Repository Structure

MAC-Testbed	
+ PhyRadio[M]	Core implementation of MAC-agnostic PHY
+ PhyRadio2[M]	Core implementation of CSMA supporting PHY
+ PhyRadioMsg.h	Basic header definitions for PHY
+ Phy*, SignalStrength*, CarrierSense*	nesC interfaces for PhyRadio and PhyRadio2
+ ptimer	High performance high accuracy timer implementation
+ macs	
- polling	Full implementation of Polling MAC layer
- s-mac	Half-working s-mac port
testbed-apps	
+ AckCSMAHeadTest	Cluster Head test program for the CSMA MAC with ACK to data packets
+ AckCSMANodeTest	Cluster Node test program for the CSMA MAC with ACK to data packets
+ CSMAHeadTest	Cluster Head test program for the CSMA MAC without ACK
+ CSMANodeTest	Cluster Node test program for the CSMA MAC without ACK
+ PHYTestRecv	Early PHY test program that receives data
+ PHYTestSend	Early PHY test program that sends data
+ PhyCSMATestRecv	Early PHY test program that receives data
+ PhyCSMATestSend	Early PHY test program that uses CSMA to send data
+ PollHeadTest	Cluster Head test program for the polling MAC
+ PollNodeTest	Cluster Node test program for the polling MAC
+ PrecisionTimerTest	Test program comparing Performance Timers (ptimer) with tinyOS standard timers
* config.h	Configuration file to fine-tune some of the parameters of the tests. This file can be found in most *Test directories
* *Msg.h	Header file containing the structure definition of the packets used for that test or that MAC. These have to be synchronized across the same test (i.e. AckCSMANodeTest and AckCSMAHeadTest must have the same CSMAMsg.h)

testbed-docs	
+ csmadata_eval.m	MATLAB support file for csmatestresults.m
+ polldata_eval.m	MATLAB support file for polltestresults.m
+ csmatestresults.m	MATLAB program to analyse a cleaned csv test results file generated by the *CSMA*Test programs
+ polltestresults.m	MATLAB program to analyse a cleaned csv test results file generated by the Poll*Test programs
+ *.csv	Comma Separated Value (CSV) raw test results files

Generating and Analysing Test Results

The test programs output test results over the serial port in a CSV-like format. This has to be captured from the beginning of the test, where the first column shows a '1' to the end of the test, usually a '5' in the first column.

The captured data has then to be cleaned up using the following process:

1) Remove all non-numerical strings, such as: “CRC”, “HPL”, “Battery”, “Device rebooted”, etc.
2) Open with Excel or a similar program and save again as a .csv file (NOT XLS!!!). This will basically pad all lines with commas so that every line has the same number of commas.

The cleaned up csv file can then be imported as usual in Matlab. The csmatestresults and polltestresults Matlab programs can then be used to generate an Excel XLS spreadsheet containing the analysed test results. They are used as follows:

> csmatestresults(1, ack_data, 'ack_csma_-15dBm_noise.xls', 'A2')

> polltestresults(1, polling_data, 'polling_9_-25dBm_noise.xls', 'A2')

The format is:

> <program>(<test number>, <matlab data identifier/variable>, <output xls file>, <output row/column in xls file>)

Interfaces

BackoffControl

```
command result_t enableBackoff();
command result_t disableBackoff();
```

Enable or disable CSMA mode. If disabled, packets will be sent out immediately no matter if the channel is busy.

```
command result_t setMode(uint8_t random);
```

Set to 1 for random backoff (limits set with setRandomLimits) or 0 for static backoff (always the value specified by setBackoffTime)

```
command result_t setRandomLimits(int16_t min, int16_t max);
command result_t setBackoffTime(int16_t time);
```

Set the backoff time or time limits in microseconds.

```
command result_t setRetries(uint8_t retr);
```

Set the maximum number of retries for every packet before the packet transmission fails if no free channel is detected.

CarrierSense

```
command result_t start();
event result_t channelIdle();
event result_t channelBusy();
```

Start a channel CA. If the channel is idle, the event channelIdle() will be called, otherwise the event channelBusy() occurs.

```
command result_t setThreshold(int8_t threshold);
```

Set the CCA Threshold (for both CarrierSense and CSMA), in dBm.

```
command result_t setMode(int8_t mode);
```

Set the CCA Mode (refer to the CC2420 datasheet; for both CarrierSense and CSMA).

PhyComm

```
command result_t txPkt(void* packet, uint8_t length);
```

Transmit a packet.

```
command result_t reTxPkt();
```

Retransmit the last sent packet.

```
event result_t txPktDone(void* packet, uint8_t error);
```

Packet transmission finished event. If error is not 0, an error occurred and the packet is likely not to have been transmitted.

```
command result_t cancelTxPkt();
```

Only works in CSMA mode. Will cancel a packet transmission if it's currently being backed off.

```
event result_t startSymDetected(void* packet);
```

Event notifying the reception of a SFD (start of frame delimiter). Due to a bug in the CC2420 hardware, this event will be signalled twice for one reception (also for EFD)

```
event void* rxPktDone(void* packet, uint8_t error);
```

Event notifying that a packet has been received. If error is not 0, an error occurred during the reception and the data is not valid.

PhyRadioControl

```
command result_t setFrequency(uint16_t freq);
```

Set the operating frequency, in MHz.

```
command result_t setPower(uint8_t power);
```

Set the operating power (refer to CC2420 datasheet for values and their meanings)

PhyState

```
command result_t idle();
```

Put radio into idle (wake up from sleep). Whenever the radio is idle again, the startDone callback occurs again.

```
command result_t sleep();
```

Put radio into sleep mode. Whenever the radio is in sleep mode, the stopDone callback occurs again.

SignalStrength

```
command int8_t getRSSI();
```

Read out the current RSSI value from the CC2420 Radio. The value is in dBm.

PollHeadComm

```
command result_t requestData(uint8_t u_node_id, void *data, uint8_t
length);
```

Request data from the specified node id, and the specified request packet “data”.

```
event result_t requestDataDone(uint8_t id, void *data, uint8_t error);
```

Data request finished for node “id”. If there was an error (timeout, …), error will not be 0. Otherwise the data will be correct..

```
command result_t cancelRequest();
```

Cancel a polling request. This will cause a requestDataDone() event.

```
command result_t setSleepInterval(uint32_t sleep_jiffies);
```

Set the sleep interval to be broadcasted to the nodes in the beacon message.

PollNodeComm

```
event result_t dataRequested(void *data);
```

A data request has been received. The packet is available as 'data'.

```
event result_t ackReceived(void *data);
```

An ACK has been received for the data we've sent.

```
command result_t txData(void *data, uint8_t length);
```

Transmit the data to the head after a data request.

```
event result_t dataTxFailed();
```

Data transmission failed (no ACK received, ...)