OMNeT++ Community Summit, 2016

An outline of the new IEEE 802.11 model in the INET framework

Levente Mészáros

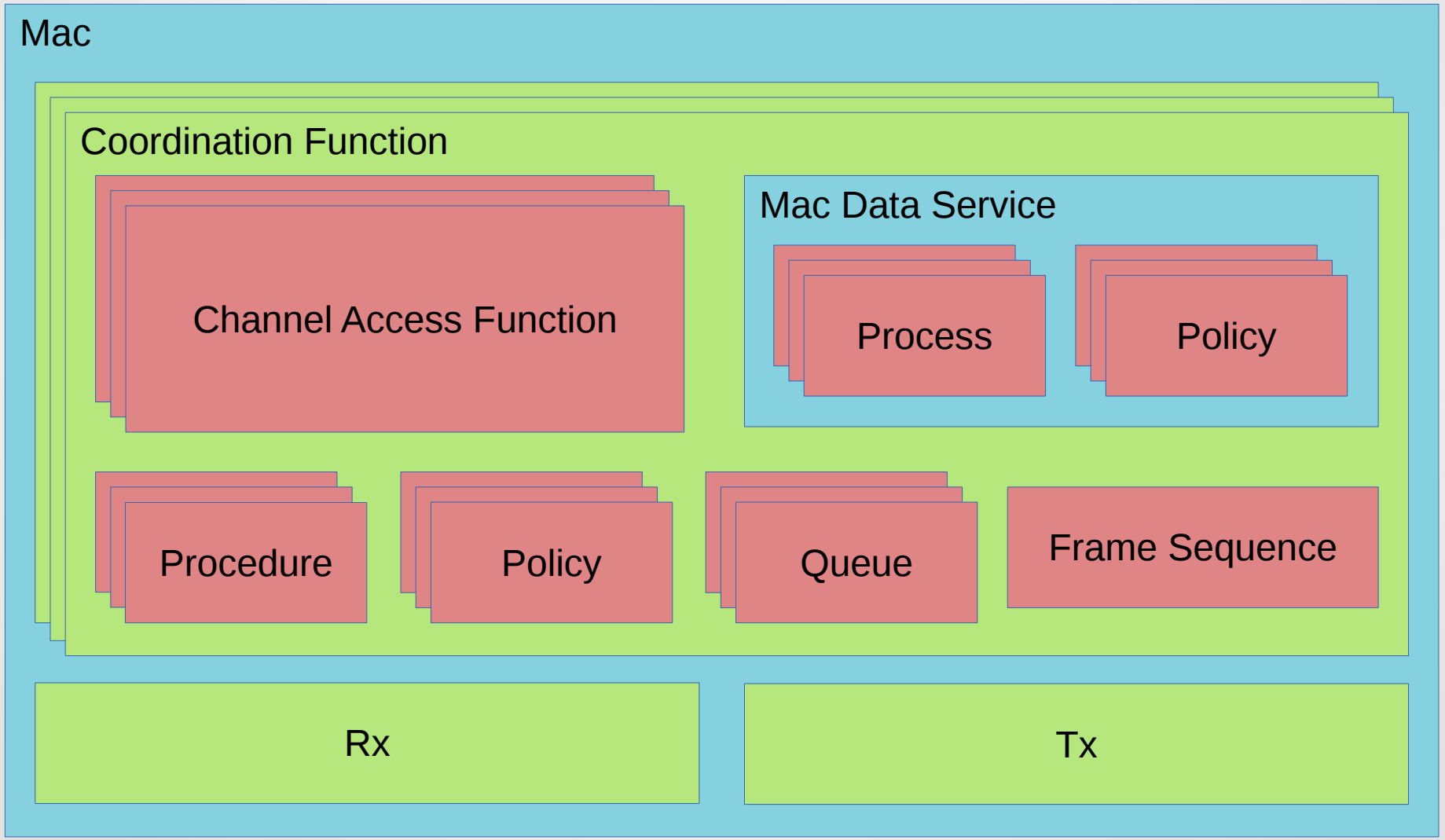
Quick Recap

- The old model was a dead end
- Design draft for a new model was presented at the OMNeT++ Community Summit 2015 in Zurich
- First version was released in INET-3.1.1, October, 2015
- The old model was replaced in INET-3.2, December, 2015
- Benjámin M. Seregi works on the model ever since
  - First design draft has been significantly reworked

Model Goals

- Full-featured and validated model
- Directly implement the standard
  - Implementation mirrors the concepts in the Standard
- Modular, pluggable architecture
  - Allow experimentation
  - Widely configurable

Conceptual Architecture



Some Experimentation Options

- New policies can be defined by the user to allow experimenting with non-standard scenarios
  - Custom ACK policy (e.g. for long-range wifi)
  - Custom policy for RTS/CTS protection
  - Fragmentation/aggregation policy
  - Block ACK agreement initiatation/termination policy
- Custom rate selection and new rate control algorithms
- Custom backoff procedure
- New frame sequences

Experimenting with Rate Management

- Rate selection
  - Assigns rate based on frame type and receiver

```
class INET_API IRateSelection {
  public:
    virtual const IIeee80211Mode *computeMode(Ieee80211Frame *frame) = 0;
    virtual const IIeee80211Mode *computeResponseCtsFrameMode(Ieee80211RTSFrame *rtsFrame) = 0;
    virtual const IIeee80211Mode *computeResponseAckFrameMode(Ieee80211DataOrMgmtFrame *dataOrMgmtFrame) = 0;
};
```

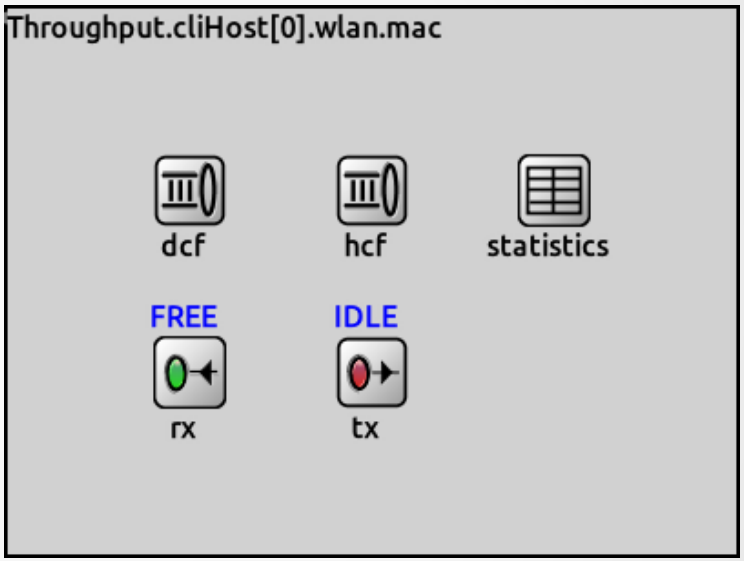
- Rate control
  - Determines optimal rates based on channel quality

```
class INET_API IRateControl {
  public:
    virtual const IIeee80211Mode *getMode() = 0;
    virtual void processTransmittedFrame(const Ieee80211Frame *frame, int retryCount, bool isSuccessful, bool isGivenUp) = 0;
    virtual void processReceivedFrame(const Ieee80211Frame *frame, const Ieee80211ReceptionIndication *receptionIndication) = 0;
};
```

Contents

Coordination Functions

- Implemented as compound modules using C++ classes derived from cModule
  - `Dcf`
  - `Hcf` (`Edca` only)
- Unimplemented
  - `Pcf`
  - `Mcf`

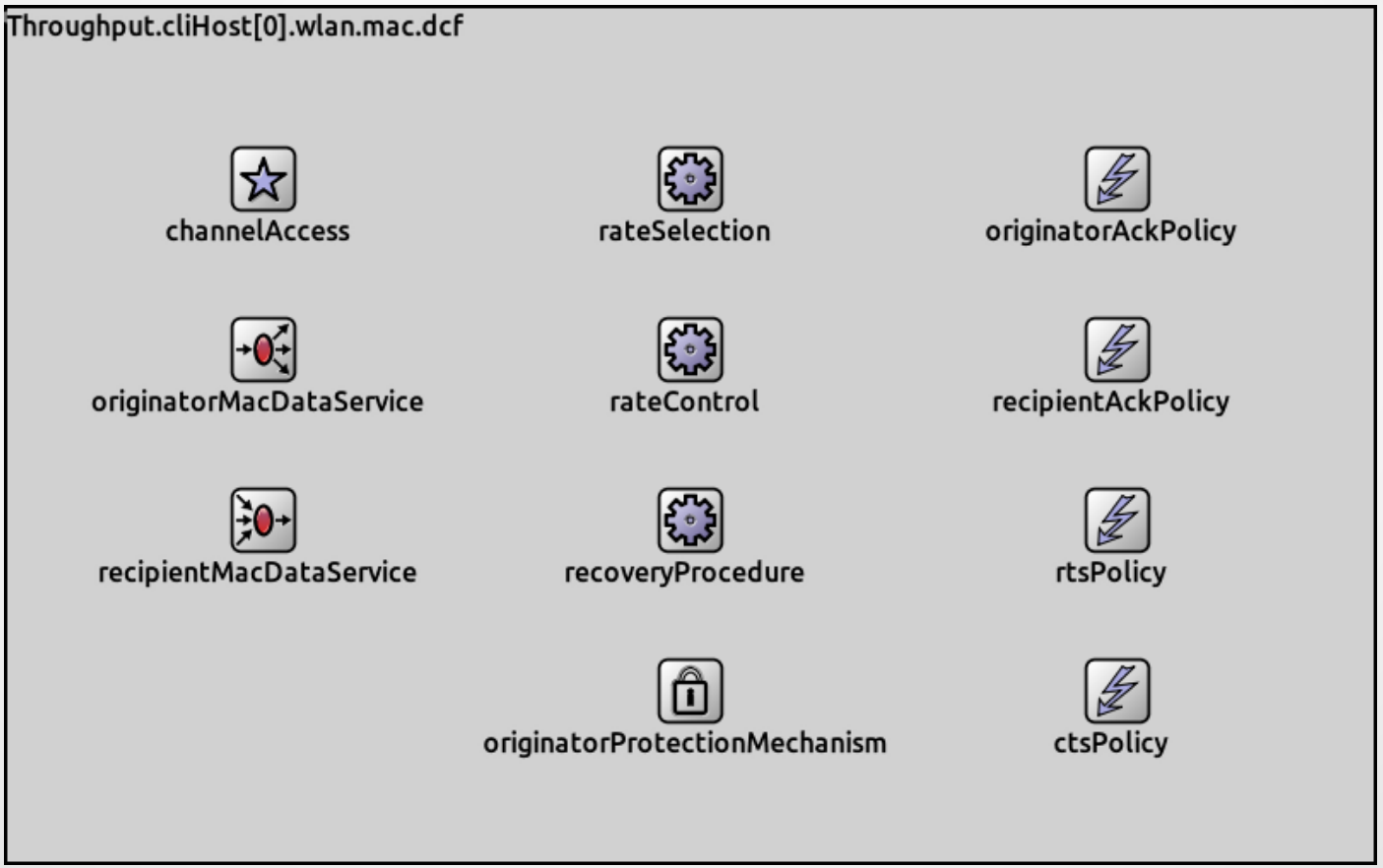


IEEE 802.11 Mac

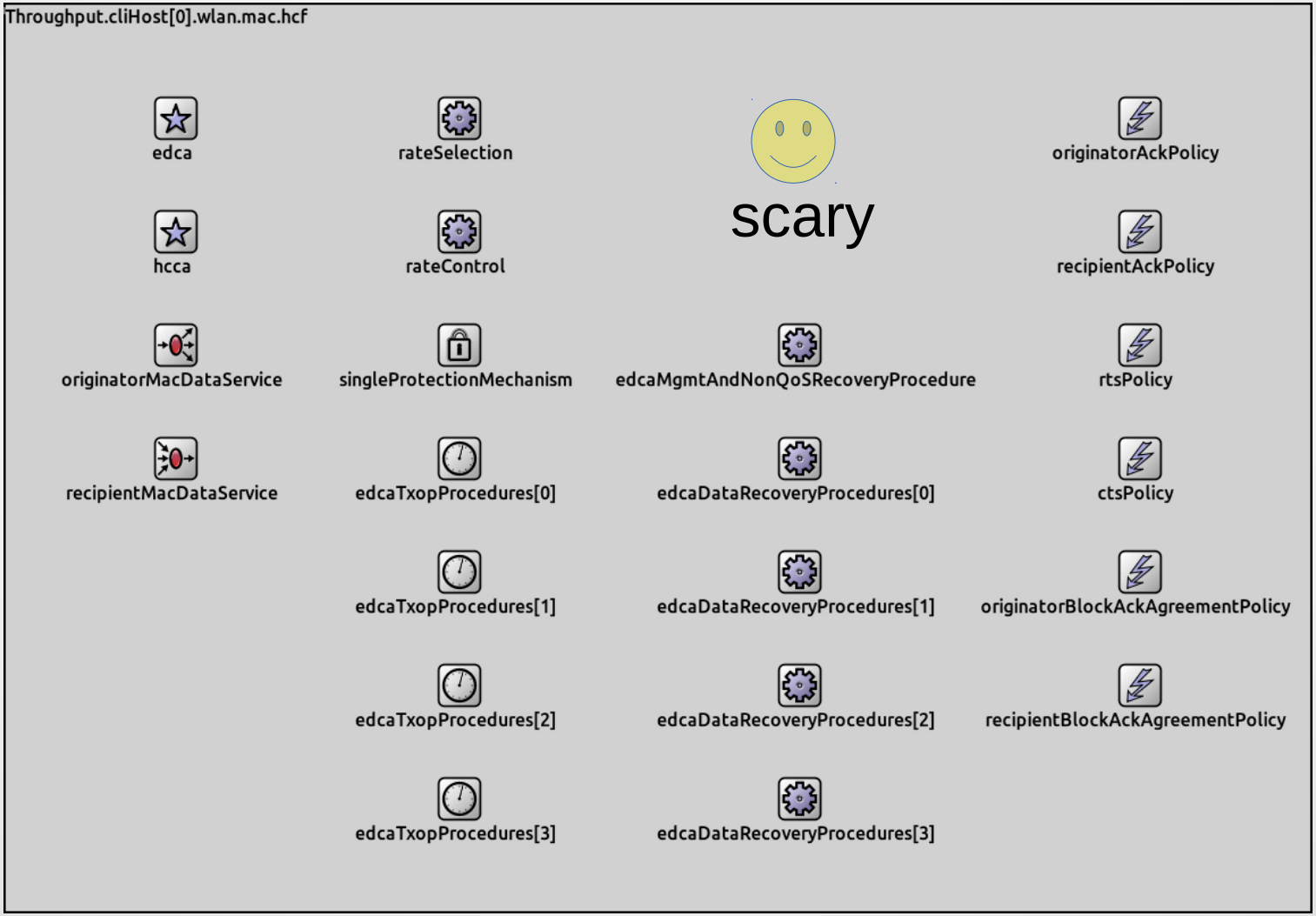
```
class INET_API ICoordinationFunction {
  public:
    virtual void processUpperFrame(Ieee80211DataOrMgmtFrame *frame) = 0;
    virtual void processLowerFrame(Ieee80211Frame *frame) = 0;
};
```

Distributed Coordination Function (Dcf)

- Submodules communicate via direct C++ method calls



Hybrid Coordination Function (Hcf)



Contents

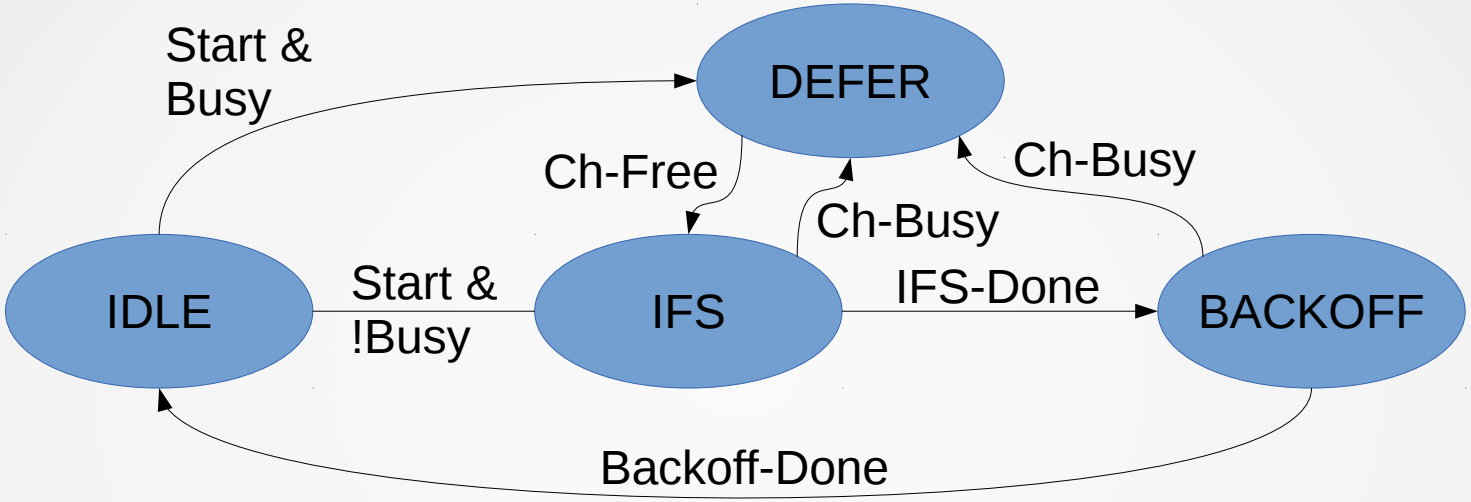
Channel Access Functions

- Implemented as compound modules using C++ classes derived from cModule
  - Dcaf
  - Edcaf

Channel Access Function

```
class INET_API IChannelAccessFunction {
  public:
    class ICallback {
      public:
        virtual void channelGranted(IChannelAccessFunction *channelAccess) = 0;
    };
  public:
    virtual void requestChannel(ICallback *callback) = 0;
    virtual void releaseChannel(ICallback *callback) = 0;
};
```

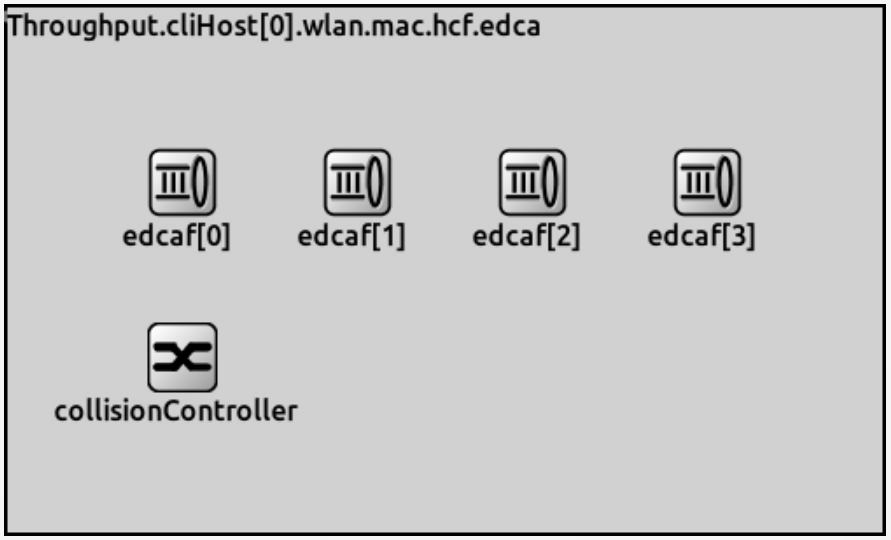
Backoff Procedure




```
class INET_API IBackoffProcedure {
  public:
    class ICallback {
      public:
        virtual void expectedBackoffProcedureFinish(simtime_t time) = 0;
        virtual void backoffProcedureFinished() = 0;
    };
    virtual void startBackoffProcedure(int cw, simtime_t ifs, simtime_t eifs,
                                       simtime_t slotTime, ICallback *callback) = 0;
};
```

Enhanced Distributed Channel Access (Edca)

- `Edca` contains one `Edcaf` per access category (`AC`)
- `EdcaCollisionController` resolves internal collisions




```
void Dcaf::backoffProcedureFinished()
{
    owning = true;
    callback->channelGranted(this);
}
```

```
void Edcaf::backoffProcedureFinished() {
    if (!collisionController->isInternalCollision(this)) {
        owning = true;
        callback->channelGranted(this);
    }
}
```

Contents

Procedures

- Procedures answer *how* to do something as opposed to *when*
- Our procedure implementations directly follow the standard
- Implemented as C++ classes
  - Backoff procedure
  - Ack procedure
  - Rts/Cts procedure
  - Block Ack Agreement procedure
  - Block Ack procedure
  - Recovery procedure
  - Protection mechanism
  - TxOp procedure

Procedure Example

- Keeps track of frame reception statuses for block ack agreements
- `RecipientBlockAckAgreementProcedure` contains

  `map<pair<MACAddress, Tid>, BlockAckAgreement>`
- `BlockAckAgreement` contains

  Starting sequence number

  Buffer size

  Expiration time

  BlockAckRecord
- `BlockAckRecord` contains

  `pair<SeqNum, FragNum> → Status` (arrived or not)

Contents

Policies

- Policies answer *when* as opposed to *how*
- Meant to be easily replaceable with custom versions
- Implemented as simple modules
  - Ack policy
  - Rts / Cts policy
  - Fragmentation policy
  - Aggregation policy
  - Block ack agreement policy

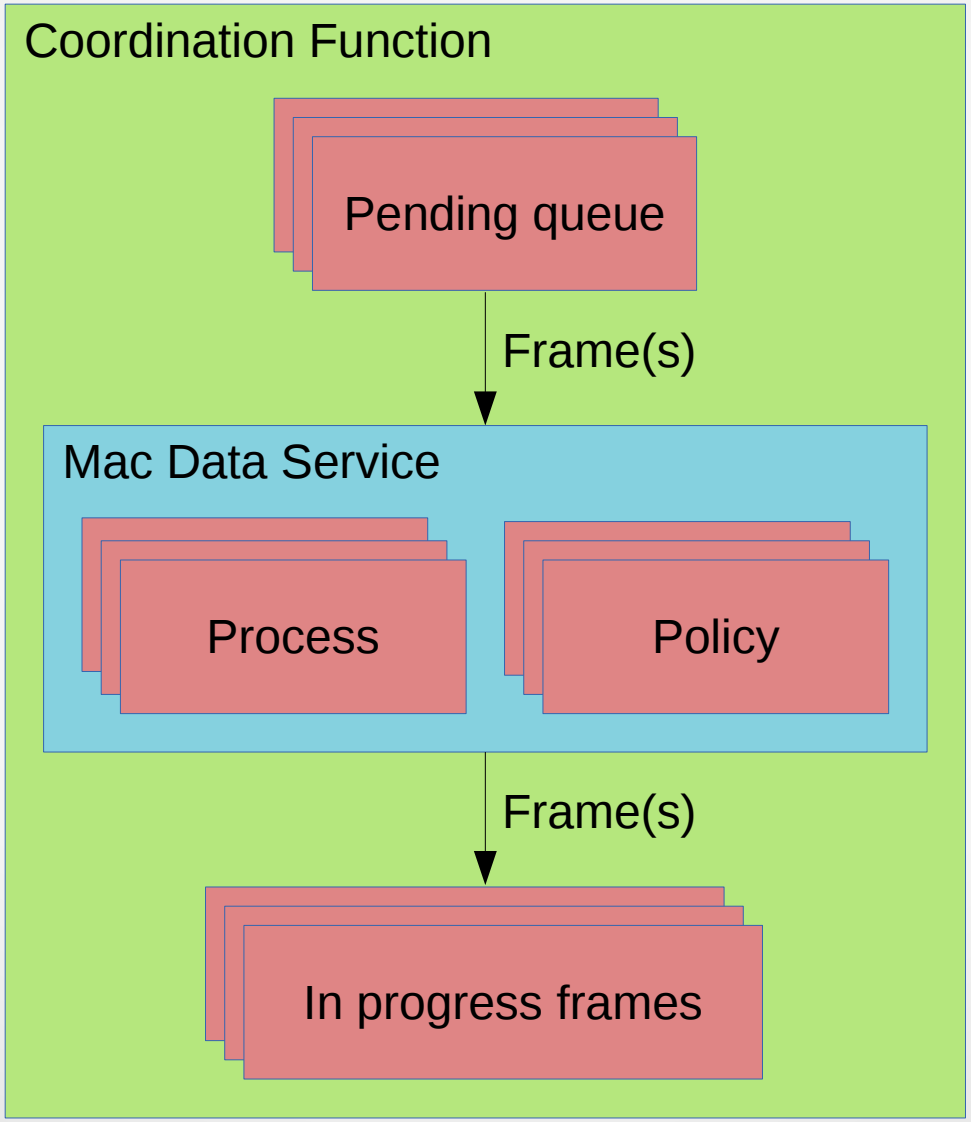
Policy Example

- `OriginatorBlockAckAgreementPolicy` determines
  - when to initiate a new agreement
  - when to terminte an existing agreement
- `OriginatorQoSAckPolicy` determines
  - ack policy subfield for outgoing data frames

    `NORMAL_ACK, BLOCK_ACK, NO_ACK`
  - when to send `BlockAckReq`

Contents

Data Flow at the Originator



As Defined in the Standard

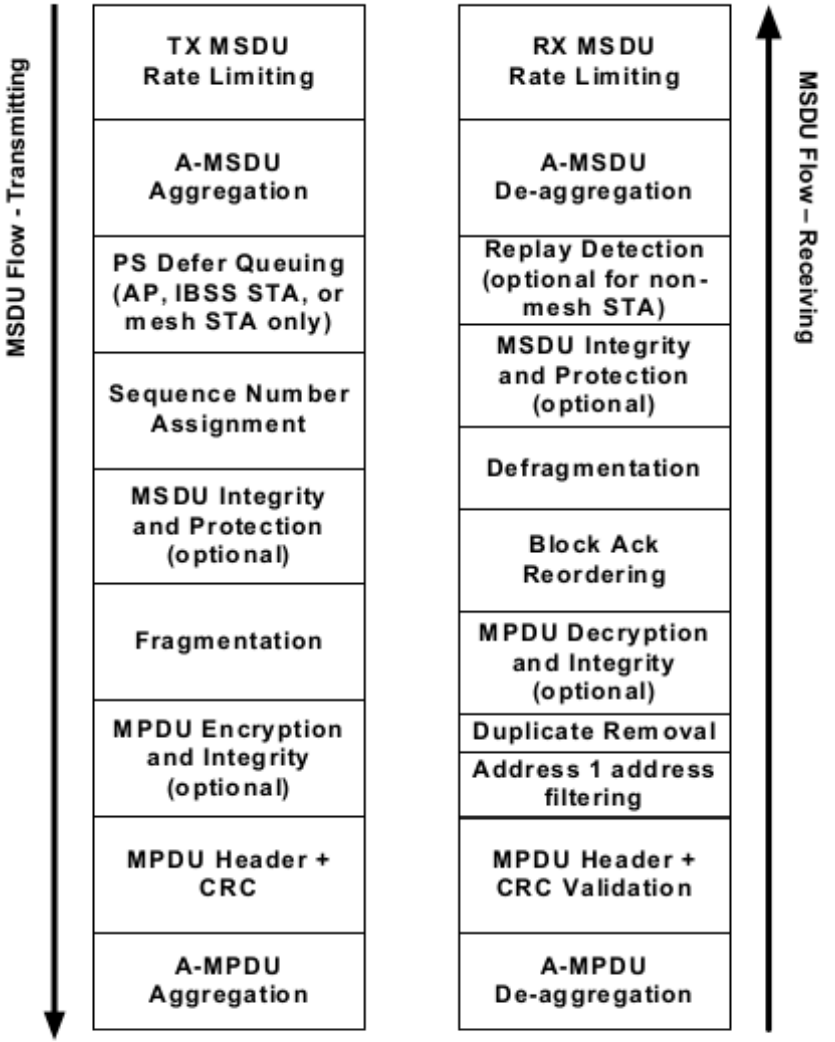


Figure 5-1—MAC data plane architecture

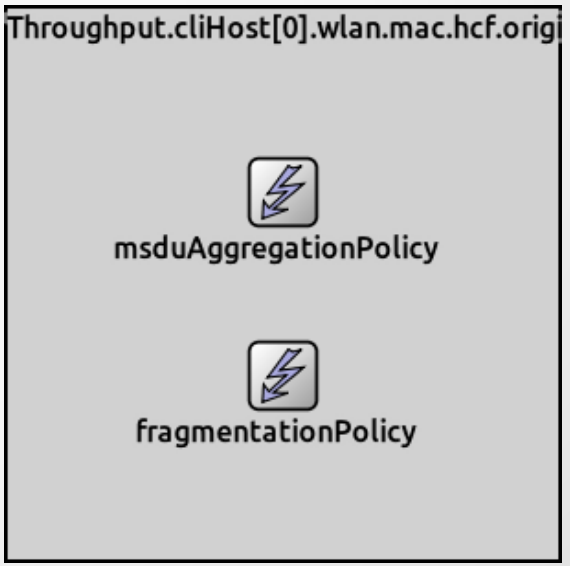
Correspondence to the Standard

```
class INET_API OriginatorQoSMacDataService : public IOriginatorMacDataService, public cSimpleModule
{
    protected:
        // Figure 5-1—MAC data plane architecture
        // MsduRateLimiting *msduRateLimiting = nullptr;
        ISequenceNumberAssignment *sequenceNumberAssigment = nullptr;
        // MsduIntegrityAndProtection *msduIntegrityAndProtection = nullptr;
        // MpduEncryptionAndIntegrity *mpduEncryptionAndIntegrity = nullptr;
        // MpduHeaderPlusCrc *mpduHeaderPlusCrc = nullptr;
        IFragmentationPolicy *fragmentationPolicy = nullptr;
        IFragmentation *fragmentation = nullptr;
        IMsduAggregationPolicy *aMsduAggregationPolicy = nullptr;
        IMsduAggregation *aMsduAggregation = nullptr;
        // PsDeferQueueing *psDeferQueueing = nullptr;
        // AMpduAggregation *aMpduAggregation = nullptr;
```

```
OriginatorQoSMacDataService::Fragments* OriginatorQoSMacDataService::extractFramesToTransmit
{
    // if (msduRateLimiting)
    //    txRateLimitingIfNeeded();
    Ieee80211DataOrMgmtFrame *frame = nullptr;
    if (aMsduAggregationPolicy)
        frame = aMsduAggregateIfNeeded(pendingQueue);
    if (!frame)
        frame = pendingQueue->pop();
    // PS Defer Queueing
    if (sequenceNumberAssigment)
        frame = assignSequenceNumber(frame);
    // if (msduIntegrityAndProtection)
    //    frame = protectMsduIfNeeded(frame);
    Fragments *fragments = nullptr;
    if (fragmentationPolicy)
        fragments = fragmentIfNeeded(frame);
    if (!fragments)
        fragments = new Fragments({frame});
```

Implementation

- Implemented as compound modules using C++ classes derived from cModule
- Contains processes implemented as C++ classes
  - Sequence number assignment / Duplicate removal
  - Fragmentation / Defragmentation
  - Aggregation / Deaggregation
  - Block Ack reordering
- Contains policies as submodules
  - Fragmentation policy
  - Aggregation policy



Originator mac data service

Contents

Correspondence to the Standard

frame-sequence =

([**CTS**] (**Management** +*broadcast* | **Data** +*group*)) |

([**CTS** | **RTS CTS** | **PS-Poll**] {frag-frame **ACK**} last-frame **ACK**) |

(**PS-Poll ACK**) |

([**Beacon** +*DTIM*] {cf-sequence} [**CF-End** [+*CF-Ack*]])|

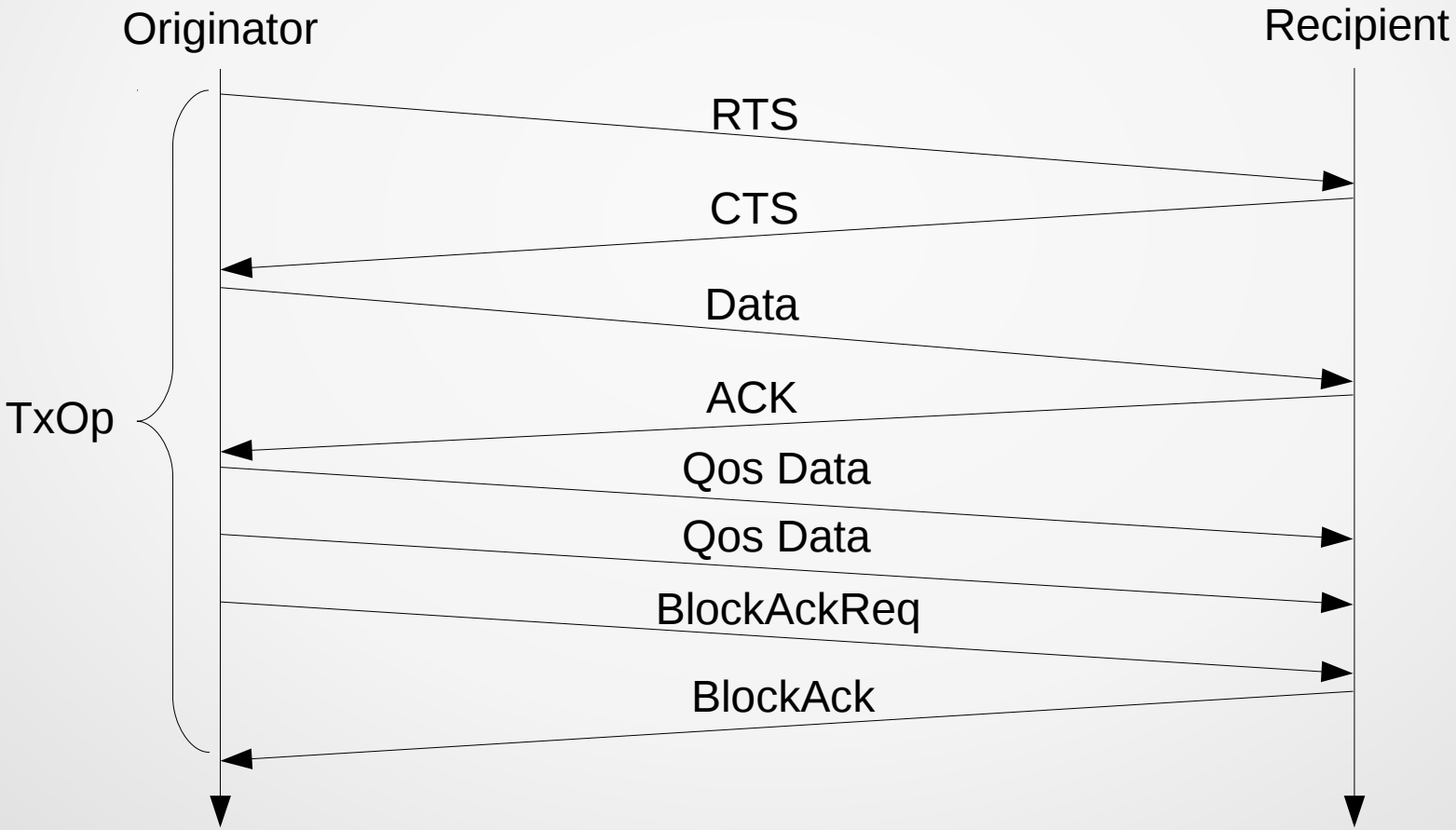
hcf-sequence |

mcf-sequence;

```
DcfFs::DcfFs() :
    // Excerpt from G.2 Basic sequences (p. 2309)
    // frame-sequence =
    //   ( [ CTS ] ( Management + broadcast | Data + group ) ) |
    //   ( [ CTS | RTS CTS] {frag-frame ACK } last-frame ACK )
    AlternativesFs({new SequentialFs({new OptionalFs(new SelfCtsFs(), OPTIONALFS_PREDICATE(isSelfCtsNeeded)),
                                      new AlternativesFs({new ManagementFs(), new DataFs()},
                                                         ALTERNATIVESFS_SELECTOR(selectMulticastDataOrMgmt))}),
                    new SequentialFs({new OptionalFs(new AlternativesFs({new SelfCtsFs(), new SequentialFs({new RtsFs(), new CtsFs()})},
                                                                        ALTERNATIVESFS_SELECTOR(selectSelfCtsOrRtsCts)),
                                                     OPTIONALFS_PREDICATE(isCtsOrRtsCtsNeeded)),
                                      new RepeatingFs(new SequentialFs({new FragFrameFs(), new AckFs()}),
                                                      REPEATINGFS_PREDICATE(hasMoreFragments)),
                                      new SequentialFs({new LastFrameFs(), new AckFs()})})},
                   ALTERNATIVESFS_SELECTOR(selectDcfSequence))
{
}
```

Hcf Frame Sequence Example

RTS CTS (**Data**+*individual*) **ACK** (**Data** +*QoS* +*individual* +*block-ack*) (**Data** +*QoS* +*individual* +*block-ack*) **BlockAckReq BlockAck**



Implementation

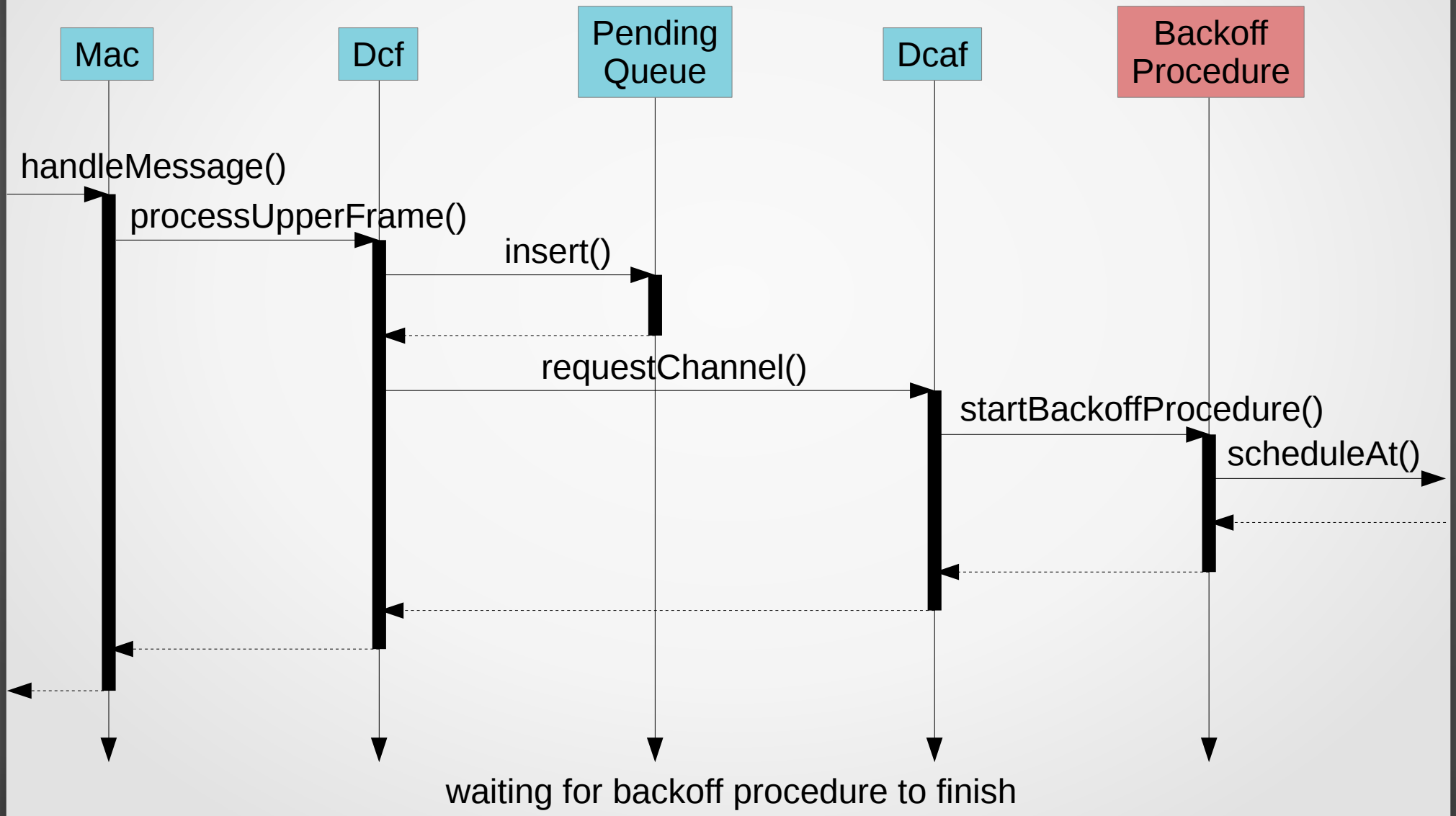
- Implemented as C++ classes
- Coordination functions have their own frame sequences directly corresponding to the 802.11 Annex G. (normative)
  - `DcfFs`, `PcfFs`, `HcfFs`, and `McfFs`
- Building blocks
  - `SequentialFs`, `OptionalFs`, `RepeatingFs`, `AlternativeFs`
  - `FragFrameFs`, `AckFs`, `SelfCtsFs`, etc.

Contents

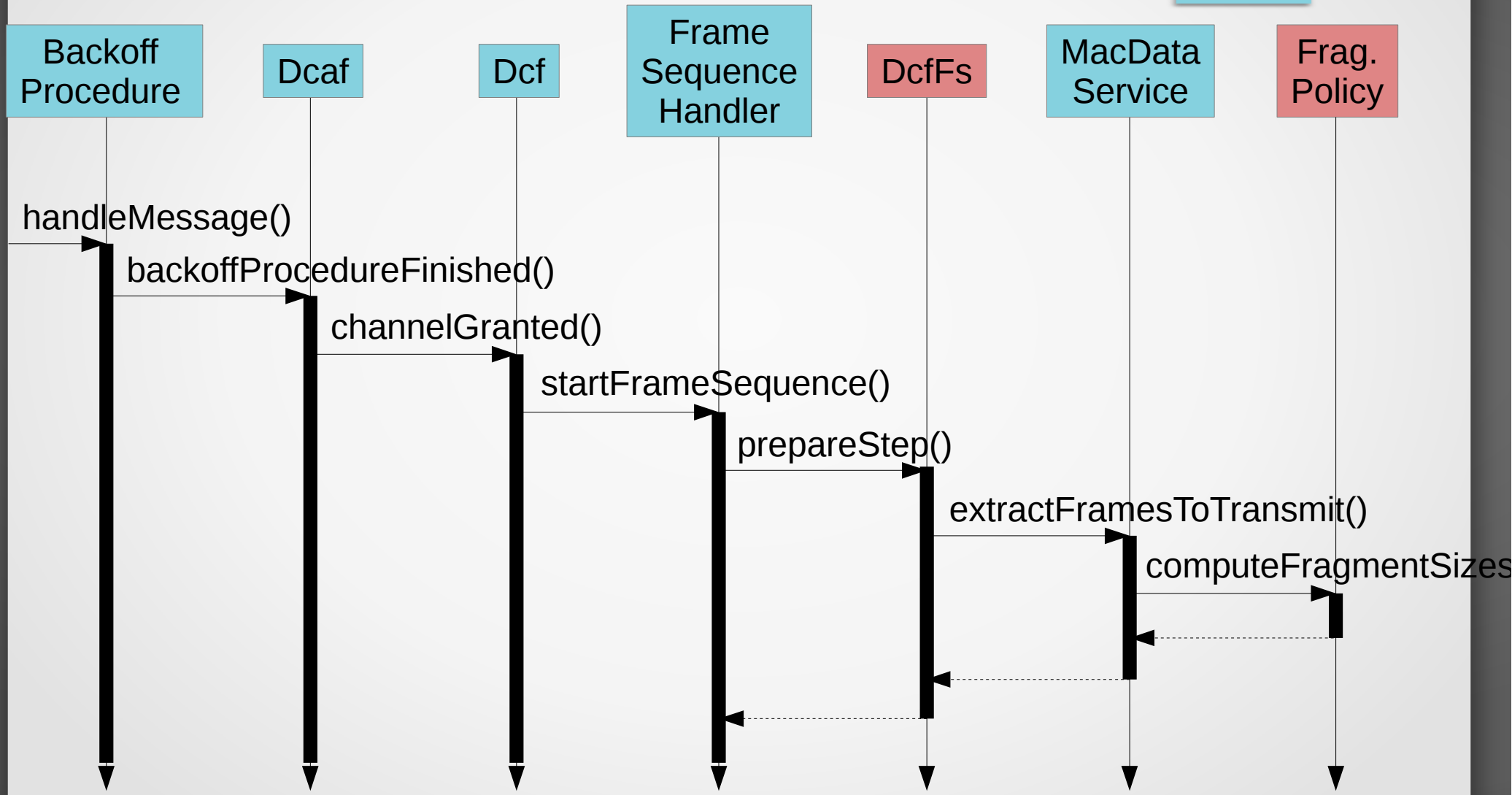
Data ACK Frame Sequence Example

- Processing the **Data** Frame at the Originator
  - **Data** Frame Arrived → Backoff Procedure Started
  - Backoff Procdedure Finished → **Data** Frame Extracted
  - **Data** Frame Extracted → **Data** Transmission Started
  - **Data** Transmission Finished → Waiting for **ACK** Started
- Processing the **Data** Frame at the Recipient
- Processing the **ACK** Frame at the Originator

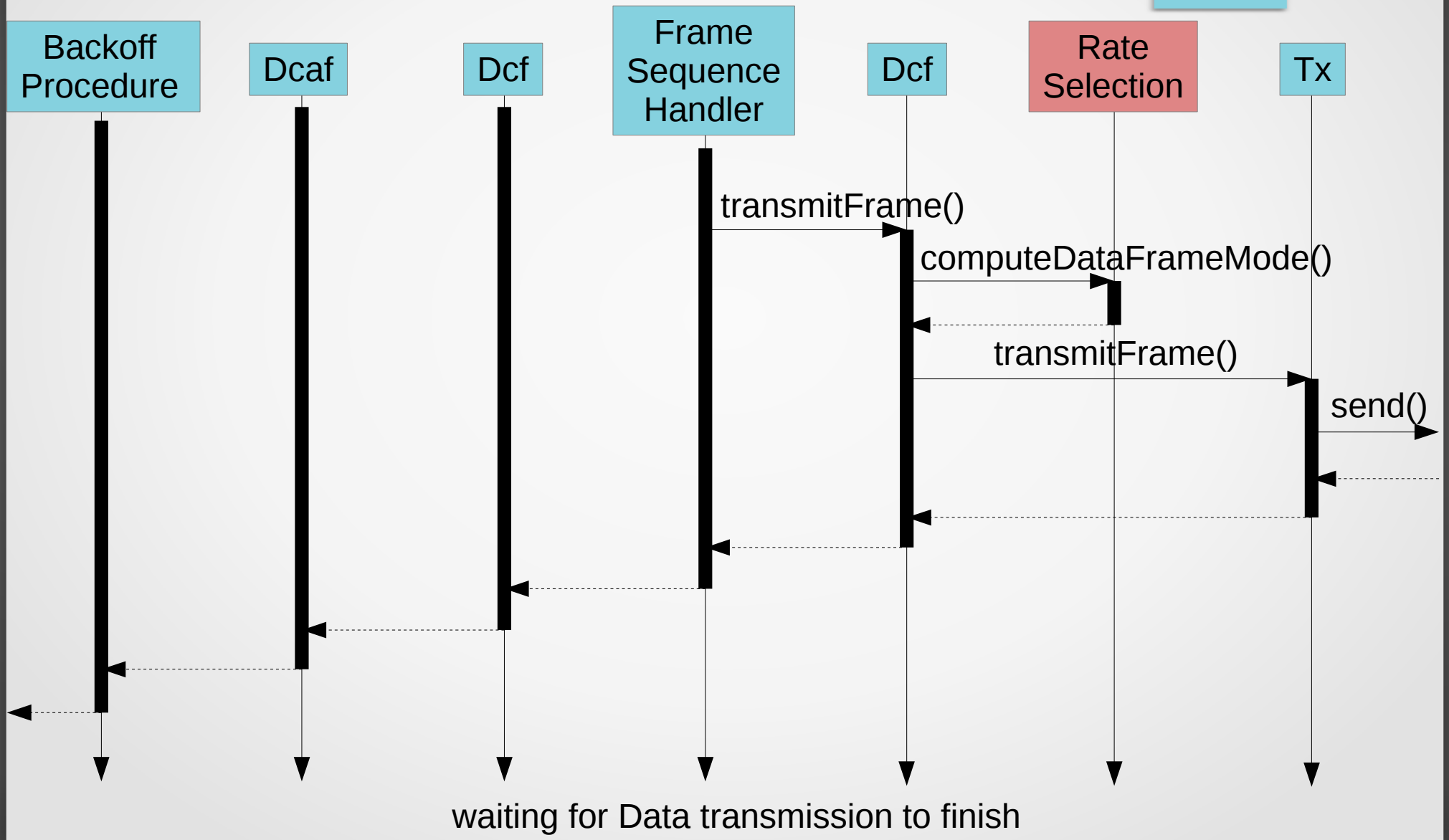
Data Frame Arrived → Backoff Procedure Started



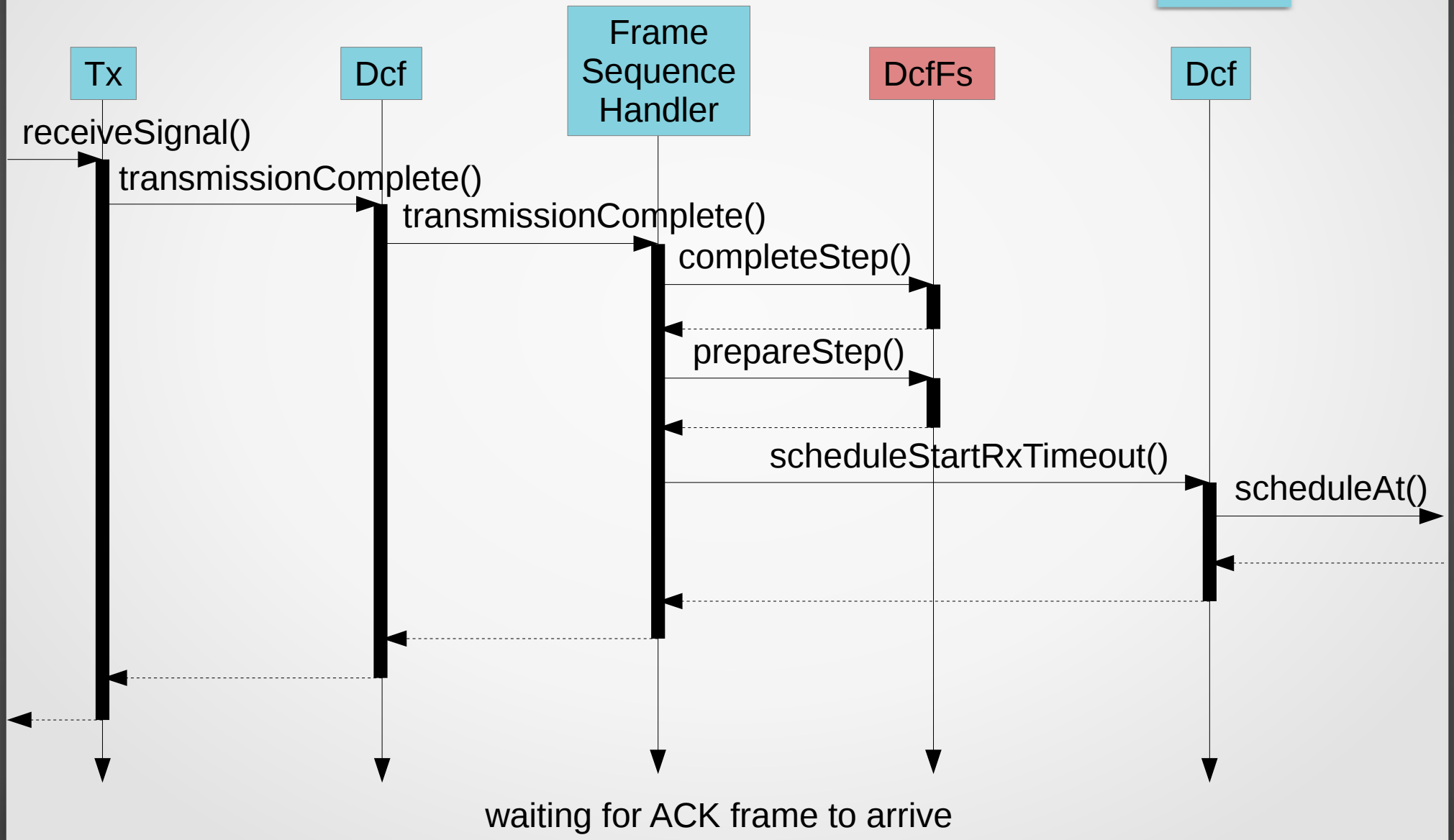
Backoff Procdedure Finished → **Data** Frame Extracted



Data Frame Extracted → **Data** Transmission Started



Data Transmission Finished → Waiting for **ACK** Started



Frequently Asked Questions

- When will it be available?
  - Needs more work on: validation, logging, visualization
- Is it compatible with the current version?
  - It's meant to be (`Ieee80211CompatibleMac`)
- What features are implemented?
  - New: block ack, MSDU aggregation
  - Still missing: Hcca, Pcf, Mcf, MPDU aggregation, frame lifetime, etc.
- Can I build a simplified MAC?
  - Yes (work in progress)

Questions and Answers

Thank you for your kind attention!