Wireless Sensor Networks

Chiara Buratti

c.buratti@unibo.it
+39 051 20 93147

Office Hours:
Tuesday 3 – 5 pm @ Main Building, third floor

Credits: 6

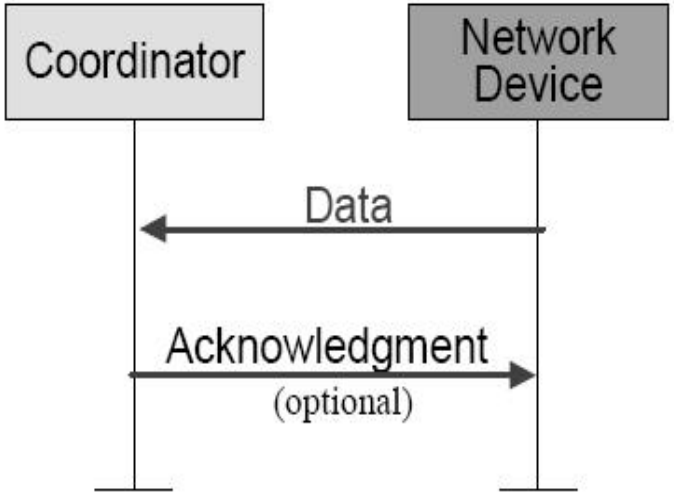
Syllabus: Laboratory Activities

1. PAN Formation
2. Data Transfer (point-to-point)
3. MAC Protocol (point-to-point)
4. NET Protocol (small network)

Chiara Buratti
DEIS, University of Bologna

Data Transfer: Non Beacon-Enabled Mode

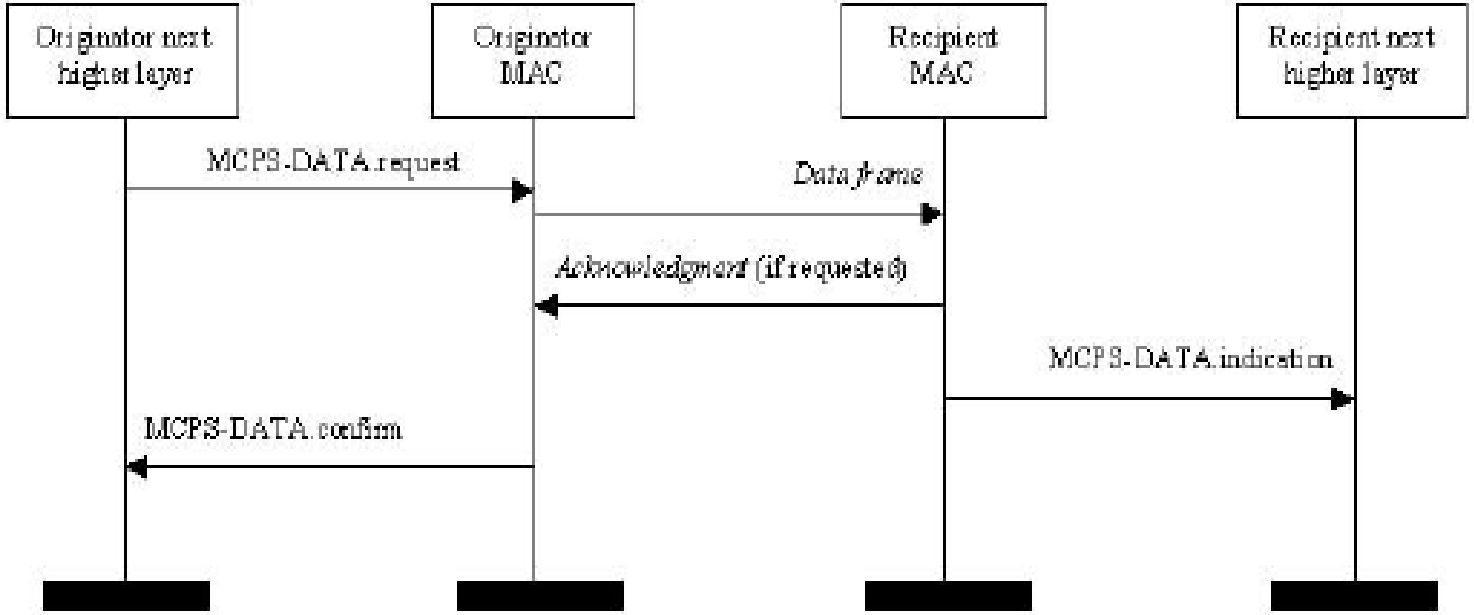
From Device to Coordinator



ACK packets are sent one TAT (192 μs) after the reception of the packet, without the use of CSMA/CA

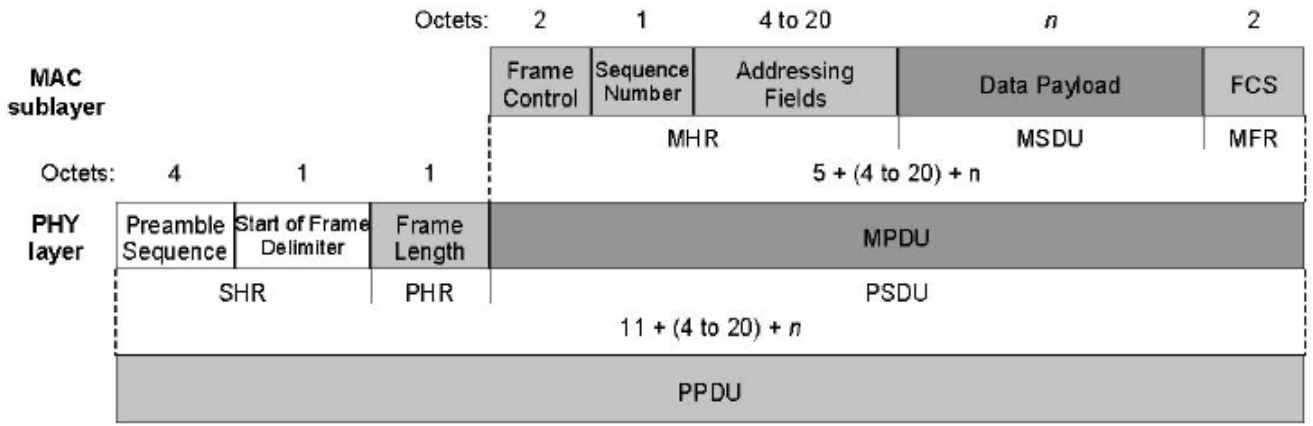
Chiara Buratti
DEIS, University of Bologna

Data Transfer: Primitives



Data Transfer: Primitives

MCPS-DATA.request (
SrcAddrMode,
SrcPANId,
SrcAddr,
DstAddrMode,
DstPANId,
DstAddr,
msduLength,
msdu,
msduHandle,
TxOptions
)



Data Transfer: Primitives

Name	Type	Valid range	Description
SrcAddrMode	Integer	0 x 00–0 x 03	The source addressing mode for this primitive and subsequent MPDU. This value can take one of the following values: 0 x 00 = no address (addressing fields omitted). 0 x 01 = reserved. 0 x 02 = 16 bit short address. 0 x 03 = 64 bit extended address.
SrcPANId	Integer	0 x 000–0 x ffff	The 16 bit PAN identifier of the entity from which the MSDU is being transferred.
SrcAddr	Device address	As specified by the SrcAddrMode parameter	The individual device address of the entity from which the MSDU is being transferred.
DstAddrMode	Integer	0 x 00–0 x 03	The destination addressing mode for this primitive and subsequent MPDU. This value can take one of the following values: 0 x 00 = no address (addressing fields omitted). 0 x 01 = reserved. 0 x 02 = 16 bit short address. 0 x 03 = 64 bit extended address.
DstPANId	Integer	0 x 0000–0 x ffff	The 16 bit PAN identifier of the entity to which the MSDU is being transferred.
DstAddr	Device address	As specified by the DstAddrMode parameter	The individual device address of the entity to which the MSDU is being transferred.
msduLength	Integer	≤ aMaxMACFrameSize	The number of octets contained in the MSDU to be transmitted by the MAC sublayer entity.
msdu	Set of octets	—	The set of octets forming the MSDU to be transmitted by the MAC sublayer entity.
msduHandle	Integer	0 x 00–0 x ff	The handle associated with the MSDU to be transmitted by the MAC sublayer entity.
TxOptions	Bitmap	0000 xxxx (where x can be 0 or 1)	The transmission options for this MSDU. These are a bitwise OR of one or more of the following: 0 x 01 = acknowledged transmission. 0 x 02 = GTS transmission. 0 x 04 = indirect transmission. 0 x 08 = security enabled transmission.

Data Transfer: Primitives

MCPS-DATA.indication (
SrcAddrMode,
SrcPANId,
SrcAddr,
DstAddrMode,
DstPANId,
DstAddr,
msduLength,
msdu,
mpduLinkQuality,
SecurityUse,
ACLEntry
)

Data Transfer: Primitives

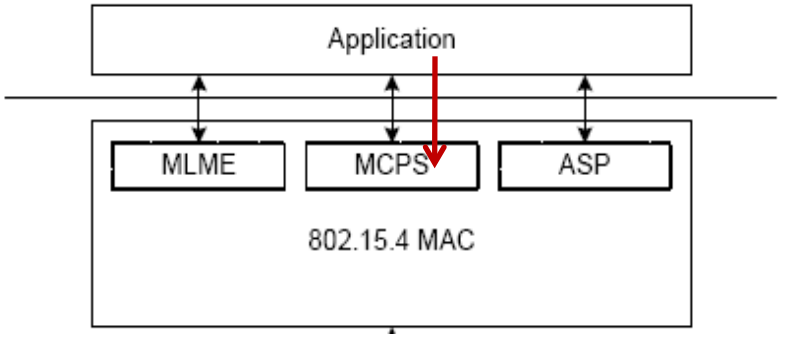
MCPS-DATA.confirm (
msduHandle,
status
)

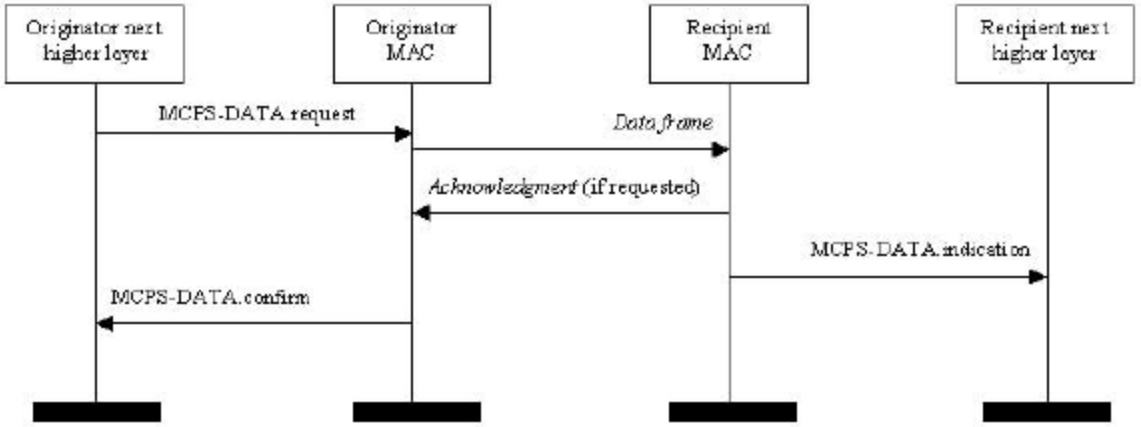
Name	Type	Valid range	Description
msduHandle	Integer	0 x 00–0 x ff	The handle associated with the MSDU being confirmed.
status	Enumeration	SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, CHANNEL_ACCESS_FAILURE, INVALID_GTS, NO_ACK, UNAVAILABLE_KEY, FRAME_TOO_LONG, FAILED_SECURITY_CHECK, or INVALID_PARAMETER	The status of the last MSDU transmission.

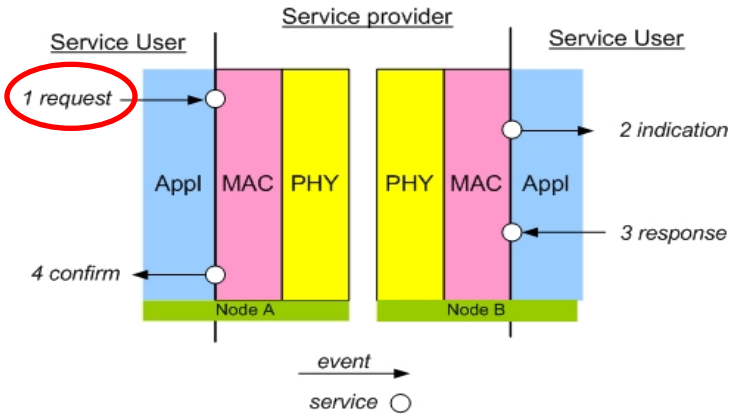
Chiara Buratti
DEIS, University of Bologna

Message Types: NWK-to-MCPS direction

Message Identifier (primNwkToMcps_t)	802.15.4 NWK to MCPS Primitives
gMcpsDataReq_c	MCPS-DATA.Request



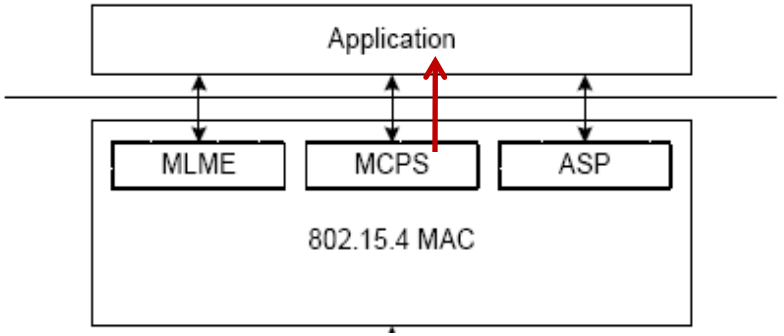


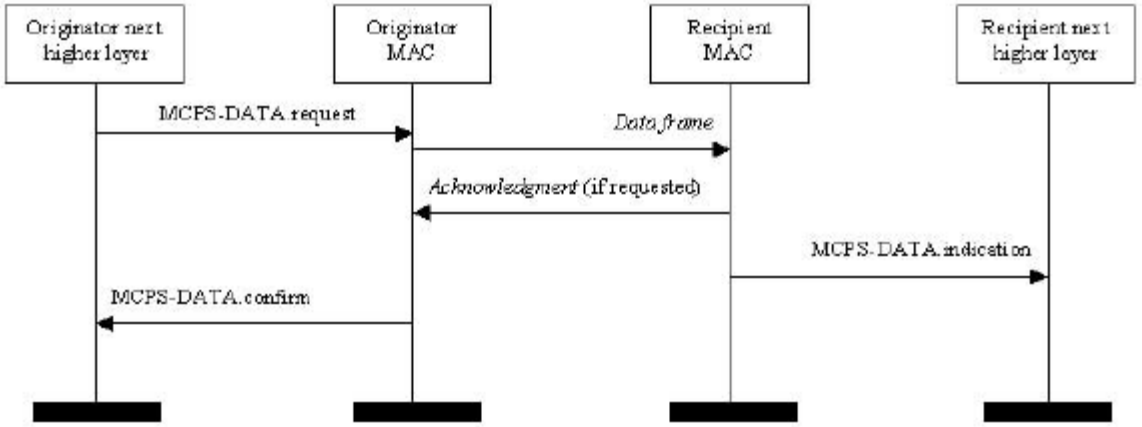


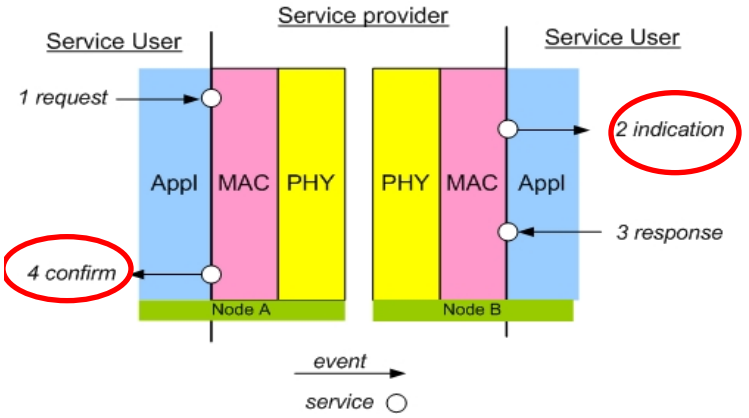
Chiara Buratti
DEIS, University of Bologna

Message Types: MCPS-to-NWK direction

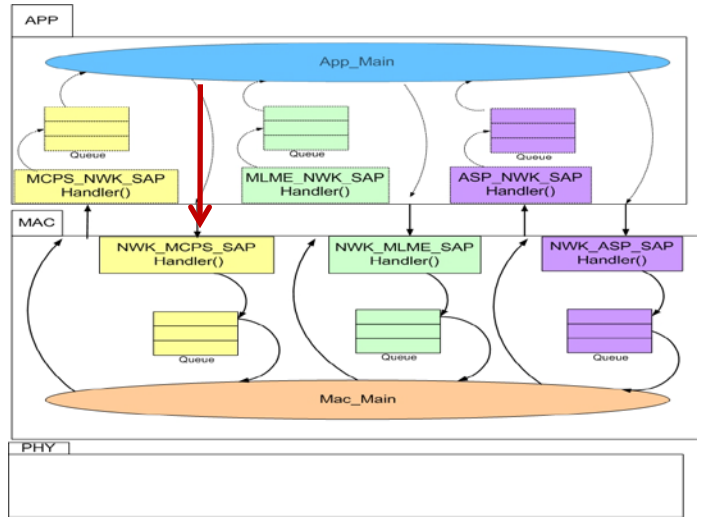
Message Identifier (primMcpsToNwk_t)	802.15.4 MCPS to NWK Primitives
gMcpsDataCnf_c	MCPS-DATA.Confirm
gMcpsDataInd_c	MCPS-DATA.Indication





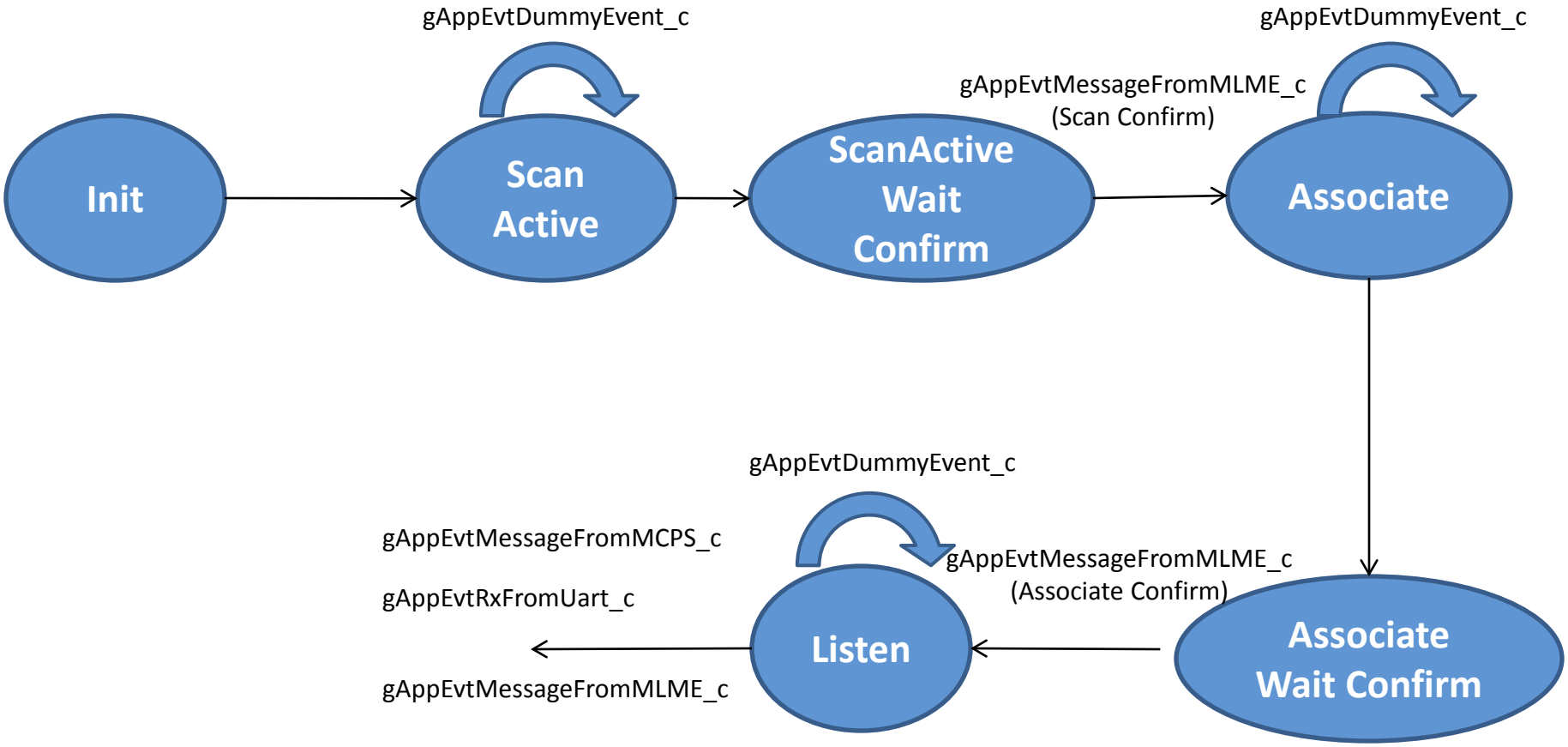


Data Transfer: An Example



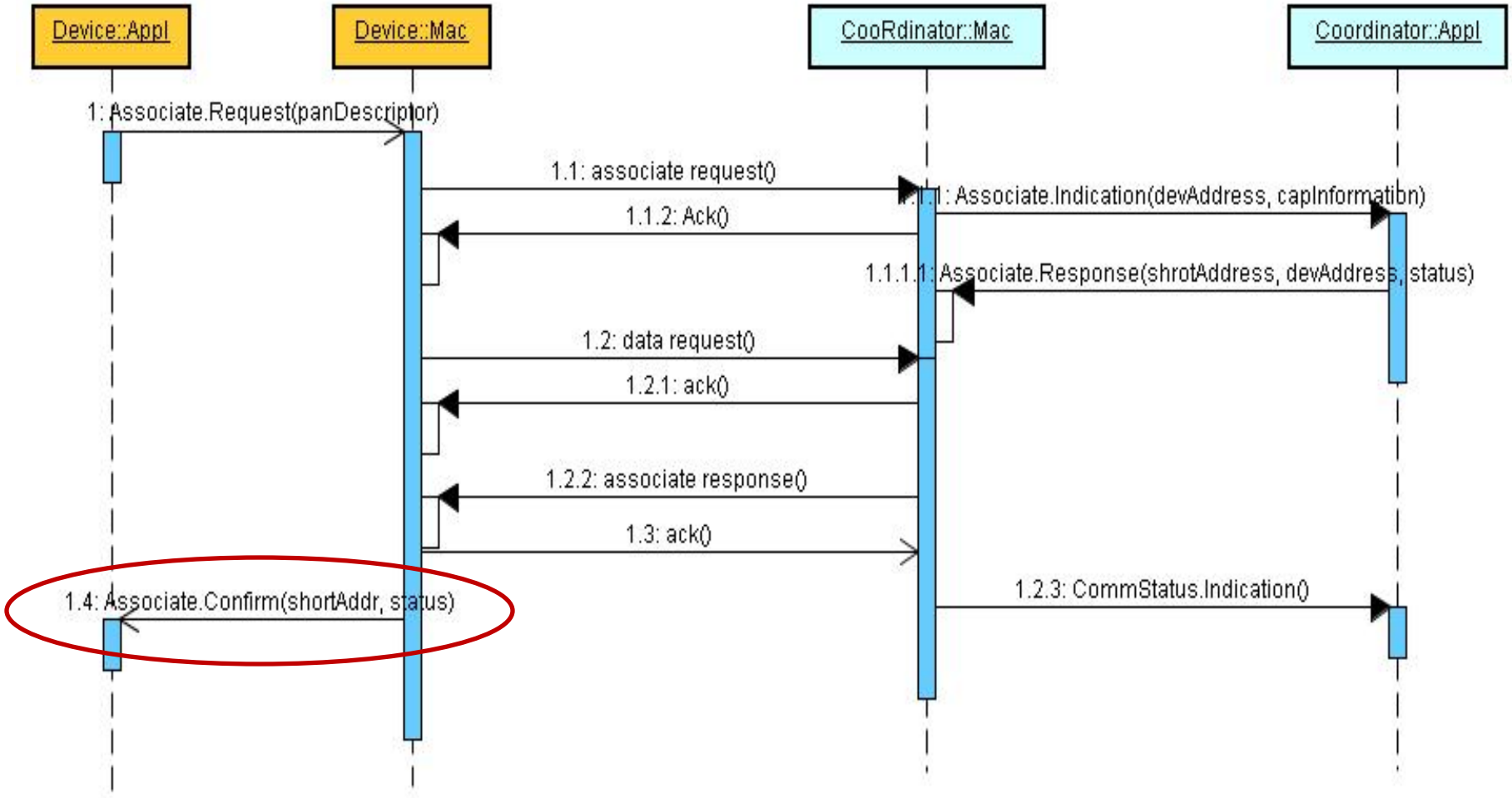

```
static void App_TransmitUartData (void)
{  static uint8_t keysBuffer[mMaxKeysToReceive_c];
    static uint8_t keysReceived = 0;
  /* get data from UART */
 if( keysReceived < mMaxKeysToReceive_c )
 { if(UartX_GetByteFromRxBuffer(&keysBuffer[keysReceived]))
  { keysReceived++;  }
 }
 if( (mcPendingPackets < mDefaultValueOfMaxPendingDataPackets_c) && (mpPacket == NULL) )
 { /* If the maximum number of pending data buffes is below maximum limit and we do not have a data
buffer already then allocate one. */
   mpPacket = MSG_AllocType(nwkToMcpsMessage_t);
 }
 if(mpPacket != NULL)
 { /* get data from UART */
    mpPacket->msgData.dataReq.pMsdu = &keysBuffer[0];
    /* Data was available in the UART receive buffer. Now create an MCPS-Data Request message
containing the UART data. */
    mpPacket->msgType = gMcpsDataReq_c;
    /* Create the header using coordinator information gained during the scan procedure. Also use the
short address we were assigned by the coordinator during association. */
    FLib_MemCpy(mpPacket->msgData.dataReq.dstAddr, mCoordInfo.coordAddress, 8);
    FLib_MemCpy(mpPacket->msgData.dataReq.srcAddr, maMyAddress, 8);
    FLib_MemCpy(mpPacket->msgData.dataReq.dstPanId, mCoordInfo.coordPanId, 2);
    FLib_MemCpy(mpPacket->msgData.dataReq.srcPanId, mCoordInfo.coordPanId, 2);
    mpPacket->msgData.dataReq.dstAddrMode = mCoordInfo.coordAddrMode;
    mpPacket->msgData.dataReq.srcAddrMode = mAddrMode;
    mpPacket->msgData.dataReq.msduLength = keysReceived;
    /* Request MAC level acknowledgement of the data packet */
    mpPacket->msgData.dataReq.txOptions = gTxOptsAck_c;
    /* Give the data packet a handle. The handle is returned in the MCPS-Data Confirm message. */
    mpPacket->msgData.dataReq.msduHandle = mMsduHandle++;
#ifdef gMAC2006_d    /* Don't use security */
   mpPacket->msgData.dataReq.securityLevel = 0;
#endif
    /* Send the Data Request to the MCPS */
    (void) MSG_Send(NWK_MCPS, mpPacket);
    /* Prepare for another data buffer */
    mpPacket = NULL;
    mcPendingPackets++;
    /* Receive another pressed keys */
    keysReceived = 0;
 }
 /*If the keysBuffer[] wasn't send over the air because of too many pending packets, try to send it later  */
  if (keysReceived)  { TS_SendEvent(gAppTaskID_c, gAppEvtRxFromUart_c);}
}
```

End Device Finite State Machine



To generate a dummy event call:
TS_SendEvent(gAppTaskID_c, gAppEvtDummyEvent_c);

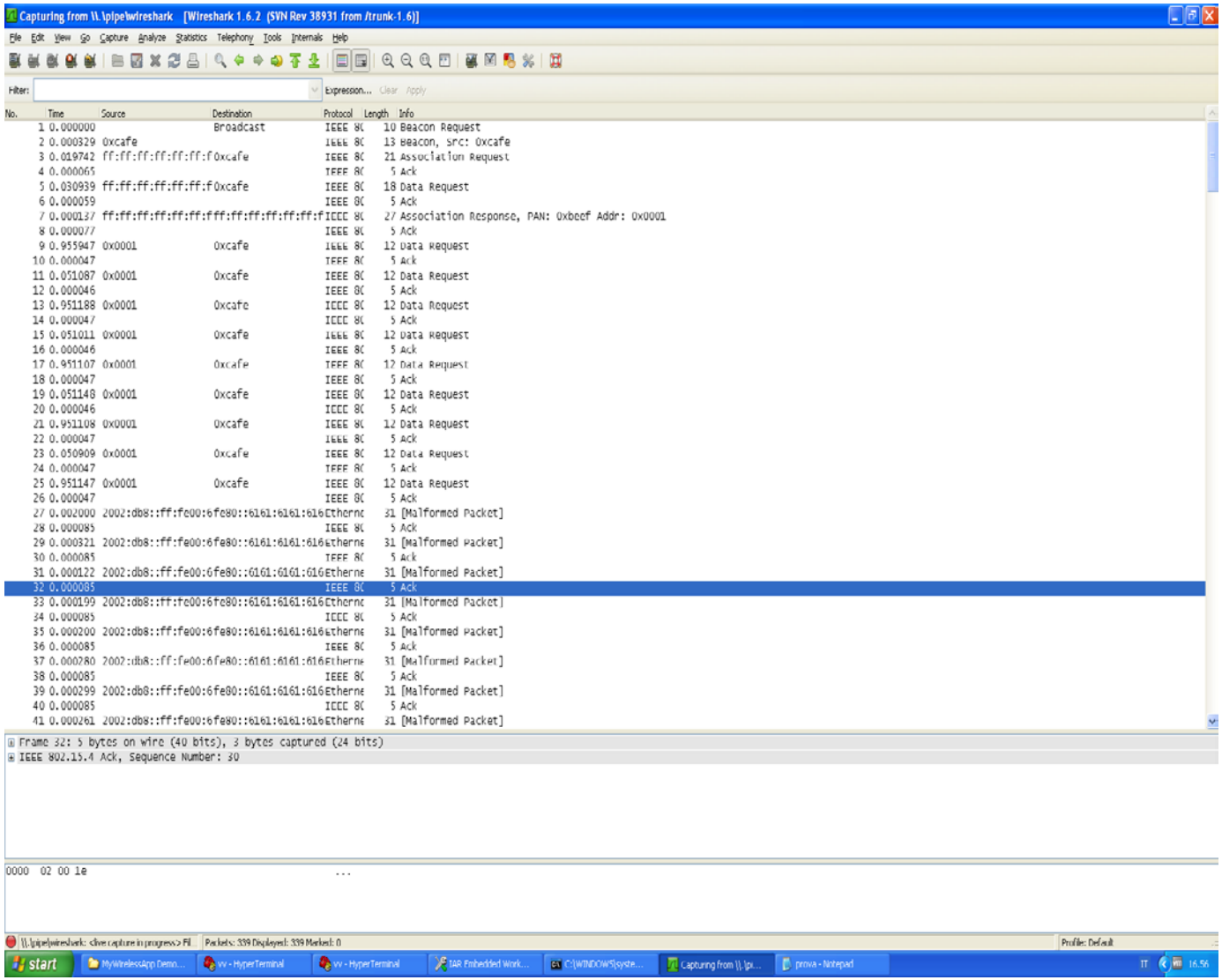
Generation after the Association



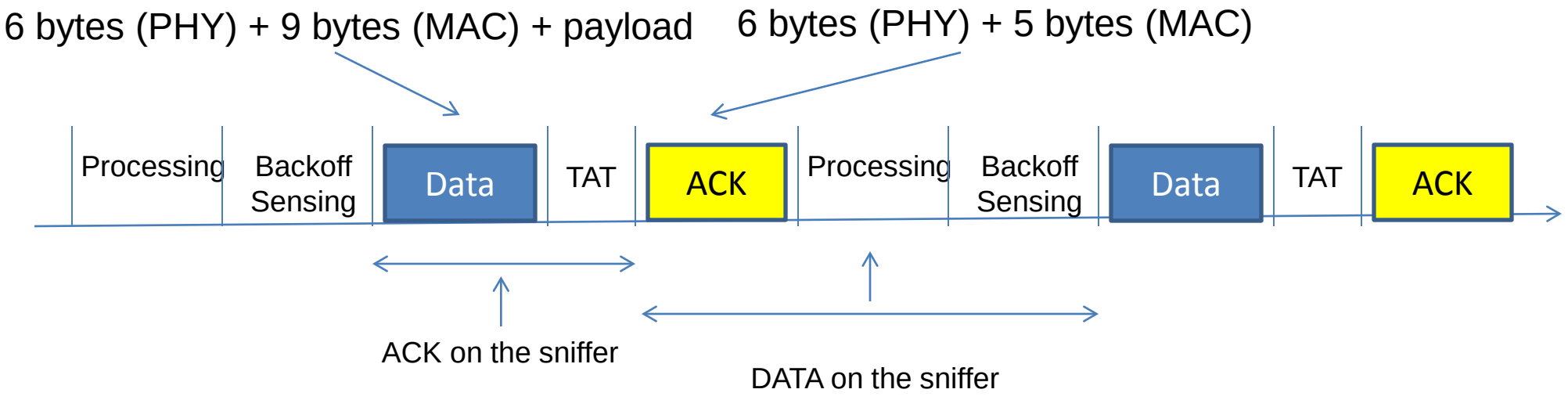
Generation after an event: Handle Keys

```
static void App_HandleKeys
(  key_event_t events  /*IN: Events from keyboard module */)
{  switch ( events )
   {  case gKBD_EventSW1_c:
      case gKBD_EventSW2_c:
      case gKBD_EventSW3_c:
      case gKBD_EventSW4_c:
      case gKBD_EventLongSW1_c:
      case gKBD_EventLongSW2_c:
      case gKBD_EventLongSW3_c:
      case gKBD_EventLongSW4_c:
      if(gState == stateInit)
        {
        StopLed1Flashing();
        StopLed2Flashing();
        StopLed3Flashing();
        StopLed4Flashing();
        LCD_ClearDisplay();
        LCD_WriteString(1,"Application");
        LCD_WriteString(2,"    started");
        TS_SendEvent(gAppTaskID_c, gAppEvtDummyEvent_c);
        }
   }
}
```

Sniffer

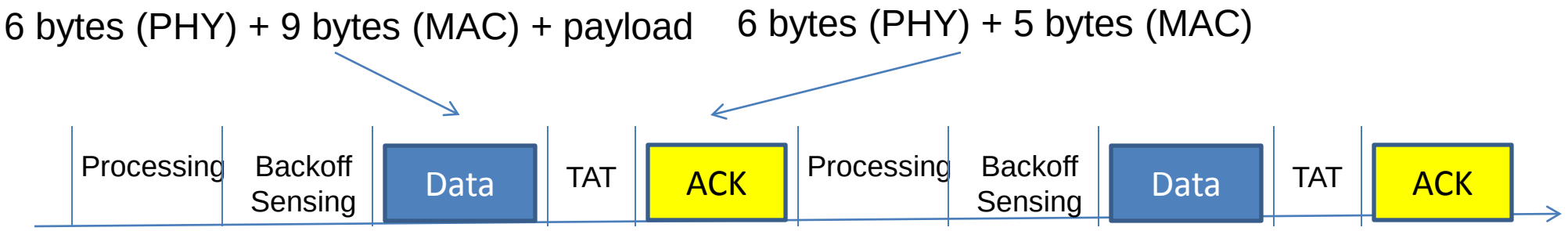

Capturing from \\.\pipe\wireshark [Wireshark 1.6.2 (SVN Rev 38931 from /trunk-1.6)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Length Info
1 0.000000 Broadcast IEEE 80 10 Beacon Request
2 0.000329 0xcafe IEEE 80 13 Beacon, src: 0xcafe
3 0.019742 ff:ff:ff:ff:ff:ff:f 0xcafe IEEE 80 21 Association Request
4 0.000065 IEEE 80 5 Ack
5 0.030939 ff:ff:ff:ff:ff:ff:f 0xcafe IEEE 80 18 Data Request
6 0.000059 IEEE 80 5 Ack
7 0.000137 ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:f IEEE 80 27 Association Response, PAN: 0xbeef Addr: 0x0001
8 0.000077 IEEE 80 5 Ack
9 0.955947 0x0001 0xcafe IEEE 80 12 Data Request
10 0.000047 IEEE 80 5 Ack
11 0.051087 0x0001 0xcafe IEEE 80 12 Data Request
12 0.000046 IEEE 80 5 Ack
13 0.951188 0x0001 0xcafe IEEE 80 12 Data Request
14 0.000047 IEEE 80 5 Ack
15 0.051011 0x0001 0xcafe IEEE 80 12 Data Request
16 0.000046 IEEE 80 5 Ack
17 0.951107 0x0001 0xcafe IEEE 80 12 Data Request
18 0.000047 IEEE 80 5 Ack
19 0.051148 0x0001 0xcafe IEEE 80 12 Data Request
20 0.000046 IEEE 80 5 Ack
21 0.951108 0x0001 0xcafe IEEE 80 12 Data Request
22 0.000047 IEEE 80 5 Ack
23 0.050909 0x0001 0xcafe IEEE 80 12 Data Request
24 0.000047 IEEE 80 5 Ack
25 0.951147 0x0001 0xcafe IEEE 80 12 Data Request
26 0.000047 IEEE 80 5 Ack
27 0.002000 2002:db8::ff:fe00:6fe80::6161:6161:616 Etherne 31 [Malformed Packet]
28 0.000085 IEEE 80 5 Ack
29 0.000321 2002:db8::ff:fe00:6fe80::6161:6161:616 Etherne 31 [Malformed Packet]
30 0.000085 IEEE 80 5 Ack
31 0.000122 2002:db8::ff:fe00:6fe80::6161:6161:616 Etherne 31 [Malformed Packet]
32 0.000085 IEEE 80 5 Ack
33 0.000199 2002:db8::ff:fe00:6fe80::6161:6161:616 Etherne 31 [Malformed Packet]
34 0.000085 IEEE 80 5 Ack
35 0.000200 2002:db8::ff:fe00:6fe80::6161:6161:616 Etherne 31 [Malformed Packet]
36 0.000085 IEEE 80 5 Ack
37 0.000780 2002:db8::ff:fe00:6fe80::6161:6161:616 Etherne 31 [Malformed Packet]
38 0.000085 IEEE 80 5 Ack
39 0.000299 2002:db8::ff:fe00:6fe80::6161:6161:616 Etherne 31 [Malformed Packet]
40 0.000085 IEEE 80 5 Ack
41 0.000261 2002:db8::ff:fe00:6fe80::6161:6161:616 Etherne 31 [Malformed Packet]
Frame 32: 5 bytes on wire (40 bits), 3 bytes captured (24 bits)
IEEE 802.15.4 Ack, Sequence Number: 30
0000 02 00 1e ...
start

Sniffer



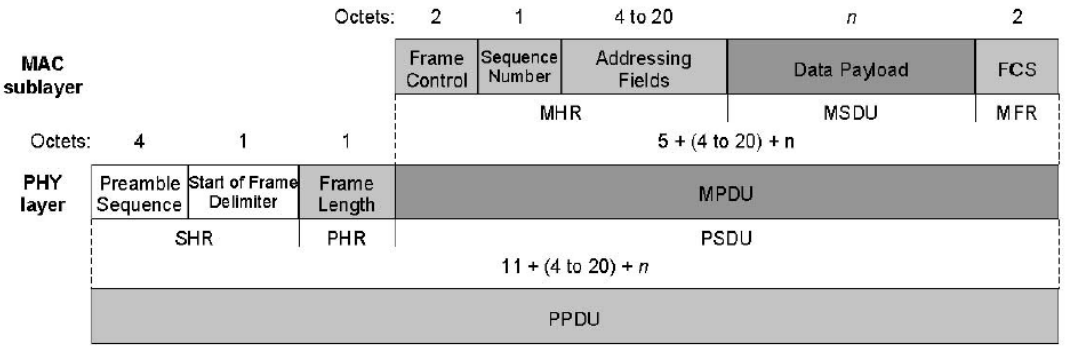
Time between the reception of the first bit of the previous packet and the first bit of the current packet

Sniffer



TAT = 12 T_s = 192 μs
Packet Size =
11 + addressing fields + payload

Addressing fields = 6 Bytes



Wireless Sensor Networks

Chiara Buratti

www.chiaraburatti.org

c.buratti@unibo.it