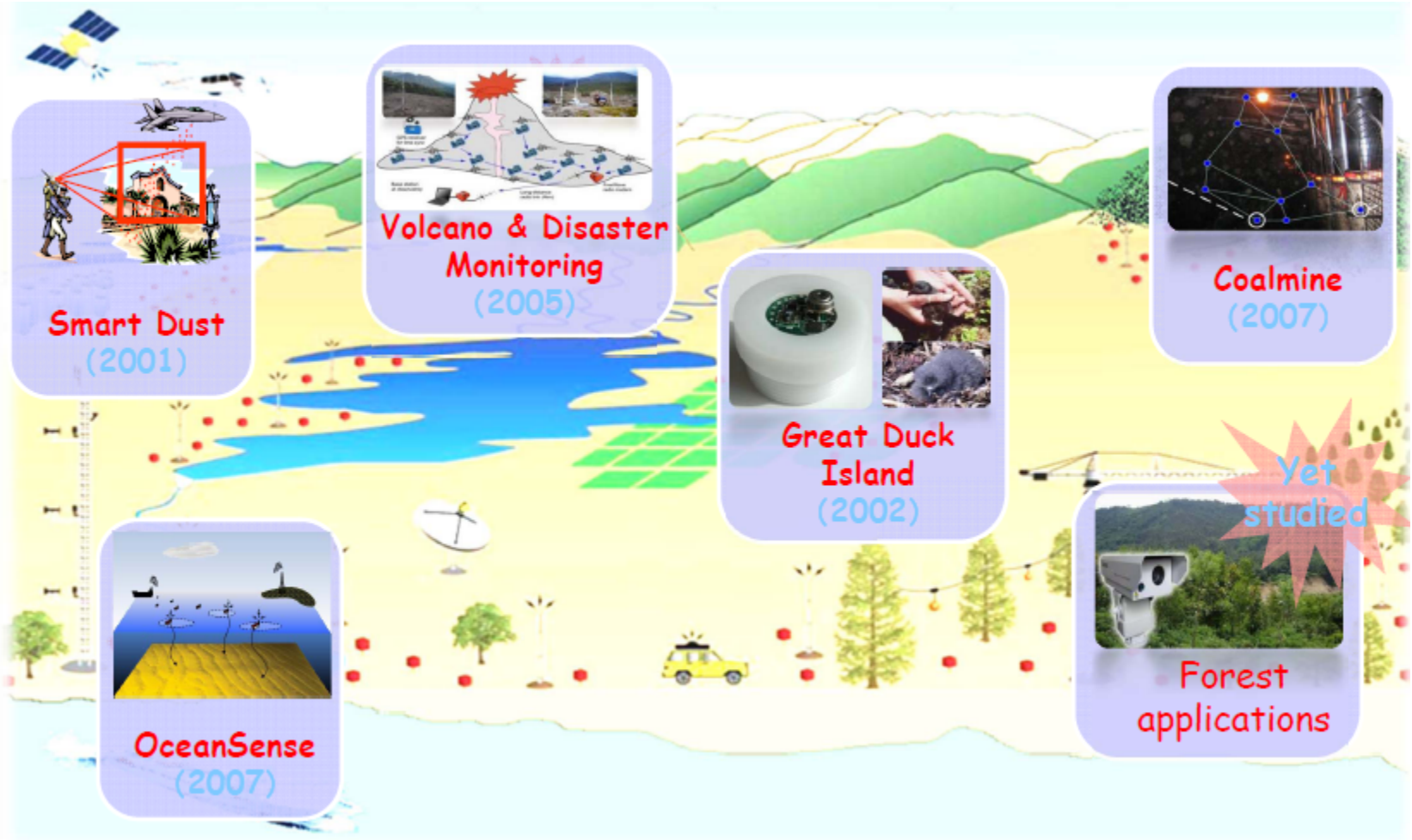
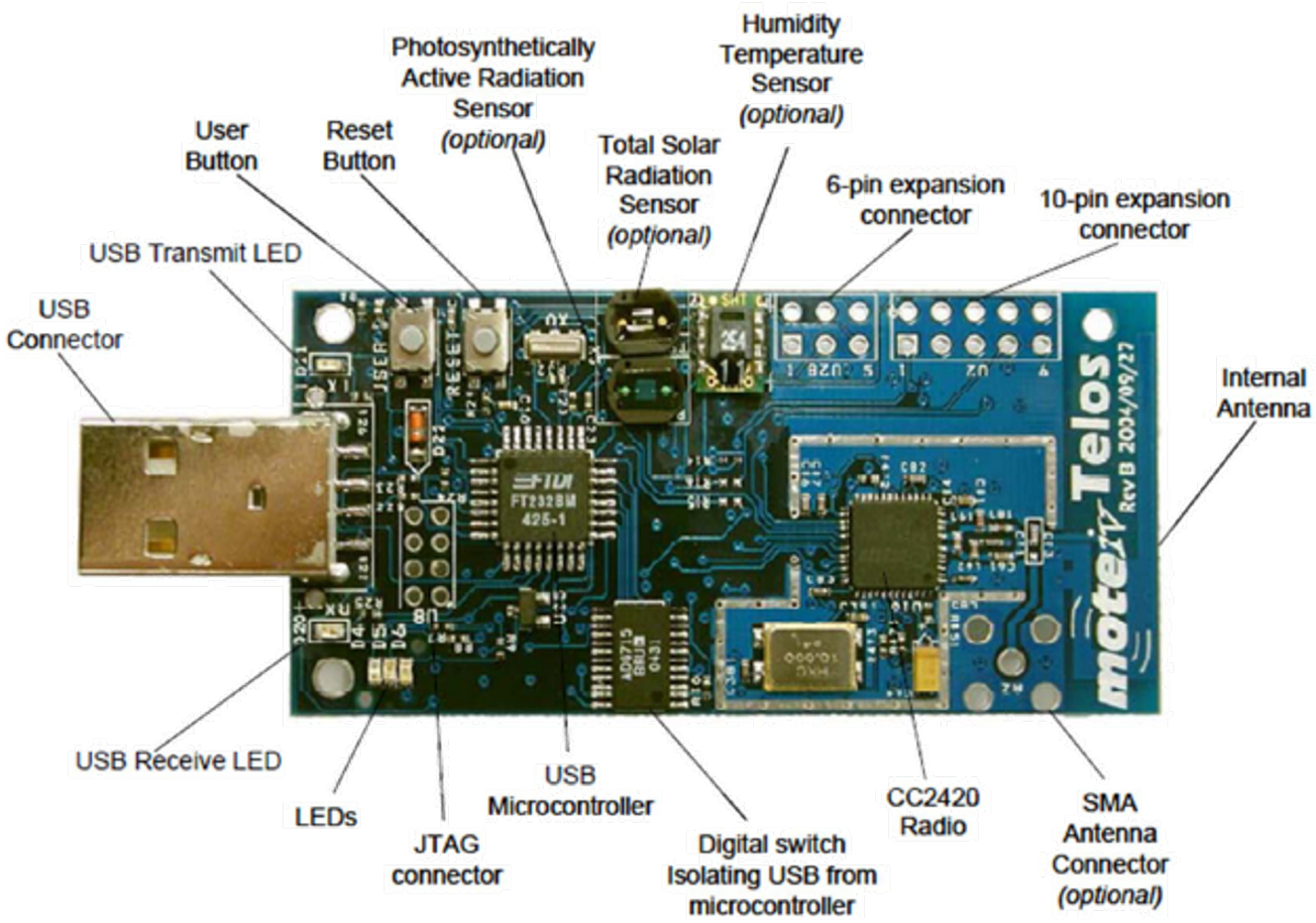
CSE 220 – C Programming

C Programming in IoT

Wireless Sensor Networks



Wireless Sensors - TelosB



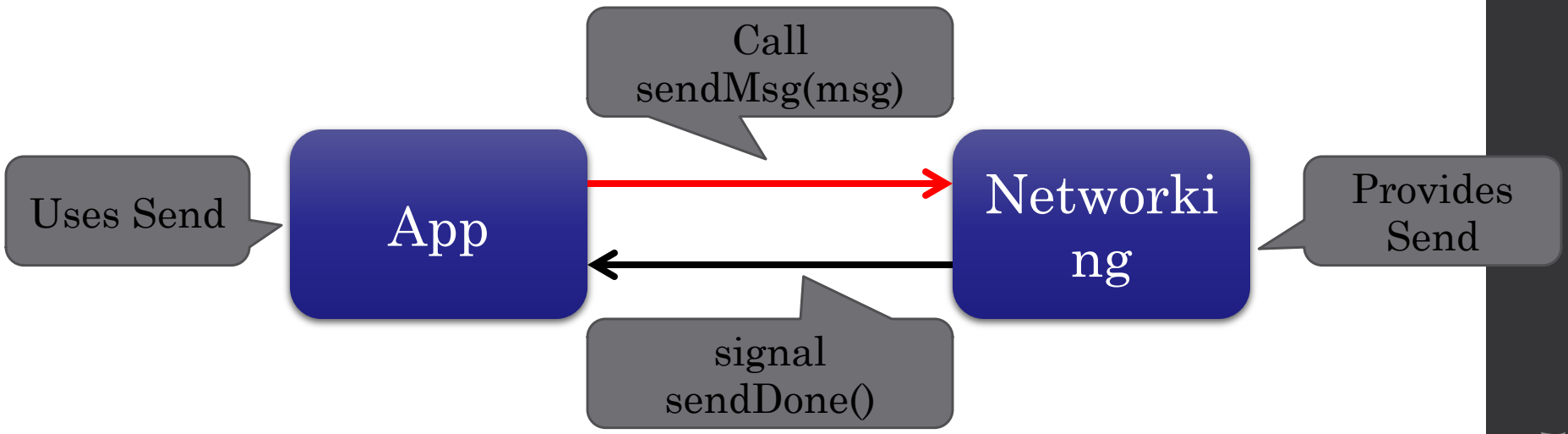
nesC/TinyOS

- ***nesC***
- ***Split-phase*** Execution
  - **Command**, active function calls
  - **Event**, passive function calls

www.tinyos.net

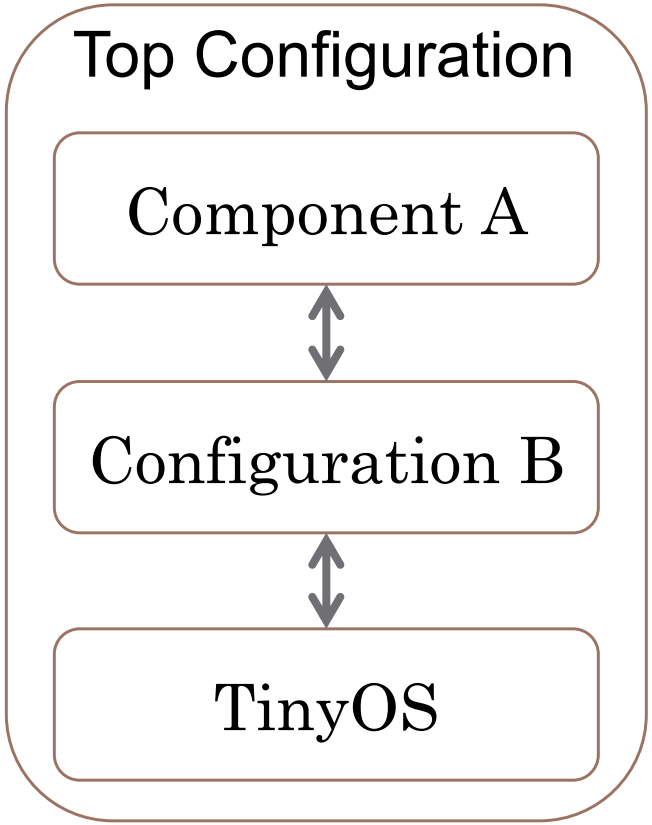
nesC/TinyOS

- **Modules/Components**
  - ***Uses*** and ***provides*** interfaces.
- **Interface** declares commend and event.



nesC/TinyOS

- **Configuration**
  - Component binding
  - Configuration nesting
- nesC preprocessor



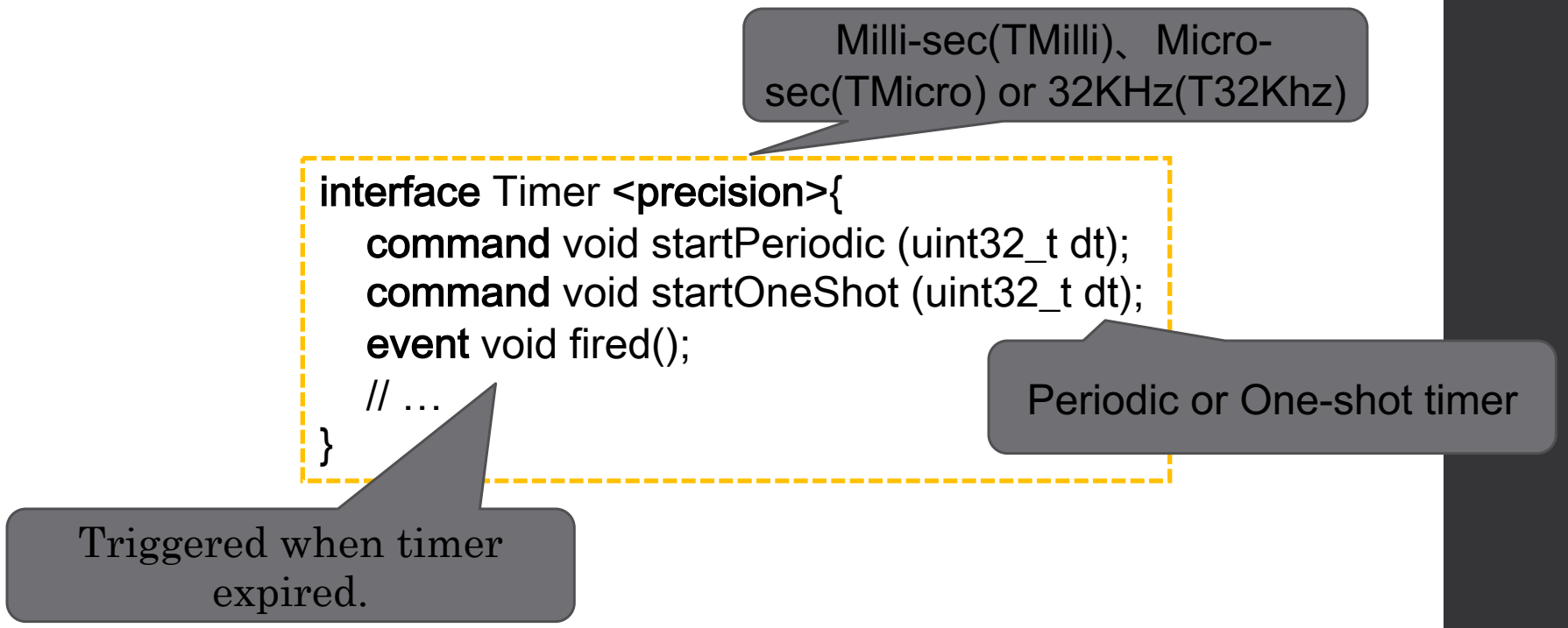
Example

- Printf_to_Radio
  - Local variable ***counter***, increasing periodically
  - Each ***counter*** change, print ***counter*** value, then broadcast a message that contains ***ID*** and ***counter***
  - Print received ***ID*** and ***counter*.**

Print-to-Radio

- Components
  - Timer
    - TinyOS Timer library
  - Radio
    - TinyOS Point-to-point communication library
    - Active Message interface duplicates radio recourse
      - TCP/UDP port
  - Print

Timer Interface



Timer

```
#include <Timer.h>

module PrintToRadioC  {
    uses interface Boot;
    uses interface Timer<TMilli> as Timer0;
} implementation {
    uint16_t counter = 0;
    event void Boot.booted() {
        call Timer0.startPeriodic(1000);
    }
    event void Timer0.fired() {
        counter++;
    }
}
```

Tmilli Timer

Triggered when MCU is booted.

Communication Interface

```
interface AMSend {
  command error_t send(am_addr_t addr,message_t* msg,uint8_t len);
  event void sendDone(message_t* msg,error_t error);
  // …
}
```

Radio sends message.

Triggered when msg is successfully transmitted.

```
interface Packet {
  command void* getPayload(message_t* msg, uint8_t len);
  // …
}
```

Return a pointer pointing to the payload of the message.

Receive Interface

```
interface Receive {
event message_t* receive(message_t* msg,void* payload, uint8_t len);
// …
}
```

Triggered when a message is received.

Radio Initialization

```
module PrintfToRadioC {
  uses {
    // …
    interface SplitControl as AMControl;
    interface Packet;
    interface AMSend;
    interface Receive;
  }
}
implementation {
  //…
  event void Boot.booted() {
    call AMControl.start();
  }
  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS)
      call CountTimer.startPeriodic(1000);
    else
      call AMControl.start();
  }
  //…
}
```

Initialize a radio

Retry after a failure

Send and Receive

```
#include “printf.h”
//…
implementation {
    bool busy = FALSE;  message_t pkt;
    //...
    event void Timer0.fired() {
        //...
        if (!busy) {
            BlinkToRadioMsg* btrpkt = (PrintToRadioMsg*)(call Packet.getPayload(&pkt, sizeof (PrintToRadioMsg)));
            btrpkt->nodeid = TOS_NODE_ID;
            btrpkt->counter = counter;
            if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(PrintToRadioMsg)) == SUCCESS) {
              busy = TRUE;
            }
        }
    }
    event void AMSend.sendDone(message_t* msg, error_t error) {  if (&pkt == msg) {  busy = FALSE;}  }
    event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
        if (len == sizeof(PrintToRadioMsg)) {
            PrintToRadioMsg* btrpkt = (PrintToRadioMsg*)payload;
            printf(“%d,%d\n”, btrpkt->nodeid, btrpkt->counter);
            printfflush();
        }
        return msg;
    }
}
```

```
typedef nx_struct {
    nx_uint16_t nodeid;
    nx_uint16_t counter;
} PrintToRadioMsg;
```

Sending status, message buffer

Payload pointer

Change sending status

Change sending status

Print and return msg.

Configuration

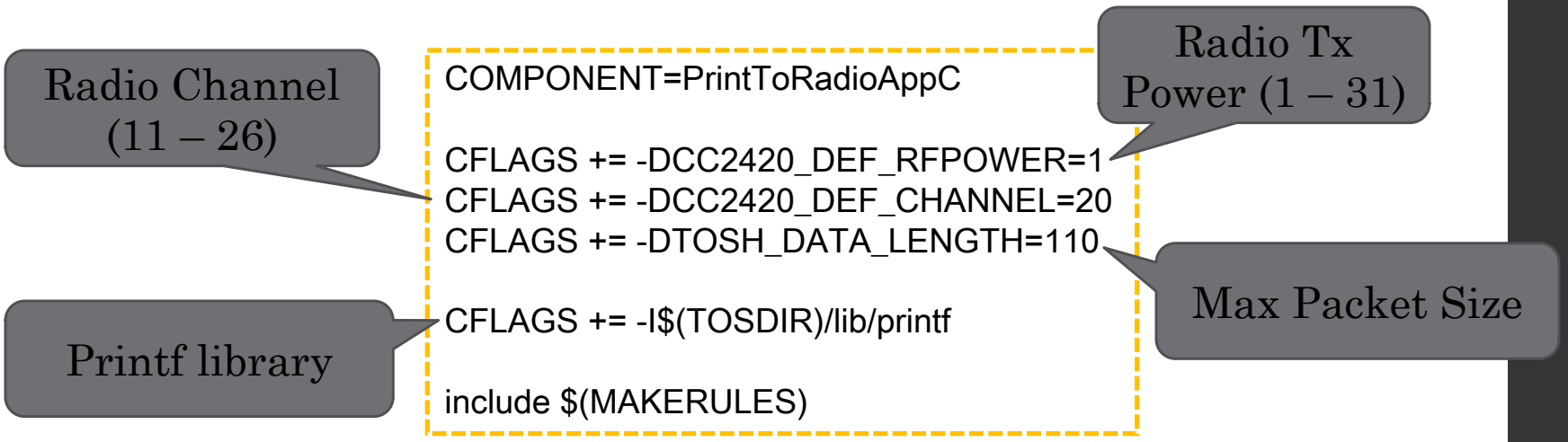
```
configuration PrintfToRadioAppC {
}
implementation {
  components MainC;
  components PrintToRadioC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_PRINTTORADIO);
  components new AMReceiverC(AM_PRINTTORADIO);

  App.Boot -> MainC;
  App.Timer0 -> Timer0;
  App.Packet -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.AMSend -> AMSenderC;
  App.Receive -> AMReceiverC;
}
```

```
enum {
  AM_PRINTTORADIO = 6,
};
```

Network port

Makefile


Radio Channel (11 – 26)
Printf library
Radio Tx Power (1 – 31)
Max Packet Size
COMPONENT=PrintToRadioAppC
CFLAGS += -DCC2420_DEF_RFPOWER=1
CFLAGS += -DCC2420_DEF_CHANNEL=20
CFLAGS += -DTOSH_DATA_LENGTH=110
CFLAGS += -I$(TOSDIR)/lib/printf
include $(MAKERULES)

Program Organization