8 NS2 Implementation

In this section we present some extensions and changes brought to the implementation of the MAC layer described in the draft IEEE802.11e/D12.0 and realized using *Network Simulator 2*. We will first describe the basic functionalities of the MAC layer and how it works, and then provide the specifics of the new scheduler interface and how it has been implemented. The MAC layer calls functions of the scheduler layer, but the contrary is not true; this allows the MAC layer to work with different types of scheduler. The main changes brought to legacy 802.11e Ns2 implementation are:

- *Common scheduler interface* - A flexible interface has been devised, such that it can be used to develop different types of scheduler without modifying the MAC layer code.
- *New frames support* - The previous version has been extended to support other types of frame. The new frames are:
  - QoS Data
  - QoS CF-Poll
  - QoS CF-Ack
  - QoS Data + QoS CF-Ack
  - QoS Data + QoS CF-Poll
  - QoS CF-Ack + QoS CF-Poll
  - QoS Data + QoS CF-Ack + QoS CF-Poll.
- *Piggybacking capability* - The piggybacking behavior can be enabled or disabled by setting the *piggyback* bit from TCL. Turning off the piggybacking feature means that it is possible to transmit only the frames: QoS Data, QoS CF-Ack and QoS CF-Poll. Even if this bit can assume a different value for each QSTA and QAP, it should be set to the same value for all the QoS stations in the system in order to guarantee the correct behavior of the MAC layer.
- *Qack capability* - Also the Qack feature is managed completely at the MAC layer and it can be disabled or enabled by setting the *qack bit* from TCL. It is possible to communicate to the QAP a value of the *Qack* bit different for each QSTA.
- *ACK policies support* - There are four possible ACK policies: NO ACK, BLOCK ACK, NORMAL ACK, NO EXPLICIT ACK. All these policies, but the BLOCK ACK one have been implemented at the MAC layer. The NO EXPLICIT policy can be used only at the MAC layer, but NO ACK and NORMAL ACK can be used also by the scheduler to enable or disable the ACK feedback.

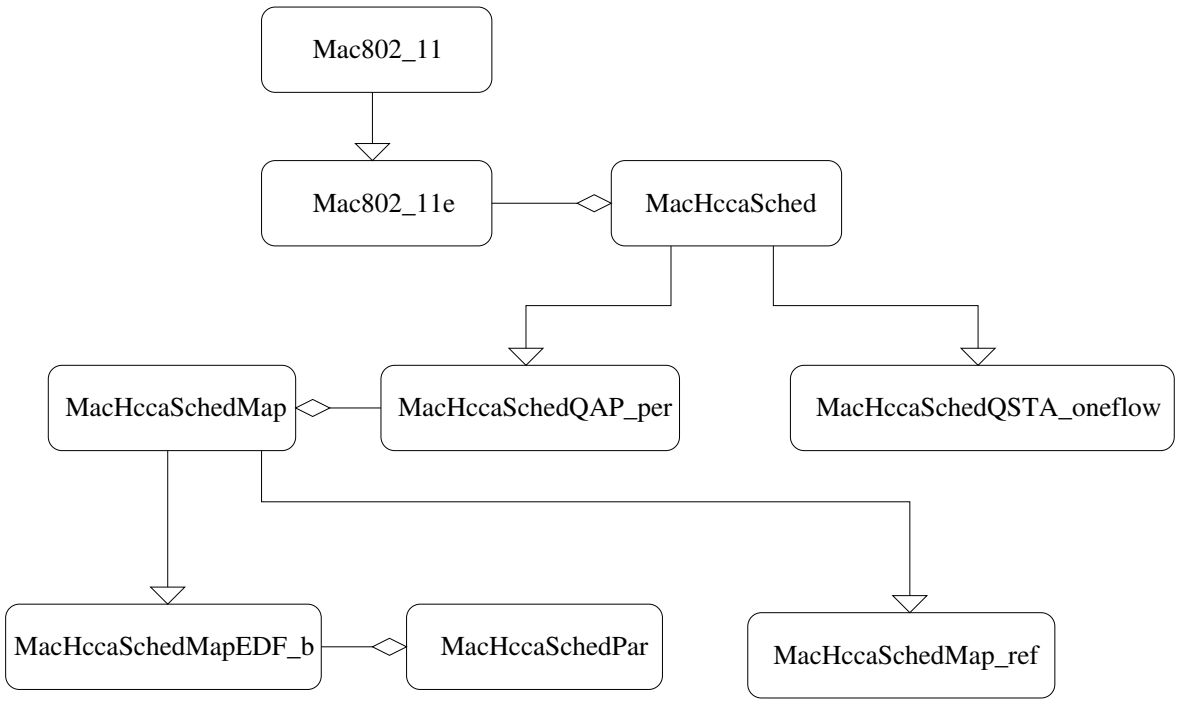

Figure 63: Ns2 – Class Diagram

8.1 Class Diagram

We present now a brief description of the classes that we have extended in order to implement the EDF-B algorithm:

- **Mac802_11** - The functionalities of the legacy MAC layer are implemented by the class Mac802_11. We have extended this class so that it can manage also the new QoS types. A more detailed list of the changes brought to this class is provided in section 8.2.
- **Mac802_11e** - The class is used to extend the legacy MAC layer in order to implement the HCCA function used by the EDF-B scheduler. A complete description of the HCCA features is provided in section 8.3.
- **MacHccaSched** - We have realized a common interface containing all the functions that should be implemented by any scheduler. This way, we have isolated the scheduler and the MAC layers. A class, which represents a scheduler, should be derived from this class and should implement its virtual methods complying with the specifics in section 8.4.
- **MacHccaSchedQSTA_oneflow** - This class represents the scheduler used by the QSTAs. It is called *oneflow* because it manages at most one TS per direction. This choice permits to focus on the performance of the QAP scheduler.
- **MacHccaSchedQAP_periodic** - Both the implemented QAP schedulers exploit an offline builded schedule to serve the TSs at run-time. The online func-

tionalities of the two schedulers are exactly the same, once they are provided with a structure representing the offline schedule. This class is therefore used to implement those common functionalities that are required at run-time. It uses the offline schedule produced by the class *MacHccaSchedMap* to serve the TSs.

- **MacHccaSchedMap** - This class provides the *MacHccaSchedQAP_periodic* class with an interface to a data structure that may represent any periodic offline schedule.

- **MacHccaSchedMapEDF_B** - A list of TXOPs similar to the one produced by the reference scheduler. EDF-B needs some scheduler parameters, derived from the TSPEC elements, to build the offline schedule; therefore *MacHccaSchedMapEDF_B* needs the class *MacHccaSchedPar*, that perform the mapping procedure, to retrieve those parameters.

- **MacHccaSchedMap_ref** - This class specializes *MacHccaSchedMap* to produce the offline schedule needed by the reference scheduler. The main data structure is a list of TXOPs whose elements specify the start time and duration of those TXOPs.

- **MacHccaSchedPar** - This class is used to perform the parameters mapping procedure needed by EDF-B. We have not included this procedure directly in the class *MacHccaSchedMapEDF_B* because the offline schedule and the parameters mapping are independent.

8.2 Class Mac802_11 - Legacy Mac802.11 Changes

The following changes have been brought to the legacy IEEE 802.11 MAC layer to make it compatible with the new QoS extensions:

- **QoS frame introduction** - The following frame types have been added:
  - MAC_Subtype_QoS_Data
  - MAC_Subtype_QoS_Null
  - MAC_Subtype_QoS_Poll
  - MAC_Subtype_QoS_ACK
  - MAC_Subtype_QoS_Data_ACK
  - MAC_Subtype_QoS_ACK_Poll
  - MAC_Subtype_QoS_Data_Poll
  - MAC_Subtype_QoS_Data_ACK_Poll

  This frame types have also been added to the frame demultiplexing, done when a packet is discarded.

- **ACK policies** - The following ACK policies have been defined:

- ACK_ACKP_NORMAL_ACK
- MAC_ACKP_NO_ACK
- MAC_ACKP_NOEXP_ACK
- MAC_ACKP_BLOCK_ACK

- **QoS control and MAC802.11e frame header**: A new header type has been added to the legacy MAC layer to support the QoS functionalities. This header has one more field, respect to the legacy ones, called *qos_control* which contains some fields of the *QoS Info* element described in the IEEE 802.11e draft. These fields are: TID, ackp bit, eosp bit and txop duration.

- **Saturation** - We have added also a functionality to the MAC layer that permits to make a station transmit in saturation. It means that, when a packet is received correctly, it is immediately scheduled another packet for transmission. Two new commands has been added to configure the saturation of a given station:

```
$mac saturation - Used to turn on the saturation

$mac saturation pktsize - Used to set the packet size
```

  where *$mac* is a reference to the MAC layer object.

8.3 Class Mac802_11e - Mac IEEE 802.11e Layer

8.3.1 Data Structures

We introduce a new terminology to group the frames depending on what they contain. In the list below there are the names of the frames sets :

- *QoS Data+* - This is the name of all the frame that contains a Data in their type: MAC_QoS_Data, MAC_QoS_Data_ACK, MAC_QoS_Data_Poll, MAC_QoS_Data_ACK_Poll

- *QoS ACK+* - This is the name of all the frame that contains a CF-Ack in their type: MAC_QoS_ACK, MAC_QoS_Data_ACK, MAC_QoS_ACK_Poll, MAC_QoS_Data_ACK_Poll

- *QoS Poll+* - This is the name of all the frame that contains a Poll in their type: MAC_QoS_Poll, MAC_QoS_Data_Poll, MAC_QoS_ACK_Poll, MAC_QoS_Data_ACK_Poll

Now we present a description of the data structures used at the MAC layer: **Class Status**

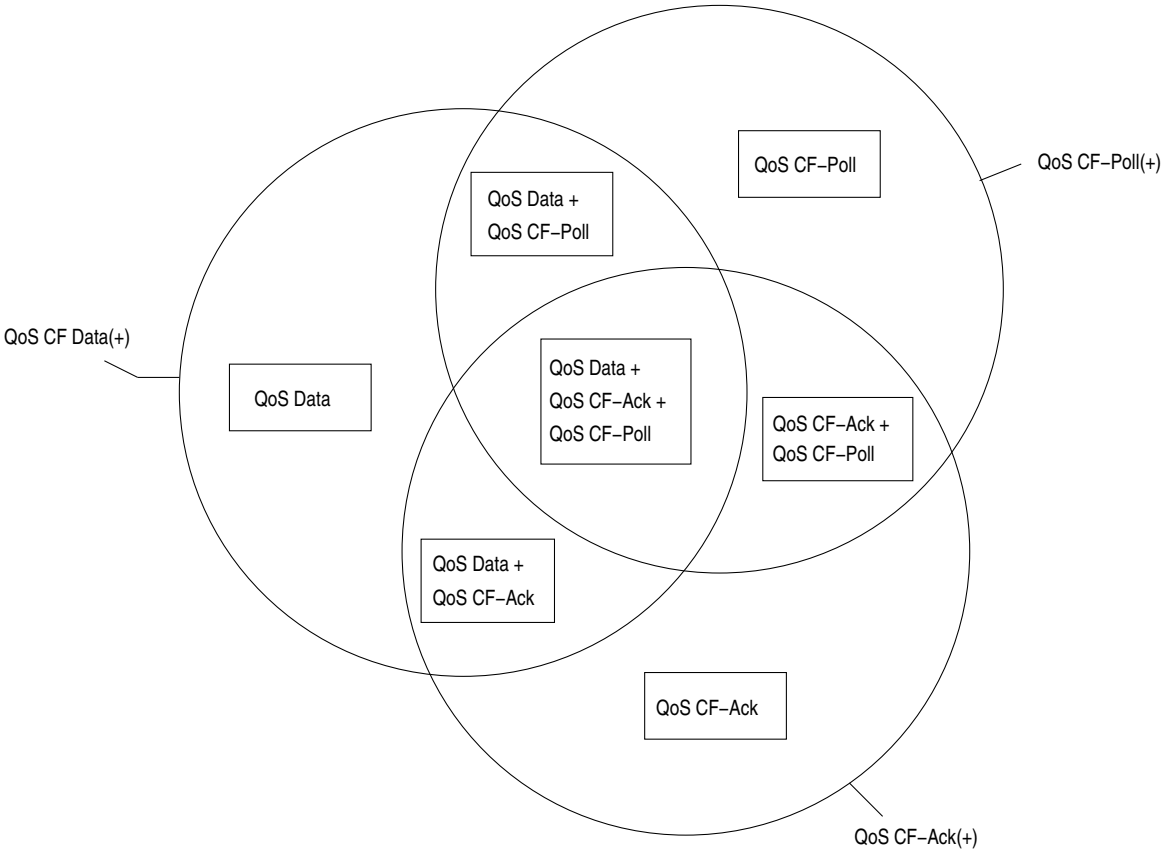

Figure 64: Ns2 – Frame type groups

An instance of this class is used to keep the internal state of the MAC layer of both the QAP and the QSTAs. It is defined a variable called **status_** of type *Status*, considered as an aggregate of various information that must be kept during the execution to guarantee the correct behavior of the simulator.

The field **has_control_**, specifies if the QoS station has or not the control of the medium. When a QoS station holds the control of the medium it has the opportunity to transmit one or more frames and it should recover from an error situation. Therefore the behavior of a QSTA and the QAP changes substantially whether the QoS station has or not the control of the medium.

The state of the next *QoS ACK+* frame to be transmitted depends on the following parameters. In fact, it elapses an interval of time between the reception of a *QoS Data+* frame and the send of an acknowledgment as response; during this time the information about the TID and MAC address of the ACK recipient is kept in the fields **ack_tid_** and **ack_dst_**. The field **is_next_ack_** is a boolean and, if it is true, the next frame to be transmitted should be a *QoS ACK+* frame and the data contained in the fields **ack_tid_** and **ack_dst_** are significative.

The field **sense_idle_**, is set to TRUE when there should be a transmission after sensing the medium idle for a certain period of time that is specified by another field called **idle_time_**.

The **piggyback_** field is used to know if the piggybacking feature is active or not.

The field **hcca_tx_active_** that is TRUE if the current transmission is controlled by the HCCA function; FALSE otherwise.

Finally, as a workaround to some problems in the implementation due to the incorrect interaction between the HCCA function and the legacy 802.11 MAC layer, we have introduce the field **atomic_** used to protect the HCCA transmissions. When its value is TRUE, it is not possible for the legacy access function to transmit any frame.

Class SchedulerTxData

In the MAC layer of each QoS station there is one instance **schedTx_** of type *SchedulerTxData*. It is used as a transmission buffer to make the scheduler communicate to the MAC layer the next frame that has to be scheduled for transmission. A reference to this variable is passed to the scheduler when the function *deque()* is called and then, the reference is used to keep the information needed to fill the frame fields. This variable contains information about the next frame to be sent, after it has been executed the *deque* function of the scheduler. This information is kept until it is called the next *deque* function and thus it can be used also to save the state about the last frame sent. The class fields are:

- *u_char subtype_* - This field contains the type of the next frame to be sent. The only valid values that can be assigned to it by the scheduler are: MAC_QoS_Data, MAC_QoS_Poll and MAC_QoS_Data_Poll, since the ACK frame response is completely managed by the MAC layer. This types, in fact, are then transformed into the QoS ACK(+) ones by the MAC layer if an ACK can be piggybacked in the next frame to be transmitted.

- *Packet* p_* - This is a pointer to the frame that has to be transmitted. It is different from NULL if and only if the next frame contains data in its payload.

- *int dst_* - This field is the destination address of the frame to be transmitted.

- *double txop_ (expressed in μs)* - It specifies the duration of the txop and is only meaningful if the frame type is QoS Poll(+).

- *int queue_size_ (expressed in bytes)* - It specifies the queue size at the QSTA. This field is valid if the frame is transmitted by a QSTA and its type is equal to QoS Data(+).

- *double duration_ (expressed in μs)* - It is used to set the NAV of the stations that access the medium using EDCA or DCF.

- *u_char eosp_* - This field contains the *eosp* bit.

- *int tid_* - This field contains the *tid* bit.

- *u_char ackp_* - This field contains the *ackp* bit and is used to communicate to the MAC layer the acknowledgment policy that has to be used to transmit the frame. Its value can be one of the following: MAC_ACKP_NO_ACK, MAC_ACKP_NORMAL_ACK, MAC_ACKP_BLOCK_ACK, MAC_ACKP_NOEXP_ACK.

8.3.2 Timers

There are many types of event that can happen at the MAC layer. We have classified these events according to the reason why they have been generated. A timer is associated to each type of event. A description of the timers used follows:

	*subtype_* $\neq$ NO_PKT	*subtype_* = NO_PKT
*is_next_ack_* = TRUE	It is transmitted a *QoS Ack(+)* frame	It is transmitted a QoS CF-Ack frame
*is_next_ack_* = FALSE	It is transmitted a QoS Data, QoS CF-Poll or QoS Data + QoS CF-Poll frame	It is not transmitted any frame and the station looses the control of the medium

Table 18: Ns2 – Action undertaken by the MAC layer when a frame is dequed

- **mhHCCATxTimer_** - This timer is used to trigger the transmission of a frame controlled by the HCCA function. The transmission can happen in several occasions: after the QAP senses the medium idle for PIFS, when an interval of time equal to SIFS or PIFS has elapsed after the end of a frame transmission and the sender still has the control of the medium, when a CF-Poll frame is received by a QSTA. When this timer expires the MAC layer calls the scheduler function *deque()* to retrieve the next frame to be transmitted. The information about the next frame are stored by the scheduler in the variable *schedTx_* and then read by the MAC layer to establish the following actions to undertake. There are two fields of the variable *schedTx_* that influence the type of the next frame that has to be transmitted: *is_next_ack_* and *subtype_*. The action undertaken by the MAC layer after calling the *deque()* function are summarized in Table 18.

  The triggering of *mhHCCATxTimer_* also causes the QAP to regain the control of the medium that has been passed previously to a QSTA. All the frames transmitted under the control of the HCCA function are sent by this timer, despite the fact that their types may be different or it should be waited a different period of time before sending them. It has been chosen to use only one timer for all the transmissions because the operations needed in order to transmit a frame are very similar for all the possible frame sequences. Besides, the actions are almost the same whether we consider the QAP or the QSTA; the only difference between the two types of station is when they achieve or loose the control of the

medium. In fact, only the QAP can achieve the control of the medium when this timer triggers, because it means that the medium has been sensed idle for PIFS. Anyway when the dequed frame type is equal to NO_PKT, the control is lost by both the QAP and the QSTA.

- **mhCap_** - This timer is used by the QAP to start a new CAP when it loses the control of the medium or when the HCCA access function is activated for the first time. The transmissions controlled by the HCCA function are grouped into CAPs and each CAP ends when the QAP looses the control of the medium. When it happens the MAC layer of the QAP calls the function *get_next_cap()*, of the scheduler interface, to retrieve the next CAP start time that is then used to set the *mhCap_* timer. The start of the next CAP is therefore established by the scheduler just after the end of the previous one. When the *mhCap_* timer expires, it is sensed the medium idle for PIFS and then the QAP regains the control of the medium. Once it has the control, the MAC layer asks the scheduler for the next frame sequence to be transmitted.

  After the *mhCap_* timer expires, the QAP can transmit more than one downlink frame, also addressed to different QSTAs, without the need to start the timer again. This is not possible if there are uplink transmissions in the CAP, because, after transmitting a QoS CF-Poll frame, the QAP looses the control of the medium. When a sequence of TXOPs are granted to the QSTAs, the *mhCap_* timer is in fact started after each QoS CF-Poll frame. The choice to trigger this timer every time the QAP looses the control of the medium brings hence to a higher triggering activity when uplink frame sequences are transmitted.

  When the *mhCap_* timer is triggered just after the transmission of a QoS CF-Poll frame, it is possible that it expires before the end of the current TXOP. If that happens the start of the new CAP should be delayed until the end of the current CAP to avoid uncorrect frame sequences. In order to avoid a malfunction due to overlapping CAP, if the timer expires when a previous CAP has already started but has not finished yet, it is neither sensed the medium idle nor transmitted any frame. This does not cause an interruption of the HCCA transmissions because the *mhCap_* timer is triggered just after the end of the current CAP. The end of a CAP at the QAP occurs when the type of the frame returned by the *deque()* function is equal to NO_PKT and hence, at this time, the control of the control of the medium is lost and the *mhCap_* timer triggered.

- **mhTxEnd_** - This timer is used to trigger the execution of some operations needed at the end of a frame transmission. As soon as the transmission ends, the sender must recover from the absence of the expected response from the peer if it has the control of the medium; therefore, the *mhHCCATxTimer_* timer must be triggered to make it expire after PIFS, so that the control is recovered if it is not detected a PHY-RXSTART indication after SIFS. A station should undertake the recovery procedure only if it has the control of the medium and the

ACK policy of the transmitted frame is equal to MAC_ACKP_NORMAL_ACK or MAC_ACKP_NOEXP_ACK. On the contrary if the ACK policy is equal to MAC_ACKP_NO_ACK or MAC_ACKP_BLOCK_ACK, the *mhHCCATxTimer_* timer expires after SIFS so that a new transmission can be started after this interval of time because it is not required the reception of an acknowledgment. In any case the medium state is set idle and the transmitter is set inactive.

8.3.3 Events

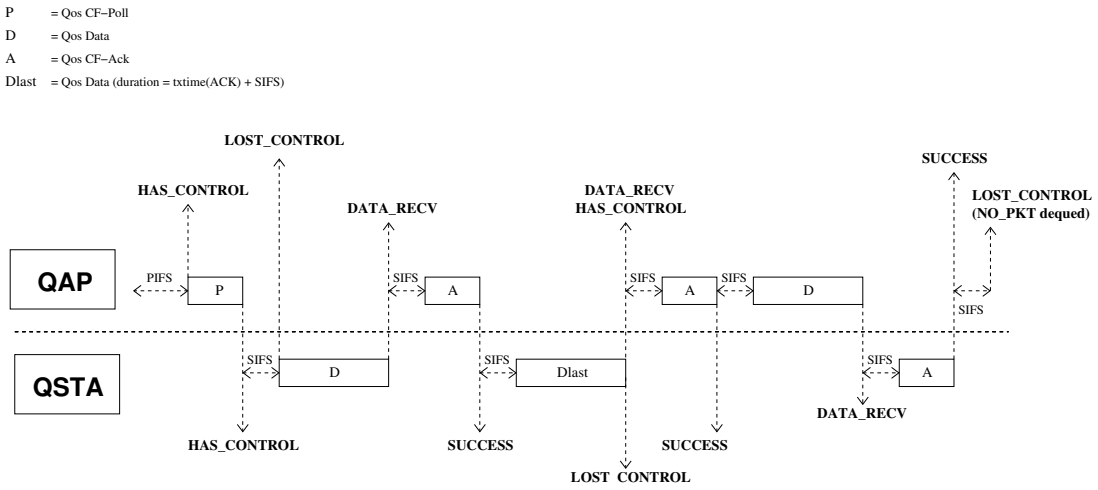

Figure 65: Ns2 – Events notified during a CAP without errors or collisions

The events are notified by the MAC layer to the scheduler to provide it with information about the MAC state. Our purpose has been to reduce the communication between the MAC layer and the scheduler as much as possible in order to maximize the efficiency of the simulator. The communication between the two entities we have provided the scheduler with the faculty to filter the events coming from the MAC layer. The scheduler in fact is notified of an event only if it has communicated previously to the MAC layer that it is interested in that particular type of event. The events are notified by the function *event()* that takes, as parameters, the event name and a pointer to the received packet (if there is one). A brief description of the events is presented in the following list:

- **HCCA_HAS_CONTROL** - This event is notified to a QoS station when it gains the control of the medium. The control is achieved by a QSTA when it receives a QoS Poll(+) frame and by the QAP when it senses the medium idle for PIFS after the start of a CAP or when it receives a QoS Data(+) frame whose *duration* field is equal to the time needed to transmit a QoS CF-Ack frame plus SIFS. Once a QoS station is notified of this event, it has the opportunity to transmit sequences of frames until it is notified that the control is passed to the

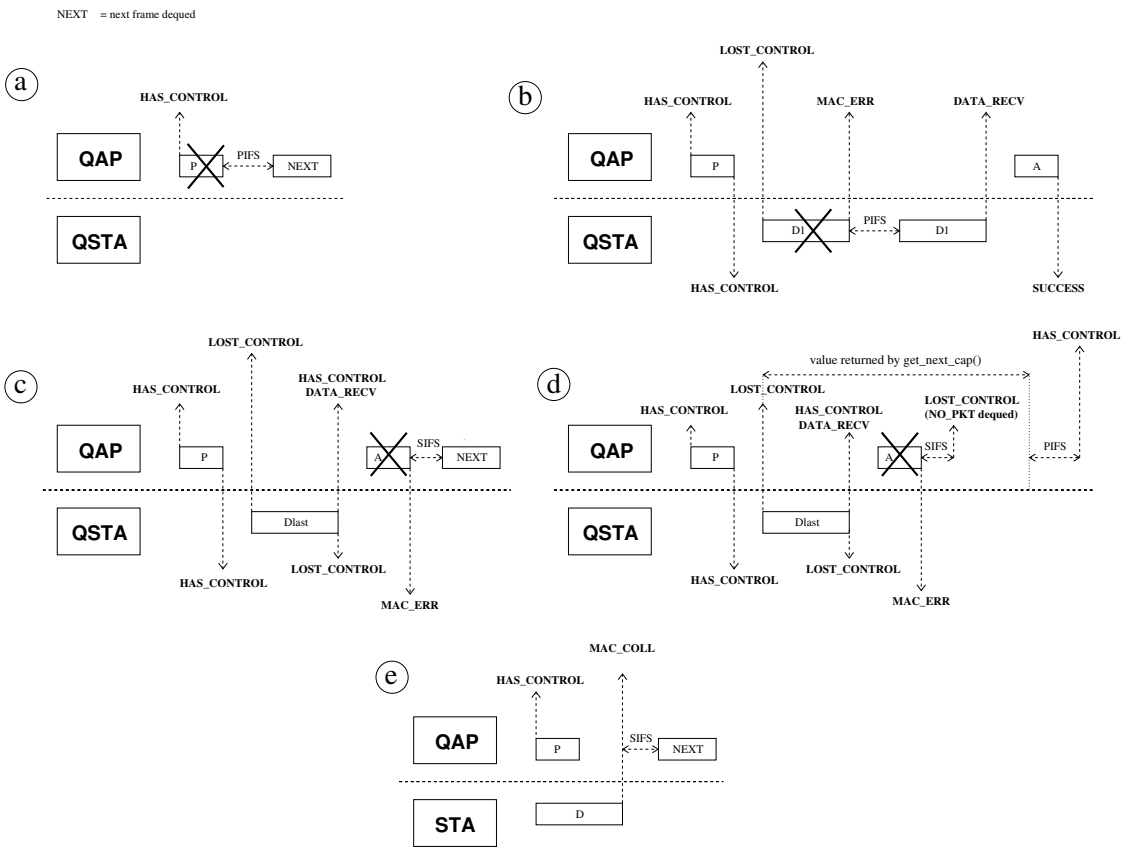

Figure 66: Ns2 – Events notified during a TXOP in presence of errors or collisions

peer or to the EDCA function. If this event is notified to the QAP, the *mhCap_* timer is stopped because there is no more the need to start a new CAP.

- **HCCA_LOST_CONTROL** - This event is notified to a QoS station when the control of the medium is passed to the peer or to the EDCA function. When a station does not have the control of the medium it cannot transmit any type of frames but QoS CF-Ack. If the control is lost by the QAP, it is also called the function *get_next_cap()* to retrieve the time at which the next CAP should be started and hence set the *mhCap_* timer such that it expires at that time.

  The control is lost at either the QAP or a QSTA when there are not frames to be transmitted but they still have the control of the medium. It is the MAC layer that asks the scheduler for the next frame to be transmitted by calling the function *deque()* and then reads the returned frame fields in the variable *schedTx_*. If the field *subtype_* of that variable is equal to NO_PKT, the control is lost. If we consider only the QSTA, the control of the medium is lost also when it is detected the PHY-RXSTART indication after sending the last frame that fits in a TXOP. QSTAs are required to set the NAV of the last Data frame of a burst to the time required to transmit a QoS CF-Ack plus SIFS. On the contrary, considering the QAP, the control is lost when it is detected the indication PHY-RXSTART after sending a QoS Poll(+) frame.

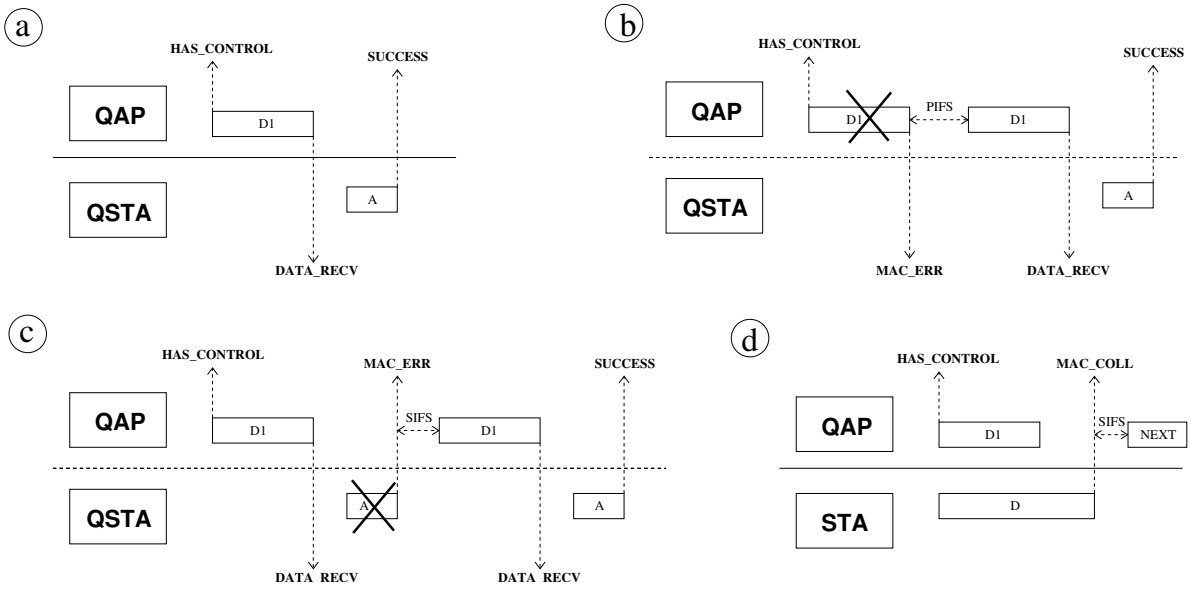

Figure 67: Ns2 – Events notified during a downlink transmission in presence of errors or collisions

- **HCCA_DATA_RECV** - This event is used to notify the QAP and QSTA schedulers that a QoS Data(+) frame has been correctly received.
- **HCCA_SUCCESS** - This event is used to notify the QAP and QSTA schedulers that an acknowledgment to a previously sent frame has been correctly received.
- **HCCA_TX_END** - This event is notified to the scheduler of a station that has sent a QoS frame as soon as the transmission ends and hence when the timer *mhTxEnd_* expires.
- **HCCA_HCCATX_HAND** - This event is notified to the scheduler when the timer *mhHCCATxTimer_* expires and there is the opportunity, for the QoS station, to transmit a frame.
- **HCCA_CAP_HAND** - This event is notified to the scheduler when the timer *mhCap_* expires indicating the start of a new CAP. It is notified only at the QAP. After the timer expiration the QAP has to sense the medium idle for PIFS before gaining the control of the medium.
- **HCCA_RECV** - This event is notified to the scheduler when any frame destined to this station is received and it is detected the PHY-RXEND indication.
- **HCCA_TRANSMIT** - This event is notified to the scheduler when the station transmits a frame and it is detected the PHY-TXSTART indication.
- **HCCA_MAC_COLL** - This event is notified to a QSTA when a collision of a transmitted frame is detected at the MAC layer.

- **HCCA_MAC_ERR** - This event is notified to a QoS station that receives incorrectly a frame. The MAC layer check the function *error()*, which belongs to the common header, to know if the frame is corrupted.

A set of typical frame exchange sequences is depicted in the figures from 65 to 67. We have indicated in the figures, as dashed arrows, the main events that are notified during those frame sequences.

The figures from 68 to 70 are the state charts of the MAC layer. The transition from a state to another one is represented by a solid arrow; above that arrow it is also specified the event that has triggered the transition. The main state chart is different whether we consider the QAP or the QSTA, while the other two, which describe the station behavior when it has or not the control of the medium, are the same.

8.3.4 Functions

In the following list it is provided a description of the main functions of the MAC layer:

- **void recv(Packet *p, Handler *h)** - This function is used to manage a received downlink or uplink packet. This function has been overridden. The operations performed are:

  - When p points to a downlink packet, which can be classified as an HCCA frame and has been assigned with a proper TID, that packet is enqueued at the scheduler using the function *enque()*, while the legacy *recv()* is not called.
  - The legacy *Mac802_11::recv()* function is called if the packet is not destined to the HCCA scheduler.
  - The calling of this function correspond to the physical PHY-RXSTART indication. When it is detected such indication, the control of the medium is lost in two cases. The first one is when the QAP has already sent a QoS Poll(+) frame and is waiting a response from the peer to interrupt the *mhHCCATxTimer_* timer needed to recover from the absence of the expected reply. The second one is when a QSTA has already transmitted a QoS Data frame, whose *duration* field is equal to the time needed to transmit a QoS CF-Ack frame plus SIFS, and is waiting such indication to know if the control has passed correctly to the QAP. It is then called the function *has_control()* with FALSE as parameter, to pass the control to the peer. In any case the timer *mhHCCATxTimer_* is stopped.
  - This function is called either when it is received a packet coming from the physical layer or when it is received a packet coming from the scheduler. When the received packet comes from the physical layer but a transmission has already been started, it is signaled a collision by the event HCCA_MAC_COLL and the error flag of the received packet is set.

- **void recv_timer (Packet *p, Handler *h)** - This function is called when it is detected the PHY-RXEND indication, i.e. the reception has finished. If it is not received a QoS frame, the legacy *recv_timer()* is called. The following operations are performed even if the frame is not destined to the station that receives it:

  - When it is received a self QoS CF-Poll frame by the QAP, the packet is discarded and the medium is set to idle.
  - If the frame is not received correctly due to channel error, collision or because it is destined to another station, and the station has the control of the medium, it is activated the recovery procedure that will trigger the transmission of a frame after SIFS by setting the timer *mhHCCATxTimer_*.
  - If it is received a frame, whose type is *QoS Ack(+)* but it is not addressed to the QSTA that receives it, and if the *qack* bit of the receiving QSTA is set, it is notified the successful transmission of the last sent frame by notifying to the scheduler the event HCCA_SUCCESS.
  - If the receiving QoS station address is different from the frame destination address, the *nav_* value is updated and the *mhNav_* expiration time is set to the *duration* field of the received frame.

  After the actions mentioned above, the packet is discarded if it is not addressed to the receiving QoS station. The following operations are performed only when there is a match between the frame destination address and the receiver address:

  - If the ACK policy of the received frame is NORMAL ACK, the TID and destination address of the next acknowledgment frame are stored in the variable *status_*. It is also set the field *is_next_ack* of that variable such that the next frame type will be *QoS Ack(+)*.
  - If the received frame type is *QoS Poll(+)* it means that the receiving QoS station is a QSTA and that it has gained the control of the medium passed by the QAP. A QSTA gains the control only if the *duration* field of the received *QoS Poll (+)* frame is not equal to zero. In fact whenever a *zero poll* is received by a QSTA, it should not undertake the recovery procedure and should not transmit more than one frame; therefore the control should not be passed to that QSTA.
  - If the received frame type is equal to *QoS Data(+)*, the received packet is passed to the upper layers.
  - If the received frame type is equal to QoS Null or *QoS Data(+)* the event HCCA_DATA_RECV is notified to the scheduler.
  - If the received frame type is equal to *QoS Ack(+)* and the QoS station has previously sent a frame that requires an immediate acknowledgment, the event HCCA_SUCCESS is notified.

- If the received frame is destined to the QAP and its duration is equal to the time required to transmit a QoS CF-Ack frame plus SIFS, it means that it is the last frame transmitted by the QSTA in the current TXOP. At this time the QAP should therefore gain the control of the medium and start the next frame sequence.
- If the station has the control of the medium or an ACK must be transmitted, a frame SIFS after the PHY-RXEND indication has to be sent. In any case the medium state is set idle.

- **void capHandler (void)** - This functions is the handler of *mhCap_* timer. This function is reached after the medium has been sensed idle for PIFS. It only transmits the next frame to be dispatched.

- **void registerEvent(HccaEvent e)** - This function is called by the scheduler if it wants to be notified by the MAC layer about event *e*. This is the only function of the MAC module that may be called by the scheduler.

- **void hccaTxHandler(void)** - This function is the handler of the *mhHCCATxTimer_* timer. If it has been called after sensing the medium idle for a certain period, the QAP gains the control of the medium. The next frame to be transmitted is dequed. If there are not any frames to be dispatched the QoS station loses the control of the medium.

- **Packet* getPacket(void)** - This function is used by the MAC layer to get information about the next packet to be transmitted. The function *deque()* of the scheduler interface is called. The next frame information is stored in the variable *schedTx_*. Then, a new *Packet* structure is initialized, using that information, and returned to the function *hccaTxHandler()* that will send the packet to the lower layers. The initialization of the packet depends on the fields of the variables *status_* and *schedTx_*. If the field *is_next_ack_* is equal to TRUE, the next frame type has to be equal to *QoS Ack(+)*. In that case, the acknowledgment can be piggybacked in the dequed frame only if all the following conditions are true:

  - The value of the variable *piggyback_* is true.
  - The destination addresses of the acknowledgment and of the dequed frame are the same, or the *qack* bit relative to the ACK recipient is equal to 1.
  - The dequed frame type is different from NO_PKT.

  If one of those conditions is not true, it is not possible to piggyback the ACK in the dequed frame and hence a QoS CF-Ack frame is sent instead. The problem is that a frame has already been dequed at the scheduler but will not be sent because is should be transmitted a QoS CF-Ack before. The scheduler should therefore be informed that the dequed frame has not been transmitted to keep its state consistent. This is accomplished by calling the function *rollback()* of the

scheduler interface that has the purpose to bring the scheduler state to the one present before the calling of the *deque()* function. If the *deque()* function does not change the scheduler state, then the *rollback()* function can be left empty. Finally, if there is no ACK pending and the dequed frame type is equal to NO_PKT, no frame is transmitted.

- **void txEndHandler (void)** - This function is the handler of *mhTxEnd_* timer and it is called when the transmission of a frame ends. It performs the following operations: if the ACK policy is equal to NO ACK or BLOCK ACK, it is transmitted a frame after SIFS by setting the timer *mhHCCATxTimer_*. On the contrary, if the policy is equal to NORMAL ACK or NO EXPLICIT ACK, it is set *mhHCCATxTimer_* is set to expire after PIFS for recovery purpose. In any case the medium state is set to idle.

- **void has_control(bool hc)** - This function is called either when a QoS station gains the control of the medium or when it passes the control to the peer. As a result, the internal MAC state is updated and it is notified to the scheduler the event HCCA_HAS_CONTROL or HCCA_LOST_CONTROL depending on the current value of the field *has_control_* of the variable *status_*. If the QoS station is a QAP, when the control is gained, a new CAP is started using the function *startCAP()*; on the contrary, when the control is passed to the peer, the *mhCap_* timer is stopped.

- **void startCAP(void)** - This function is used by the QAP to start a new CAP, setting the *mhCap_* timer. The start time of the next CAP is retrieved by calling the *get_next_cap()* function of to the schedulers interface. If the QAP has lost the control of the medium, due to the start of a TXOP, the expire time is not set to the value returned by scheduler function. In fact the next CAP start should be postponed until the end of the current TXOP. The end time is considered equal to the start time plus the maximum duration of the TXOP communicated in the *QoS Poll (+)* frame. This approach avoids interferences between the transmissions that belong to the new CAP and those that belong to the current TXOP.

8.3.5 Ns Commands

The following *ns* commands have been added to the previous command set (the words whose first character is "$" are TCL variables, the ones between square brackets are parameters, while the others are keywords):

```
$mac qack [val]
$mac qack [val] [qsta]
```

where *$mac* is a reference to the MAC layer object, *val* is the value of the *qack* bit (the possible values it can assume are: 0, 1) and *qsta* is the address of the QSTA to which

Figure 68: Ns2 – Main state chart of the MAC layer

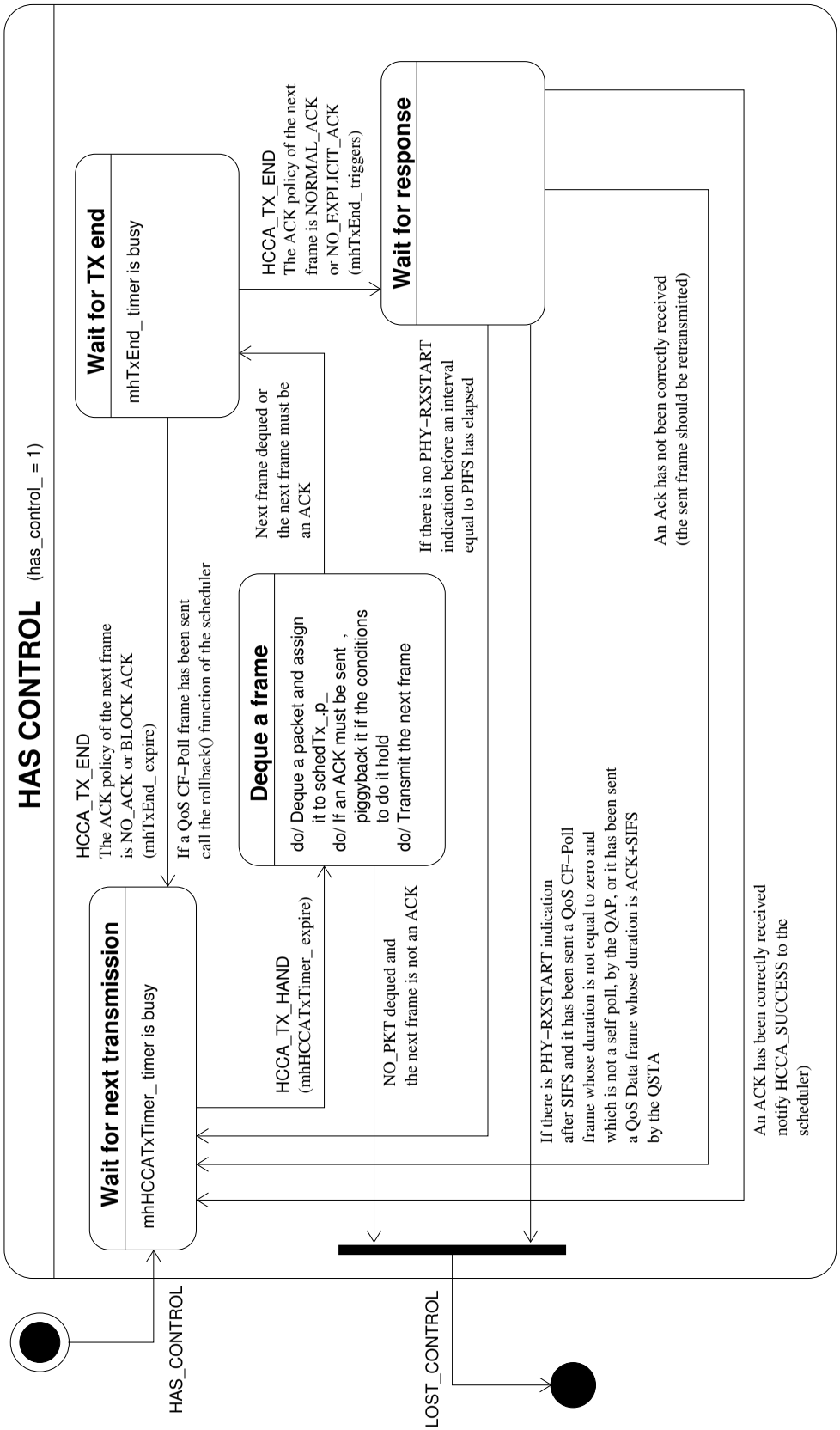

Figure 69: Ns2 – State chart of the MAC layer, when the station has the control of the medium

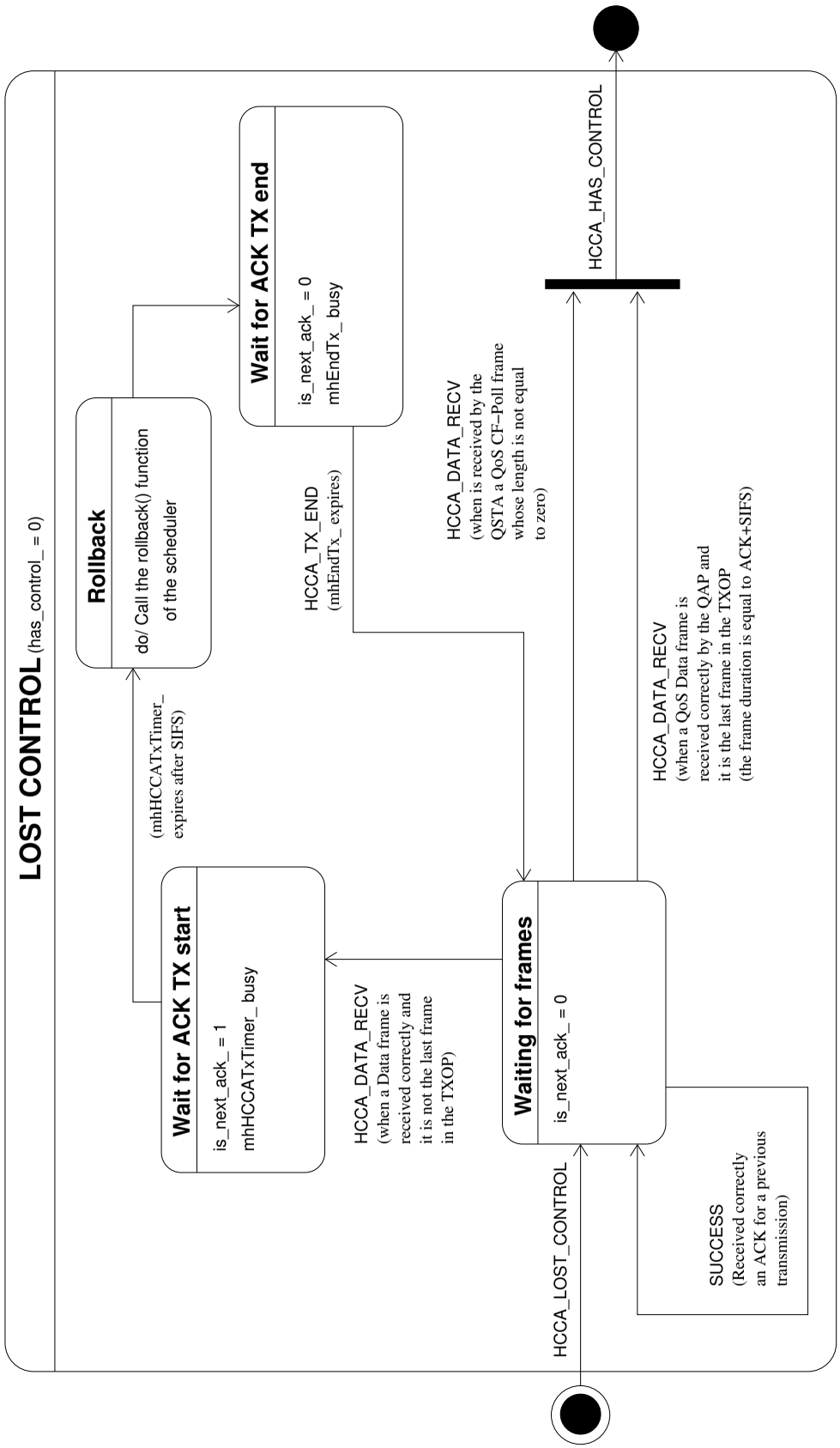

Figure 70: Ns2 – State chart of the MAC layer, when the station has not the control of the medium

the *qack* bit refers. The first command can be used only by a QSTA to set the *qack* bit, while the last one can be used only by the QAP. In order to ensure a correct behavior of the simulator, the value assigned to the *qack* bit at a QSTA should be equal to the value assigned to the *qack* bit at the QAP that refers to that QSTA. There are in fact one *qack* bit for each QSTA and an array of *n qack* bit at the QAP; each one of the array elements corresponds to different QSTA.

```
$mac piggyback [val]
```

If the value of *val* is equal to 1, the piggyback feature is activated. This command should be called using the same value at all the QoS stations in the system to avoid an inconsistent behavior of the simulator.

```
$mac scheduler set_opt ...
```

This command has a variable number of parameters that should be anyway greater than 5. It is used to call the function *command()* of the scheduler interface passing as arguments all the parameters after *set_opt*. Using this feature is possible to call directly *ns* commands of the scheduler.

8.4 Class MacHccaSched - Scheduler Interface

It is proposed in this section the common scheduler interface. Our objective has been to realize it as more as general as possible such that it can be used in the developing of several types of schedulers. In the following list it is provided a description of the functions that compose the interface:

- **void enque(Packet* p, Handler* h)** - This function is called by the MAC layer to enque a packet in one of the scheduler internal queues.
- **void deque(SchedulerTxData* schedTx_)** - This function is called by the MAC layer to retrieve the next packet to be transmitted. This function may be called also to see if the ACK can be piggybacked in the next frame. In this case, if the piggybacking is not possible, it is transmitted a QoS CF-Ack frame instead of the dequed packet. The function *rollback()* prevent the scheduler to be on an inconsistent state. The MAC layer in fact calls that function when it does not actually transmit the dequed frame. The scheduler must fill the fields of the *schedTx_* parameter to communicate the information about the next packet to be transmitted. In particular the *ackp_* field, which specifies the ACK policy, can contain only the values MAC_ACKP_NO_ACK or MAC_ACKP_NORMAL_ACK. It is not possible for the scheduler to assign the value MAC_ACKP_NOEXP_ACK to the *ackp_* field because it is reserved for the MAC layer. The deque function can be called only after having notified the event HCCA_HAS_CONTROL and it cannot be called after having notified the event HCCA_LOST_CONTROL. This implies that it is called only when the station has the control of the medium.

- **void event(HccaEvent e, Packet* rx)** - This function is used to notify the events described in the previous section. It is also passed, as a parameter, a pointer to the reception buffer because sometimes the scheduler needs information about the received frame, when it is notified about an event. This happens, e.g., when the QSTA scheduler is notified of the event HCCA_HAS_CONTROL and it wants to retrieve the TXOP length in the just received *QoS Poll(+)* frame. This function is called to notify only the events that have been registered previously by the function *register()* of the MAC layer.

- **double get_next_cap(void)** - This function is implemented only at the QAP scheduler because only the QAP can start a CAP. It must return the interval of time that has to elapse before the start of the new CAP. It is called only after the QAP has lost the control of the medium and hence after the HCCA_LOST_CONTROL event and before the HCCA_HAS_CONTROL one. If the returned value is less than zero, it is not set the next CAP start time. Finally, if the current virtual time plus the returned interval of time are less than the end time of the current TXOP, then the start of the next CAP is put off until the end of the current TXOP.

- **u_char get_queue_size(int tid)** - This function is implemented only at the QSTA scheduler and it is used by the MAC layer to retrieve the queue length for a given TID. It is called when a QSTA sent a QoS frame to the QAP and piggybacked in it the information about the queue size relative to the TID of that frame.

- **void rollback(void)** - When the MAC layer does not transmit a previously deque frame, the internal state of the scheduler can become inconsistent; therefore this function is used to avoid this situation and signal to the scheduler when it should be restored the last consistent state. If the state does not change during the deque operation, this function may be left empty.

- **int command(int argc, const char*const* argv)** - This function is used to pass a TCL command directly to the scheduler. It is called from the *command()* function of MAC layer. The TCL command to be used has been described previously.

In figure 8.4 it is shown the state chart of the scheduler, where are indicated, as arrows, the functions that belong to the scheduler interface.

8.5 Class MacHccaSchedMap - Offline Schedule Builder

This class is used to build the offline schedule needed by both the EDF-B and the reference schedulers to serve the TSs. We present here a set of functions that constitute a common interface; in our implementation those functions are used by the class *MacHccaSchedQAP_periodic* to create and manage the offline schedule and to know

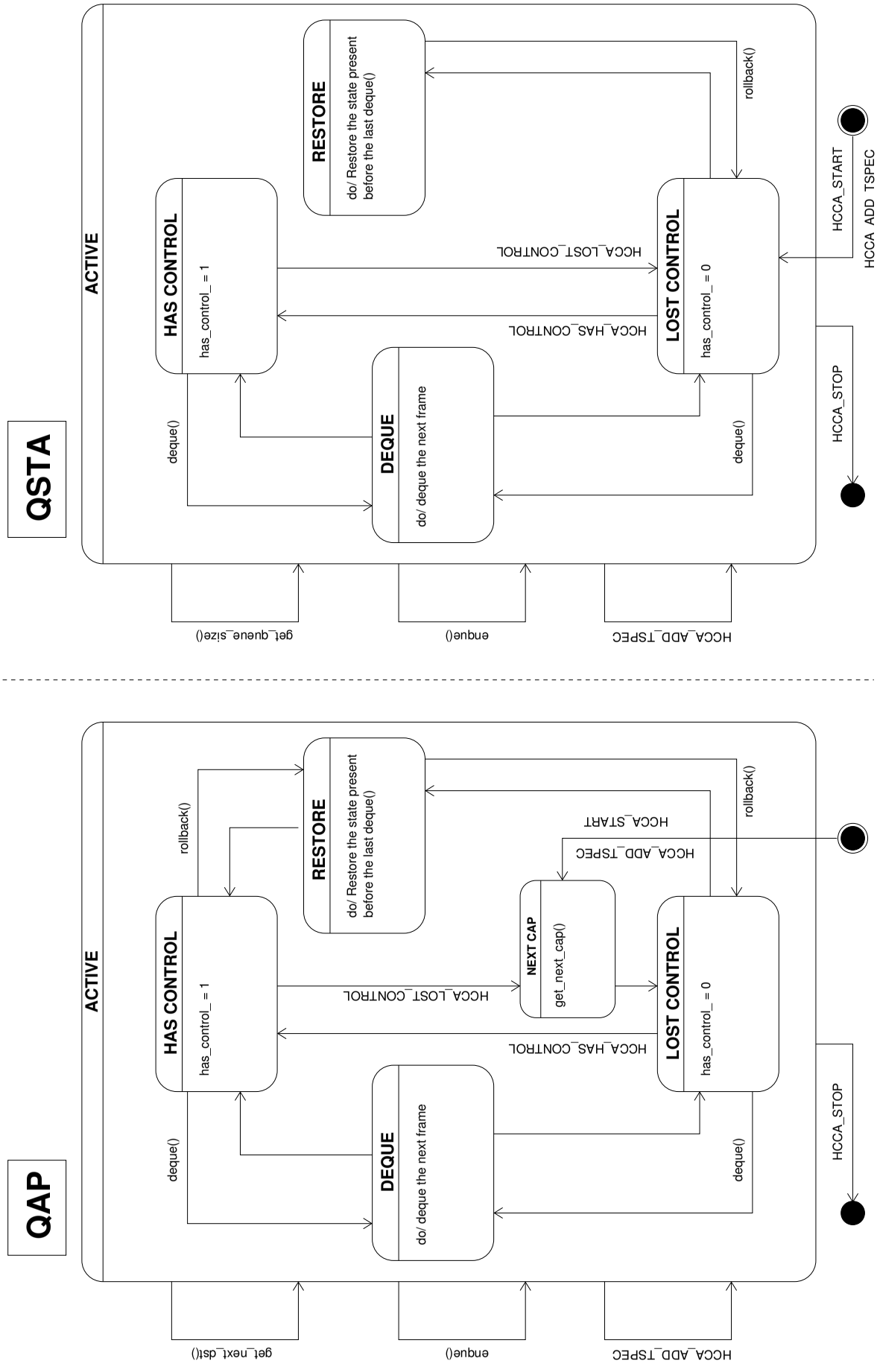

Figure 71: Ns2 – Scheduler state chart

the start time and duration of the TXOPs. A list of TXOPs has to be stored in this class. For each TXOP, all the information that concerns it has to be memorized in a structure called **txop_desc**. The *current TXOP* is the one that should be scheduled next and it is *active* if its start time is earlier than the current virtual time.

- **virtual int createMap (TSPEC& tspec)** - When this method is called, the offline schedule is actually built exploiting the TSPEC parameters passed as argument. The offline schedule is not returned, but it is store internally and the information about it can be accessed only by the proper methods of this class.
- **virtual txop_desc* getTXOPDesc (void)** - This method returns the structure that contains the information about the *current TXOP* (the one that should be scheduled next).
- **virtual void save_state(void)** - This method is used to save the information about the current TXOP. It is called by the scheduler at the start of the *deque()* function, so that the saved state can be used the to restore the information about the last TXOP granted.
- **virtual void rollback(void)** - When this method is called, it is restored the state saved by the method *save_state()*. The state is composed by the following information about the current TXOP: pointer to the structure that represents it and start time.
- **virtual void next_txop(void)** - This method moves the pointer to current TXOP to the next element in the list.

There are also two methods that are implemented directly in this class because are common to all the implementations of the offline schedule builder.

- **bool is_TXOP_active(void)** - This method return TRUE if the value of the field *TXOP_start_time* is less than the current virtual time. When it returns TRUE, it means that the current TXOP is active.
- **double get_next_cap(void)** - It returns the start time of the current TXOP.

8.6 Trace File

A new trace file format has been added to keep track of the MAC802.11e state changes and of the reception or transmission of a QoS frame mostly, for debugging purposes. It is shown an example trace file and, after that, it is described its format:

```
*) -t 98.839589 -e HCCATX_HAND -st QSTA -ad 18 -hc 1 -isA 0 -hon 1
   -isI 1 -si 0.000000 -it 98.839559
*) -t 98.839589 -e TRANSMIT -st QSTA -ad 18 -TX -dst 0 -src 18
   -ty _D_ -tid 1 -txop 0 -queue 254 -dur 5432 -eosp 1 -ackp NR -hc 1
   -isA 0 -hon 1 -isI 0 -si 0.000000 -it 98.839559 -rx 0 -tx 256
```

```
*) -t 98.839822 -e TX_END -st QSTA -ad 18 -hc 1 -isA 0 -hon 1
   -isI 1 -si 0.000000 -it 98.839822
*) -t 98.839822 -e RECV -st QAP  -ad 0 -RX -dst 0 -src 18 -ty _D_
   -tid 1 -txop 254 -queue 64768 -dur 5432 -eosp 1 -ackp NR -hc 0
   -isA 0 -hon 1 -isI 0  -si 0.000000 -it 98.839559 -rx 16 -tx 0
*) -t 98.839822 -e DATA_RECV -st QAP  -ad 0 -hc 0 -isA 1 -hon 1
   -isI 0 -si 0.000000 -it 98.839559
```

The format of the trace file depends on the logged event. The meaning of each log type follows:

- **Base log** - The fields of this log are used for all the events and contains the event name, time and station type:

  ```
  -e      Event time
  -t      Event name
  -st     Station type (QAP, QSTA)
  ```

- **Transmission and reception log** - The fields of this log are used for the events for which has to be logged the received or transmitted packet. The fields for transmission or reception are the same, but they are preceded by *-TX* and *-RX* respectively to make the user distinguish the two cases:

  ```
  -TX     tells that the following fields has to be interpreted
          as informations about the packet in transmitted
  -TR     tells that the following fields has to be interpreted
          as informations about the packet received
  -dst    destination address
  -src    source address
  -ty     packet subtype
  -tid    tid
  -queue  queue size
  -txop   txop duration
  -dur    duration, used to set the NAV of the stations which
          receives the packet
  -eosp   eosp bit
  -ackp   ackp bit
  ```

  The packet subtype has printed using three characters: the first one is set to '*A*' if the packet is of type QoS ACK(+), the second one to '*D*' if it is of type QoS Data(+) and to '*N*' is it is of type QoS Null, the third one is set to '*P*' if the type is QoS Poll(+). All the three characters are set to '_' in the other cases (see

the example). The *ackp* bit format is the following: *NR* stands for NORMAL ACK, *NO* stands for NO ACK, *EX* stands for NO EXPLICIT ACK, *BL* stands for BLOCK ACK and _ stands for unknown.

- **State log** - This is the log of the internal MAC802.11e state:

```
-hc     true if the station has the control of the medium
-isA    true if the next frame must be an ACK
-hon    true if the HCCA function has been turned on
-isI    true if the medium is idle
-si     time for which the medium has to be sensed idle
-it     last time the medium has became idle
```

8.7 Future Extensions

There are still some features of the IEEE 802.11e draft that we have not implemented yet. It is provided a list of possible future extensions:

- Beaconing
- Block Ack feature
- Direct Link Protocol (DLP)
- Multirate support