
```

The (simplified) new **IFQ_*****() macros look like:

```
        #define IFQ_DEQUEUE(ifq, m)                             \
                if (ALTQ_IS_ENABLED((ifq))                      \
                        ALTQ_DEQUEUE((ifq), (m));               \
                else                                            \
                        IF_DEQUEUE((ifq), (m));
```

Enqueue operation

The semantics of the enqueue operation is changed. In the new style, enqueue and packet drop are combined since they cannot be easily separated in many queuing disciplines. The new enqueue operation corresponds to the following macro that is written with the old macros.

```
#define    IFQ_ENQUEUE(ifq, m, error)                   \
do {                                            \
      if (IF_QFULL((ifq))) {                        \
            m_freem((m));                       \
            (error) = ENOBUFS;                  \
            IF_DROP(ifq);                       \
      } else {                                  \
            IF_ENQUEUE((ifq), (m));             \
            (error) = 0;                        \
      }                                         \
```

} while (0)

IFQ_ENQUEUE() does the following:

- queue a packet,
- drop (and free) a packet if the enqueue operation fails.

If the enqueue operation fails, *error* is set to ENOBUFS. The *m* mbuf is freed by the queuing discipline. The caller should not touch mbuf after calling **IFQ_ENQUEUE**() so that the caller may need to copy *m_pkthdr.len* or *m_flags* field beforehand for statistics. **IFQ_HANDOFF**() and **IFQ_HANDOFF_ADJ**() can be used if only default interface statistics and an immediate call to **if_start**() are desired. The caller should not use **senderr**() since mbuf was already freed.

The new style **if_output**() looks as follows:

```
        ##old-style##                    ##new-style##
                              |
int                           | int
ether_output(ifp, m0, dst, rt0)     | ether_output(ifp, m0, dst, rt0)
{                             | {
    ......                    |     ......
                              |
                              |     mflags = m->m_flags;
                              |     len = m->m_pkthdr.len;
    s = splimp();             |     s = splimp();
    if (IF_QFULL(&ifp->if_snd)) {     |     IFQ_ENQUEUE(&ifp->if_snd, m,
                              |                 error);
        IF_DROP(&ifp->if_snd);      |     if (error != 0) {
        splx(s);              |         splx(s);
        senderr(ENOBUFS);     |         return (error);
    }                         |     }
    IF_ENQUEUE(&ifp->if_snd, m);      |
    ifp->if_obytes +=         |     ifp->if_obytes += len;
              m->m_pkthdr.len;  |
    if (m->m_flags & M_MCAST)     |     if (mflags & M_MCAST)
        ifp->if_omcasts++;    |         ifp->if_omcasts++;
                              |
    if ((ifp->if_flags & IFF_OACTIVE) |     if ((ifp->if_flags & IFF_OACTIVE)
        == 0)                 |         == 0)
        (*ifp->if_start)(ifp);    |         (*ifp->if_start)(ifp);
```

```
    splx(s);                    |    splx(s);
    return (error);             |    return (error);
                                |
 bad:                           | bad:
    if (m)                      |    if (m)
        m_freem(m);             |        m_freem(m);
    return (error);             |    return (error);
 }                              | }
                                |
```

HOW TO CONVERT THE EXISTING DRIVERS

First, make sure the corresponding **if_output**() is already converted to the new style.

Look for *if_snd* in the driver. Probably, you need to make changes to the lines that include *if_snd*.

Empty check operation

If the code checks *ifq_head* to see whether the queue is empty or not, use **IFQ_IS_EMPTY**().

```
        ##old-style##                   ##new-style##
                                |
 if (ifp->if_snd.ifq_head != NULL)    | if (!IFQ_IS_EMPTY(&ifp->if_snd))
                                |
```

IFQ_IS_EMPTY() only checks if there is any packet stored in the queue. Note that even when **IFQ_IS_EMPTY**() is FALSE, **IFQ_DEQUEUE**() could still return NULL if the queue is under rate-limiting.

Dequeue operation

Replace **IF_DEQUEUE**() by **IFQ_DEQUEUE**(). Always check whether the dequeued mbuf is NULL or not. Note that even when **IFQ_IS_EMPTY**() is FALSE, **IFQ_DEQUEUE**() could return NULL due to rate-limiting.

```
        ##old-style##                   ##new-style##
                                |
 IF_DEQUEUE(&ifp->if_snd, m);        | IFQ_DEQUEUE(&ifp->if_snd, m);
                                | if (m == NULL)
                                |     return;
                                |
```

A driver is supposed to call **if_start**() from transmission complete interrupts in order to trigger the next dequeue.

Poll-and-dequeue operation

If the code polls the packet at the head of the queue and actually uses the packet before dequeuing it, use **IFQ_POLL_NOLOCK**() and **IFQ_DEQUEUE_NOLOCK**().

```
        ##old-style##                    ##new-style##
                                |
                                | IFQ_LOCK(&ifp->if_snd);
m = ifp->if_snd.ifq_head;       | IFQ_POLL_NOLOCK(&ifp->if_snd, m);
if (m != NULL) {                | if (m != NULL) {
                                |
  /* use m to get resources */  |   /* use m to get resources */
  if (something goes wrong)     |   if (something goes wrong)
                                |       IFQ_UNLOCK(&ifp->if_snd);
    return;                     |       return;
                                |
  IF_DEQUEUE(&ifp->if_snd, m);  |   IFQ_DEQUEUE_NOLOCK(&ifp->if_snd, m);
                                |   IFQ_UNLOCK(&ifp->if_snd);
                                |
  /* kick the hardware */       |   /* kick the hardware */
}                               | }
                                |
```

It is guaranteed that **IFQ_DEQUEUE_NOLOCK**() under the same lock as a previous **IFQ_POLL_NOLOCK**() returns the same packet. Note that they need to be guarded by **IFQ_LOCK**().

Eliminating IF_PREPEND()

If the code uses **IF_PREPEND**(), you have to eliminate it unless you can use a "driver managed" queue which allows the use of **IFQ_DRV_PREPEND**() as a substitute. A common usage of **IF_PREPEND**() is to cancel the previous dequeue operation. You have to convert the logic into poll-and-dequeue.

```
        ##old-style##                    ##new-style##
                                |
                                | IFQ_LOCK(&ifp->if_snd);
IF_DEQUEUE(&ifp->if_snd, m);    | IFQ_POLL_NOLOCK(&ifp->if_snd, m);
if (m != NULL) {                | if (m != NULL) {
                                |
  if (something_goes_wrong) {   |   if (something_goes_wrong) {
    IF_PREPEND(&ifp->if_snd, m);|       IFQ_UNLOCK(&ifp->if_snd);
    return;                     |       return;
  }                             |   }
                                |
```

```
                        |   /* at this point, the driver
                        |    * is committed to send this
                        |    * packet.
                        |    */
                        |   IFQ_DEQUEUE_NOLOCK(&ifp->if_snd, m);
                        |   IFQ_UNLOCK(&ifp->if_snd);
                        |
   /* kick the hardware */      |   /* kick the hardware */
 }                      | }
                        |
```

Purge operation

Use **IFQ_PURGE**() to empty the queue. Note that a non-work conserving queue cannot be emptied by a dequeue loop.

```
        ##old-style##                   ##new-style##
                        |
 while (ifp->if_snd.ifq_head != NULL) {|  IFQ_PURGE(&ifp->if_snd);
   IF_DEQUEUE(&ifp->if_snd, m);     |
   m_freem(m);                  |
 }                      |
                        |
```

Conversion using a driver managed queue

Convert **IF_*****()** macros to their equivalent **IFQ_DRV_*****()** and employ **IFQ_DRV_IS_EMPTY**() where appropriate.

```
        ##old-style##                   ##new-style##
                        |
 if (ifp->if_snd.ifq_head != NULL)    | if (!IFQ_DRV_IS_EMPTY(&ifp->if_snd))
                        |
```

Make sure that calls to **IFQ_DRV_DEQUEUE**(), **IFQ_DRV_PREPEND**() and **IFQ_DRV_PURGE**() are protected with a mutex of some kind.

Attach routine

Use **IFQ_SET_MAXLEN**() to set *ifq_maxlen* to *len*. Initialize *ifq_drv_maxlen* with a sensible value if you plan to use the **IFQ_DRV_*****()** macros. Add **IFQ_SET_READY**() to show this driver is converted to the new style. (This is used to distinguish new-style drivers.)

```
        ##old-style##                   ##new-style##
```

```
                          |
ifp->if_snd.ifq_maxlen = qsize;     | IFQ_SET_MAXLEN(&ifp->if_snd, qsize);
                          | ifp->if_snd.ifq_drv_maxlen = qsize;
                          | IFQ_SET_READY(&ifp->if_snd);
if_attach(ifp);                   | if_attach(ifp);
                          |
```

Other issues

The new macros for statistics:

```
        ##old-style##                  ##new-style##
                          |
IF_DROP(&ifp->if_snd);            | IFQ_INC_DROPS(&ifp->if_snd);
                          |
ifp->if_snd.ifq_len++;            | IFQ_INC_LEN(&ifp->if_snd);
                          |
ifp->if_snd.ifq_len--;            | IFQ_DEC_LEN(&ifp->if_snd);
                          |
```

QUEUING DISCIPLINES

Queuing disciplines need to maintain *ifq_len* (used by **IFQ_IS_EMPTY**()). Queuing disciplines also need to guarantee that the same mbuf is returned if **IFQ_DEQUEUE**() is called immediately after **IFQ_POLL**().

SEE ALSO

pf(4), pf.conf(5), pfctl(8)

HISTORY

The **ALTQ** system first appeared in March 1997 and found home in the KAME project (https://www.kame.net). It was imported to FreeBSD in 5.3 .