```c
/*
    libxbee - a C library to aid the use of Digi's Series 1 XBee modules
              running in API mode (AP=2).

    Copyright (C) 2009  Attie Grande (attie@attie.co.uk)

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "globals.h"
#include "api.h"

/* ready flag.
   needs to be set to -1 so that the listen thread can begin.
   then 1 so that functions can be used (after setup of course...) */
volatile int xbee_ready = 0;

/* ################################################################# */
/* ### Memory Handling ############################################# */
/* ################################################################# */

/* malloc wrapper function */
static void *Xmalloc(size_t size) {
  void *t;
  t = malloc(size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:malloc()");
    exit(1);
  }
  return t;
}

/* calloc wrapper function */
static void *Xcalloc(size_t size) {
  void *t;
  t = calloc(1, size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:calloc()");
    exit(1);
  }
  return t;
}

/* realloc wrapper function */
static void *Xrealloc(void *ptr, size_t size) {
  void *t;
  t = realloc(ptr,size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:realloc()");
    exit(1);
  }
  return t;
}

/* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
static void Xfree2(void **ptr) {
  free(*ptr);
  *ptr = NULL;
}

/* ################################################################# */
/* ### Helper Functions ############################################ */
/* ################################################################# */

/* #################################################################
   returns 1 if the packet has data for the digital input else 0 */
int xbee_hasdigital(xbee_pkt *pkt, int input) {
  int mask = 0x0001;
  if (input < 0 || input > 7) return 0;

  mask <<= input;
  return !!(pkt->IOmask & mask);
}

/* ##################################################################
   returns 1 if the digital input is high else 0 (or 0 if no digital data present) */
int xbee_getdigital(xbee_pkt *pkt, int input) {
  int mask = 0x0001;
  if (input < 0 || input > 7) return 0;

  if (!xbee_hasdigital(pkt,input)) return 0;

  mask <<= input;
  return !!(pkt->IOdata & mask);
}

/* ##################################################################
   returns 1 if the packet has data for the analog input else 0 */
int xbee_hasanalog(xbee_pkt *pkt, int input) {
  int mask = 0x0200;
  if (input < 0 || input > 5) return 0;

  mask <<= input;
  return !!(pkt->IOmask & mask);
}

/* ##################################################################
   returns analog input as a voltage if vRef is non-zero, else raw value (or 0 if no analog data present) */
double xbee_getanalog(xbee_pkt *pkt, int input, double Vref) {
  if (input < 0 || input > 5) return 0;
  if (!xbee_hasanalog(pkt,input)) return 0;

  if (Vref) return (Vref / 1023) * pkt->IOanalog[input];
  return pkt->IOanalog[input];
}

/* ################################################################## */
/* ### XBee Functions ############################################### */
/* ################################################################## */

/* ##################################################################
   xbee_setup
   opens xbee serial port & creates xbee listen thread
   the xbee must be configured for API mode 2
   THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
int xbee_setup(char *path, int baudrate) {
  return xbee_setuplog(path,baudrate,0);
}
int xbee_setuplog(char *path, int baudrate, int logfd) {
  t_info info;
  struct flock fl;
  struct termios tc;
  speed_t chosenbaud;

#ifdef DEBUG
  xbee.logfd = ((logfd)?logfd:stdout);
#else
  xbee.logfd = logfd;
#endif
  if (xbee.logfd) {
    xbee.log = fdopen(xbee.logfd,"w");
    if (!xbee.log) {
      /* errno == 9 is bad file descriptor (probrably not provided) */
      if (errno != 9) perror("Failed opening logfile");
      xbee.logfd = 0;
    }
  }

  /* select the baud rate */
  switch (baudrate) {
    case 1200:  chosenbaud = B1200;   break;
    case 2400:  chosenbaud = B2400;   break;
    case 4800:  chosenbaud = B4800;   break;
    case 9600:  chosenbaud = B9600;   break;
    case 19200: chosenbaud = B19200;  break;
    case 38400: chosenbaud = B38400;  break;
    case 57600: chosenbaud = B57600;  break;
    case 115200:chosenbaud = B115200; break;
    default:
      fprintf(stderr,"XBee: Unknown or incompatiable baud rate specified... (%d)\n",baudrate);
      return -1;
  };

  /* setup the connection mutex */
  xbee.conlist = NULL;
  if (pthread_mutex_init(&xbee.conmutex,NULL)) {
    perror("xbee_setup():pthread_mutex_init(conmutex)");
    return -1;
  }

  /* setup the packet mutex */
  xbee.pktlist = NULL;
  if (pthread_mutex_init(&xbee.pktmutex,NULL)) {
    perror("xbee_setup():pthread_mutex_init(pktmutex)");
    return -1;
  }

  /* setup the send mutex */
  if (pthread_mutex_init(&xbee.sendmutex,NULL)) {
    perror("xbee_setup():pthread_mutex_init(sendmutex)");
    return -1;
  }

  /* take a copy of the XBee device path */
  if ((xbee.path = Xmalloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
    perror("xbee_setup():Xmalloc(path)");
    return -1;
  }
  strcpy(xbee.path,path);

  /* open the serial port as a file descriptor */
  if ((xbee.ttyfd = open(path,O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
    perror("xbee_setup():open()");
    Xfree(xbee.path);
    xbee.ttyfd = -1;
    xbee.tty = NULL;
    return -1;
  }

  /* lock the file */
  fl.l_type = F_WRLCK | F_RDLCK;
  fl.l_whence = SEEK_SET;
  fl.l_start = 0;
  fl.l_len = 0;
  fl.l_pid = getpid();
  if (fcntl(xbee.ttyfd, F_SETLK, &fl) == -1) {
    perror("xbee_setup():fcntl()");
    Xfree(xbee.path);
    close(xbee.ttyfd);
    xbee.ttyfd = -1;
    xbee.tty = NULL;
    return -1;
  }


  /* open the serial port as a FILE* */
  if ((xbee.tty = fdopen(xbee.ttyfd,"r+")) == NULL) {
    perror("xbee_setup():fdopen()");
    Xfree(xbee.path);
    close(xbee.ttyfd);
    xbee.ttyfd = -1;
    xbee.tty = NULL;
    return -1;
  }

  /* flush the serial port */
  fflush(xbee.tty);

  /* setup the baud rate and other io attributes */
  tcgetattr(xbee.ttyfd, &tc);
  /* input flags */
  tc.c_iflag &= ~IGNBRK;           /* enable ignoring break */
  tc.c_iflag &= ~(IGNPAR | PARMRK);/* disable parity checks */
  tc.c_iflag &= ~INPCK;            /* disable parity checking */
  tc.c_iflag &= ~ISTRIP;           /* disable stripping 8th bit */
  tc.c_iflag &= ~(INLCR | ICRNL);  /* disable translating NL <-> CR */
  tc.c_iflag &= ~IGNCR;            /* disable ignoring CR */
  tc.c_iflag &= ~(IXON | IXOFF);   /* disable XON/XOFF flow control */
  /* output flags */
  tc.c_oflag &= ~OPOST;            /* disable output processing */
  tc.c_oflag &= ~(ONLCR | OCRNL);  /* disable translating NL <-> CR */
  tc.c_oflag &= ~OFILL;            /* disable fill characters */
  /* control flags */
  tc.c_cflag |= CREAD;             /* enable reciever */
  tc.c_cflag &= ~PARENB;           /* disable parity */
  tc.c_cflag &= ~CSTOPB;           /* disable 2 stop bits */
  tc.c_cflag &= ~CSIZE;            /* remove size flag... */
  tc.c_cflag |= CS8;               /* ...enable 8 bit characters */
  tc.c_cflag |= HUPCL;             /* enable lower control lines on close - hang up */
  /* local flags */
  tc.c_lflag &= ~ISIG;             /* disable generating signals */
  tc.c_lflag &= ~ICANON;          /* disable canonical mode - line by line */
  tc.c_lflag &= ~ECHO;            /* disable echoing characters */
  tc.c_lflag &= ~ECHONL;          /* ??? */
  tc.c_lflag &= ~NOFLSH;          /* disable flushing on SIGINT */
  tc.c_lflag &= ~IEXTEN;          /* disable input processing */
  /* control characters */
  memset(tc.c_cc,0,sizeof(tc.c_cc));
  /* i/o rates */
  cfsetspeed(&tc, chosenbaud);     /* set i/o baud rate */
  tcsetattr(xbee.ttyfd, TCSANOW, &tc);
  tcflow(xbee.ttyfd, TCOON|TCION); /* enable input & output transmission */

  /* allow the listen thread to start */
  xbee_ready = -1;

  /* can start xbee_listen thread now */
  if (pthread_create(&xbee.listent,NULL,(void *(*)(void *))xbee_listen_wrapper,(void *)&info) != 0) {
    perror("xbee_setup():pthread_create()");
    Xfree(xbee.path);
    fclose(xbee.tty);
    close(xbee.ttyfd);
    xbee.ttyfd = -1;
    xbee.tty = NULL;
    return -1;
  }

  usleep(100);
  while (xbee_ready != -2) {
    usleep(100);
    if (xbee.logfd) {
      fprintf(xbee.log,"XBee: Waiting for xbee_listen() to be ready...\n");
    }

  }

  /* allow other functions to be used! */
  xbee_ready = 1;

  return 0;
}

/* ################################################################
   xbee_con
   produces a connection to the specified device and frameID
   if a connection had already been made, then this connection will be returned */
xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
  xbee_con *con, *ocon;
  unsigned char tAddr[8];
  va_list ap;
  int t;
  int i;

  ISREADY;

  if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
  else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */

  va_start(ap,type);
  /* if: 64 bit address expected (2 ints) */
  if ((type == xbee_64bitRemoteAT) ||
      (type == xbee_64bitData) ||
      (type == xbee_64bitIO)) {
    t = va_arg(ap, int);
    tAddr[0] = (t >> 24) & 0xFF;
    tAddr[1] = (t >> 16) & 0xFF;
    tAddr[2] = (t >>  8) & 0xFF;
    tAddr[3] = (t      ) & 0xFF;
    t = va_arg(ap, int);
    tAddr[4] = (t >> 24) & 0xFF;
    tAddr[5] = (t >> 16) & 0xFF;
    tAddr[6] = (t >>  8) & 0xFF;
    tAddr[7] = (t      ) & 0xFF;

  /* if: 16 bit address expected (1 int) */
  } else if ((type == xbee_16bitRemoteAT) ||
             (type == xbee_16bitData) ||
             (type == xbee_16bitIO)) {
    t = va_arg(ap, int);
    tAddr[0] = (t >>  8) & 0xFF;
    tAddr[1] = (t      ) & 0xFF;
    tAddr[2] = 0;
    tAddr[3] = 0;
    tAddr[4] = 0;
    tAddr[5] = 0;
    tAddr[6] = 0;
```

```
      tAddr[7] = 0;

    /* otherwise clear the address */
    } else {
      memset(tAddr,0,8);
    }
    va_end(ap);

    /* lock the connection mutex */
    pthread_mutex_lock(&xbee.conmutex);

    /* are there any connections? */
    if (xbee.conlist) {
      con = xbee.conlist;
      while (con) {
        /* if: after a modemStatus, and the types match! */
        if ((type == xbee_modemStatus) &&
            (con->type == type)) {
          pthread_mutex_unlock(&xbee.conmutex);
          return con;

        /* if: after a txStatus and frameIDs match! */
        } else if ((type == xbee_txStatus) &&
                   (con->type == type) &&
                   (frameID == con->frameID)) {
          pthread_mutex_unlock(&xbee.conmutex);
          return con;

        /* if: after a localAT, and the frameIDs match! */
        } else if ((type == xbee_localAT) &&
                   (con->type == type) &&
                   (frameID == con->frameID)) {
          pthread_mutex_unlock(&xbee.conmutex);
          return con;

        /* if: connection types match, the frameIDs match, and the addresses match! */
        } else if ((type == con->type) &&
                   (frameID == con->frameID) &&
                   (!memcmp(tAddr,con->tAddr,8))) {
          pthread_mutex_unlock(&xbee.conmutex);
          return con;
        }

        /* if there are more, move along, dont want to loose that last item! */
        if (con->next == NULL) break;
        con = con->next;
      }

      /* keep hold of the last connection... we will need to link it up later */
      ocon = con;
    }

    /* create a new connection and set its attributes */
    con = Xcalloc(sizeof(xbee_con));
    con->type = type;
    /* is it a 64bit connection? */
    if ((type == xbee_64bitRemoteAT) ||
        (type == xbee_64bitData) ||
        (type == xbee_64bitIO)) {
      con->tAddr64 = TRUE;
    }
    con->atQueue = 0; /* queue AT commands? */
    con->txDisableACK = 0; /* disable ACKs? */
    con->txBroadcast = 0; /* broadcast? */
    con->frameID = frameID;
    memcpy(con->tAddr,tAddr,8); /* copy in the remote address */

    if (xbee.logfd) {
      switch(type) {
      case xbee_localAT:
        fprintf(xbee.log,"XBee: New local AT connection!\n");
        break;
      case xbee_16bitRemoteAT:
      case xbee_64bitRemoteAT:
        fprintf(xbee.log,"XBee: New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
        for (i=0;i<(con->tAddr64?8:2);i++) {
          fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
        }
        fprintf(xbee.log,")\n");
        break;
      case xbee_16bitData:
      case xbee_64bitData:
        fprintf(xbee.log,"XBee: New %d-bit data connection! (to: ",(con->tAddr64?64:16));
        for (i=0;i<(con->tAddr64?8:2);i++) {
          fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
```

```c
      }
      fprintf(xbee.log,")\n");
      break;
    case xbee_16bitIO:
    case xbee_64bitIO:
      fprintf(xbee.log,"XBee: New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
      for (i=0;i<(con->tAddr64?8:2);i++) {
        fprintf(xbee.log,(i?":%02X":"%02X"),tAddr[i]);
      }
      fprintf(xbee.log,")\n");
      break;
    case xbee_txStatus:
      fprintf(xbee.log,"XBee: New Tx status connection!\n");
      break;
    case xbee_modemStatus:
      fprintf(xbee.log,"XBee: New modem status connection!\n");
      break;
    case xbee_unknown:
    default:
      fprintf(xbee.log,"XBee: New unknown connection!\n");
    }
  }

  /* make it the last in the list */
  con->next = NULL;
  /* add it to the list */
  if (xbee.conlist) {
    ocon->next = con;
  } else {
    xbee.conlist = con;
  }

  /* unlock the mutex */
  pthread_mutex_unlock(&xbee.conmutex);
  return con;
}

/* ################################################################
   xbee_conflush
   removes any packets that have been collected for the specified
   connection */
void xbee_flushcon(xbee_con *con) {
  xbee_pkt *p;
  while ((p = xbee_getpacket(con)) != NULL) {
    free(p);
  }
}

/* ################################################################
   xbee_endcon
   close the unwanted connection */
void xbee_endcon2(xbee_con **con) {
  xbee_con *t, *u;
  xbee_pkt *r, *p;

  /* lock the connection mutex */
  pthread_mutex_lock(&xbee.conmutex);

  u = t = xbee.conlist;
  while (t && t != *con) {
    u = t;
    t = t->next;
  }
  if (!u) {
    /* invalid connection given... */
    if (xbee.logfd) {
      fprintf(xbee.log,"XBee: Attempted to close invalid connection...\n");
    }
    /* unlock the connection mutex */
    pthread_mutex_unlock(&xbee.conmutex);
    return;
  }
  /* extract this connection from the list */
  u->next = u->next->next;

  /* unlock the connection mutex */
  pthread_mutex_unlock(&xbee.conmutex);

  /* lock the packet mutex */
  pthread_mutex_lock(&xbee.pktmutex);

  /* if: there are packets */
  if ((p = xbee.pktlist) != NULL) {
    r = NULL;
    /* get all packets for this connection */
    do {
      /* does the packet match the connection? */
      if (xbee_matchpktcon(p,*con)) {
        /* if it was the first packet */
        if (!r) {
          /* move the chain along */
          xbee.pktlist = p->next;
        } else {
          /* otherwise relink the list */
          r->next = p->next;
        }

        /* free this packet! */
        Xfree(p);
      }
      /* move on */
      r = p;
      p = p->next;
    } while (p);
  }

  /* unlock the packet mutex */
  pthread_mutex_unlock(&xbee.pktmutex);


  Xfree(*con);
}

/* ##################################################################
   xbee_senddata
   send the specified data to the provided connection */
int xbee_senddata(xbee_con *con, char *format, ...) {
  int ret;
  va_list ap;

  ISREADY;

  /* xbee_vsenddata() wants a va_list... */
  va_start(ap, format);
  /* hand it over :) */
  ret = xbee_vsenddata(con,format,ap);
  va_end(ap);
  return ret;
}

int xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
  unsigned char data[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
  int length;

  ISREADY;

  /* make up the data and keep the length, its possible there are nulls in there */
  length = vsnprintf((char *)data,128,format,ap);

  /* hand it over :) */
  return xbee_nsenddata(con,(char *)data,length);
}

int xbee_nsenddata(xbee_con *con, char *data, int length) {
  t_data *pkt;
  int i;
  unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */

  ISREADY;

  if (!con) return -1;
  if (con->type == xbee_unknown) return -1;
  if (length > 127) return -1;

  if (xbee.logfd) {
    fprintf(xbee.log,"XBee: --== TX Packet ============--\n");
    fprintf(xbee.log,"XBee: Length: %d\n",length);
    for (i=0;i<length;i++) {
      fprintf(xbee.log,"XBee: %3d | 0x%02X ",i,data[i]);
      if ((data[i] > 32) && (data[i] < 127)) {
        fprintf(xbee.log,"'%c'\n",data[i]);
      } else{
        fprintf(xbee.log," _\n");
      }
    }
  }

  /* ########################################## */
  /* if: local AT */
  if (con->type == xbee_localAT) {
    /* AT commands are 2 chars long (plus optional parameter) */
    if (length < 2) return -1;

    /* use the command? */
    buf[0] = ((!con->atQueue)?0x08:0x09);
    buf[1] = con->frameID;

    /* copy in the data */
    for (i=0;i<length;i++) {
      buf[i+2] = data[i];
    }

    /* setup the packet */
    pkt = xbee_make_pkt(buf,i+2);
    /* send it on */
    xbee_send_pkt(pkt);

    return 0;

  /* ########################################### */
  /* if: remote AT */
  } else if ((con->type == xbee_16bitRemoteAT) ||
             (con->type == xbee_64bitRemoteAT)) {
    if (length < 2) return -1; /* at commands are 2 chars long (plus optional parameter) */
    buf[0] = 0x17;
    buf[1] = con->frameID;

    /* copy in the relevant address */
    if (con->tAddr64) {
      memcpy(&buf[2],con->tAddr,8);
      buf[10] = 0xFF;
      buf[11] = 0xFE;
    } else {
      memset(&buf[2],0,8);
      memcpy(&buf[10],con->tAddr,2);
    }
    /* queue the command? */
    buf[12] = ((!con->atQueue)?0x02:0x00);

    /* copy in the data */
    for (i=0;i<length;i++) {
      buf[i+13] = data[i];
    }

    /* setup the packet */
    pkt = xbee_make_pkt(buf,i+13);
    /* send it on */
    xbee_send_pkt(pkt);

    return 0;

  /* ########################################### */
  /* if: 16 or 64bit Data */
  } else if ((con->type == xbee_16bitData) ||
             (con->type == xbee_64bitData)) {
    int offset;

    /* if: 16bit Data */
    if (con->type == xbee_16bitData) {
      buf[0] = 0x01;
      offset = 5;
      /* copy in the address */
      memcpy(&buf[2],con->tAddr,2);

    /* if: 64bit Data */
    } else { /* 64bit Data */
      buf[0] = 0x00;
      offset = 11;
      /* copy in the address */
      memcpy(&buf[2],con->tAddr,8);
    }

    /* copy frameID */
    buf[1] = con->frameID;

    /* disable ack? broadcast? */
    buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);

    /* copy in the data */
    for (i=0;i<length;i++) {
      buf[i+offset] = data[i];
    }

    /* setup the packet */
    pkt = xbee_make_pkt(buf,i+offset);
    /* send it on */
    xbee_send_pkt(pkt);

    return 0;

  /* ########################################### */
  /* if: I/O */
  } else if ((con->type == xbee_64bitIO) ||
             (con->type == xbee_16bitIO)) {
    /* not currently implemented... is it even allowed? */
    if (xbee.logfd) {
      fprintf(xbee.log,"******* TODO ********\n");
    }
  }

  return -2;
}

/* ##################################################################
   xbee_getpacket
   retrieves the next packet destined for the given connection
   once the packet has been retrieved, it is removed for the list! */
xbee_pkt *xbee_getpacketwait(xbee_con *con) {
  xbee_pkt *p;
  int i;

  /* 50ms * 20 = 1 second */
  for (i = 0; i < 20; i++) {
    p = xbee_getpacket(con);
    if (p) break;
    usleep(50000); /* 50ms */
  }

  return p;
}
xbee_pkt *xbee_getpacket(xbee_con *con) {
  xbee_pkt *l, *p, *q;
  int c;
  if (xbee.logfd) {
    fprintf(xbee.log,"XBee: --== Get Packet ==========--\n");
  }

  /* lock the packet mutex */
  pthread_mutex_lock(&xbee.pktmutex);

  /* if: there are no packets */
  if ((p = xbee.pktlist) == NULL) {
    pthread_mutex_unlock(&xbee.pktmutex);
    if (xbee.logfd) {
      fprintf(xbee.log,"XBee: No packets avaliable...\n");
    }
    return NULL;
  }

  l = NULL;
  q = NULL;
  /* get the first avaliable packet for this connection */
  do {
    /* does the packet match the connection? */
    if (xbee_matchpktcon(p,con)) {
      q = p;
      break;
    }
    /* move on */
    l = p;
    p = p->next;
  } while (p);

  /* if: no packet was found */
  if (!q) {
    pthread_mutex_unlock(&xbee.pktmutex);
    if (xbee.logfd) {
      fprintf(xbee.log,"XBee: No packets avaliable (for connection)...\n");
    }
    return NULL;
  }

  /* if it was not the first packet */
  if (l) {
    /* otherwise relink the list */
    l->next = p->next;
  } else {
    /* move the chain along */
    xbee.pktlist = p->next;
  }
```

```c

  /* unlink this packet from the chain! */
  q->next = NULL;

  if (xbee.logfd) {
    fprintf(xbee.log,"XBee: Got a packet\n");
    for (p = xbee.pktlist,c = 0;p;c++,p = p->next);
    fprintf(xbee.log,"XBee: Packets left: %d\n",c);
  }

  /* unlock the packet mutex */
  pthread_mutex_unlock(&xbee.pktmutex);

  /* and return the packet (must be freed by caller!) */
  return q;
}

/* #################################################################
   xbee_matchpktcon - INTERNAL
   checks if the packet matches the connection */
static int xbee_matchpktcon(xbee_pkt *pkt, xbee_con *con) {
  /* if: the connection type matches the packet type OR
     the connection is 16/64bit remote AT, and the packet is a remote AT response */
  if ((pkt->type == con->type) || /* -- */
      ((pkt->type == xbee_remoteAT) && /* -- */
       ((con->type == xbee_16bitRemoteAT) ||
        (con->type == xbee_64bitRemoteAT)))) {

    /* if: the packet is modem status OR
       the packet is tx status or AT data and the frame IDs match OR
       the addresses match */
    if (pkt->type == xbee_modemStatus) return 1;

    if ((pkt->type == xbee_txStatus) ||
        (pkt->type == xbee_localAT) ||
        (pkt->type == xbee_remoteAT)) {
      if (pkt->frameID == con->frameID) {
        return 1;
      }
    } else if (pkt->sAddr64 && !memcmp(pkt->Addr64,con->tAddr,8)) {
      return 1;
    } else if (!pkt->sAddr64 && !memcmp(pkt->Addr16,con->tAddr,2)) {
      return 1;
    }
  }
  return 0;
}

/* #################################################################
   xbee_parse_io - INTERNAL
   parses the data given into the packet io information */
static int xbee_parse_io(xbee_pkt *p, unsigned char *d, int offset, int i) {
  /* copy in the I/O data mask */
  p->IOmask = (((d[offset]<<8) | d[offset + 1]) & 0x7FFF);

  /* copy in the digital I/O data */
  p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);

  /* advance over the digital data, if its there */
  i += ((p->IOmask & 0x01FF)?2:0);

  /* copy in the analog I/O data */
  if (p->IOmask & 0x0200) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
  if (p->IOmask & 0x0400) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
  if (p->IOmask & 0x0800) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
  if (p->IOmask & 0x1000) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
  if (p->IOmask & 0x2000) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
  if (p->IOmask & 0x4000) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
  if (xbee.logfd) {
    if (p->IOmask & 0x0001)
      fprintf(xbee.log,"XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
    if (p->IOmask & 0x0002)
      fprintf(xbee.log,"XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
    if (p->IOmask & 0x0004)
      fprintf(xbee.log,"XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
    if (p->IOmask & 0x0008)
      fprintf(xbee.log,"XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
    if (p->IOmask & 0x0010)
      fprintf(xbee.log,"XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
    if (p->IOmask & 0x0020)
      fprintf(xbee.log,"XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
    if (p->IOmask & 0x0040)
      fprintf(xbee.log,"XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
    if (p->IOmask & 0x0080)
      fprintf(xbee.log,"XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
    if (p->IOmask & 0x0100)
      fprintf(xbee.log,"XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
    if (p->IOmask & 0x0200)
      fprintf(xbee.log,"XBee: Analog  0: %d (~%.2fv)\n",p->IOanalog[0],(3.3/1023)*p->IOanalog[0]);
    if (p->IOmask & 0x0400)
      fprintf(xbee.log,"XBee: Analog  1: %d (~%.2fv)\n",p->IOanalog[1],(3.3/1023)*p->IOanalog[1]);
    if (p->IOmask & 0x0800)
      fprintf(xbee.log,"XBee: Analog  2: %d (~%.2fv)\n",p->IOanalog[2],(3.3/1023)*p->IOanalog[2]);
    if (p->IOmask & 0x1000)
      fprintf(xbee.log,"XBee: Analog  3: %d (~%.2fv)\n",p->IOanalog[3],(3.3/1023)*p->IOanalog[3]);
    if (p->IOmask & 0x2000)
      fprintf(xbee.log,"XBee: Analog  4: %d (~%.2fv)\n",p->IOanalog[4],(3.3/1023)*p->IOanalog[4]);
    if (p->IOmask & 0x4000)
      fprintf(xbee.log,"XBee: Analog  5: %d (~%.2fv)\n",p->IOanalog[5],(3.3/1023)*p->IOanalog[5]);
  }

  return i;
}

/* ################################################################
   xbee_listen_wrapper - INTERNAL
   the xbee_listen wrapper. Prints an error when xbee_listen ends */
static void xbee_listen_wrapper(t_info *info) {
  int ret;

  /* just falls out if the proper 'go-ahead' isn't given */
  if (xbee_ready != -1) return;
  /* now allow the parent to continue */
  xbee_ready = -2;

  info->i = -1;
  for (;;) {
    ret = xbee_listen(info);
    if (xbee.logfd) {
      fprintf(xbee.log,"XBee: xbee_listen() returned [%d]... Restarting in 250ms!\n",ret);
    }
    usleep(25000);
  }
}

/* xbee_listen - INTERNAL
   the xbee xbee_listen thread
   reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
static int xbee_listen(t_info *info) {
  unsigned char c, t, d[128];
  unsigned int l, i, chksum, o;
  int j;
  xbee_pkt *p, *q, *po;
  xbee_con *con;
  int hasCon;

  /* just falls out if the proper 'go-ahead' isn't given */
  if (info->i != -1) return -1;

  /* do this forever :) */
  while(1) {
    /* wait for a valid start byte */
    if (xbee_getRawByte() != 0x7E) continue;

    if (xbee.logfd) {
      fprintf(xbee.log,"XBee: --== RX Packet ===========--\nXBee: Got a packet!...\n");
    }

    /* get the length */
    l = xbee_getByte() << 8;
    l += xbee_getByte();

    /* check it is a valid length... */
    if (!l) {
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Recived zero length packet!\n");
      }
      continue;
    }
    if (l > 100) {
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Recived oversized packet! Length: %d\n",l - 1);
      }
      continue;
    }

    if (xbee.logfd) {
      fprintf(xbee.log,"XBee: Length: %d\n",l - 1);
    }

    /* get the packet type */
    t = xbee_getByte();

    /* start the checksum */
    chksum = t;

    /* suck in all the data */
    for (i = 0; l > 1 && i < 128; l--, i++) {
      /* get an unescaped byte */
      c = xbee_getByte();
      d[i] = c;
      chksum += c;
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: %3d | 0x%02X | ",i,c);
        if ((c > 32) && (c < 127)) fprintf(xbee.log,"'%c'\n",c); else fprintf(xbee.log," _\n");
      }
    }
    i--; /* it went up too many times!... */

    /* add the checksum */
    chksum += xbee_getByte();

    /* check if the whole packet was recieved, or something else occured... unlikely... */
    if (l>1) {
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Didn't get whole packet... :(\n");
      }
      continue;
    }

    /* check the checksum */
    if ((chksum & 0xFF) != 0xFF) {
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Invalid Checksum: 0x%02X\n",chksum);
      }
      continue;
    }

    /* make a new packet */
    po = p = Xcalloc(sizeof(xbee_pkt));
    q = NULL;
    p->datalen = 0;

    /* ########################################### */
    /* if: modem status */
    if (t == 0x8A) {
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Packet type: Modem Status (0x8A)\n");
        fprintf(xbee.log,"XBee: ");
        switch (d[0]) {
        case 0x00: fprintf(xbee.log,"Hardware reset"); break;
        case 0x01: fprintf(xbee.log,"Watchdog timer reset"); break;
        case 0x02: fprintf(xbee.log,"Associated"); break;
        case 0x03: fprintf(xbee.log,"Disassociated"); break;
        case 0x04: fprintf(xbee.log,"Synchronization lost"); break;
        case 0x05: fprintf(xbee.log,"Coordinator realignment"); break;
        case 0x06: fprintf(xbee.log,"Coordinator started"); break;
        }
        fprintf(xbee.log,"...\n");
      }
      p->type = xbee_modemStatus;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = TRUE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      /* modem status can only ever give 1 'data' byte */
      p->datalen = 1;
      p->data[0] = d[0];

    /* ########################################### */
    /* if: local AT response */
    } else if (t == 0x88) {
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Packet type: Local AT Response (0x88)\n");
        fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
        fprintf(xbee.log,"XBee: AT Command: %c%c\n",d[1],d[2]);
        if (d[3] == 0) fprintf(xbee.log,"XBee: Status: OK\n");
        else if (d[3] == 1) fprintf(xbee.log,"XBee: Status: Error\n");
        else if (d[3] == 2) fprintf(xbee.log,"XBee: Status: Invalid Command\n");
        else if (d[3] == 3) fprintf(xbee.log,"XBee: Status: Invalid Parameter\n");
      }
      p->type = xbee_localAT;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      p->frameID = d[0];
      p->atCmd[0] = d[1];
      p->atCmd[1] = d[2];

      p->status = d[3];

      /* copy in the data */
      p->datalen = i-3;
      for (;i>3;i--) p->data[i-4] = d[i];

    /* ########################################### */
    /* if: remote AT response */
    } else if (t == 0x97) {
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Packet type: Remote AT Response (0x97)\n");
        fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
        fprintf(xbee.log,"XBee: 64-bit Address: ");
        for (j=0;j<8;j++) {
          fprintf(xbee.log,(j?":%02X":"%02X"),d[1+j]);
        }
        fprintf(xbee.log,"\n");
        fprintf(xbee.log,"XBee: 16-bit Address: ");
        for (j=0;j<2;j++) {
          fprintf(xbee.log,(j?":%02X":"%02X"),d[9+j]);
        }
        fprintf(xbee.log,"\n");
        fprintf(xbee.log,"XBee: AT Command: %c%c\n",d[11],d[12]);
        if (d[13] == 0) fprintf(xbee.log,"XBee: Status: OK\n");
        else if (d[13] == 1) fprintf(xbee.log,"XBee: Status: Error\n");
        else if (d[13] == 2) fprintf(xbee.log,"XBee: Status: Invalid Command\n");
        else if (d[13] == 3) fprintf(xbee.log,"XBee: Status: Invalid Parameter\n");
        else if (d[13] == 4) fprintf(xbee.log,"XBee: Status: No Response\n");
      }
      p->type = xbee_remoteAT;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = TRUE;
      p->IOPkt = FALSE;

      p->frameID = d[0];

      p->Addr64[0] = d[1];
      p->Addr64[1] = d[2];
      p->Addr64[2] = d[3];
      p->Addr64[3] = d[4];
      p->Addr64[4] = d[5];
      p->Addr64[5] = d[6];
      p->Addr64[6] = d[7];
      p->Addr64[7] = d[8];

      p->Addr16[0] = d[9];
      p->Addr16[1] = d[10];

      p->atCmd[0] = d[11];
      p->atCmd[1] = d[12];

      p->status = d[13];

      if (p->status == 0x00 && p->atCmd[0] == 'I' && p->atCmd[1] == 'S') {
        /* parse the io data */
        if (xbee.logfd) fprintf(xbee.log,"XBee: --- Sample -----------------\n");
        xbee_parse_io(p, d, 15, 17);
        if (xbee.logfd) fprintf(xbee.log,"XBee: ----------------------------\n");
      } else {
        /* copy in the data */
        p->datalen = i-13;
        for (;i>13;i--) p->data[i-14] = d[i];
      }

    /* ########################################### */
    /* if: TX status */
    } else if (t == 0x89) {
      if (xbee.logfd) {
```

```c
        fprintf(xbee.log,"XBee: Packet type: TX Status Report (0x89)\n");
        fprintf(xbee.log,"XBee: FrameID: 0x%02X\n",d[0]);
        if (d[1] == 0) fprintf(xbee.log,"XBee: Status: Success\n");
        else if (d[1] == 1) fprintf(xbee.log,"XBee: Status: No ACK\n");
        else if (d[1] == 2) fprintf(xbee.log,"XBee: Status: CCA Failure\n");
        else if (d[1] == 3) fprintf(xbee.log,"XBee: Status: Purged\n");
      }
      p->type = xbee_txStatus;

      p->sAddr64 = FALSE;
      p->dataPkt = FALSE;
      p->txStatusPkt = TRUE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      p->frameID = d[0];

      p->status = d[1];

      /* never returns data */
      p->datalen = 0;

    /* ########################################## */
    /* if: 16 / 64bit data recieve */
    } else if ((t == 0x80) ||
               (t == 0x81)) {
      int offset;
      if (t == 0x80) { /* 64bit */
        offset = 8;
      } else { /* 16bit */

        offset = 2;
      }
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Packet type: %d-bit RX Data (0x%02X)\n",((t == 0x80)?64:16),t);
        fprintf(xbee.log,"XBee: %d-bit Address: ",((t == 0x80)?64:16));
        for (j=0;j<offset;j++) {
          fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
        }
        fprintf(xbee.log,"\n");
        fprintf(xbee.log,"XBee: RSSI: -%ddB\n",d[offset]);
        if (d[offset + 1] & 0x02) fprintf(xbee.log,"XBee: Options: Address Broadcast\n");
        if (d[offset + 1] & 0x03) fprintf(xbee.log,"XBee: Options: PAN Broadcast\n");
      }
      p->dataPkt = TRUE;
      p->txStatusPkt = FALSE;
      p->modemStatusPkt = FALSE;
      p->remoteATPkt = FALSE;
      p->IOPkt = FALSE;

      if (t == 0x80) { /* 64bit */
        p->type = xbee_64bitData;

        p->sAddr64 = TRUE;

        p->Addr64[0] = d[0];
        p->Addr64[1] = d[1];
        p->Addr64[2] = d[2];
        p->Addr64[3] = d[3];
        p->Addr64[4] = d[4];
        p->Addr64[5] = d[5];
        p->Addr64[6] = d[6];
        p->Addr64[7] = d[7];
      } else { /* 16bit */
        p->type = xbee_16bitData;

        p->sAddr64 = FALSE;

        p->Addr16[0] = d[0];
        p->Addr16[1] = d[1];
      }

      /* save the RSSI / signal strength
         this can be used with printf as:
         printf("-%ddB\n",p->RSSI); */
      p->RSSI = d[offset];

      p->status = d[offset + 1];

      /* copy in the data */
      p->datalen = i-(offset + 1);
      for (;i>offset + 1;i--) p->data[i-(offset + 2)] = d[i];

    /* ########################################## */
    /* if: 16 / 64bit I/O recieve */
    } else if ((t == 0x82) ||
               (t == 0x83)) {
      int offset, samples;
      if (t == 0x82) { /* 64bit */
        offset = 8;
        samples = d[10];
      } else { /* 16bit */
        offset = 2;
        samples = d[4];
      }
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Packet type: %d-bit RX I/O Data (0x%02X)\n",((t == 0x82)?64:16),t);
        fprintf(xbee.log,"XBee: %d-bit Address: ",((t == 0x82)?64:16));
        for (j = 0; j < offset; j++) {
          fprintf(xbee.log,(j?":%02X":"%02X"),d[j]);
        }
        fprintf(xbee.log,"\n");
        fprintf(xbee.log,"XBee: RSSI: -%ddB\n",d[offset]);
        if (d[9] & 0x02) fprintf(xbee.log,"XBee: Options: Address Broadcast\n");
        if (d[9] & 0x02) fprintf(xbee.log,"XBee: Options: PAN Broadcast\n");
        fprintf(xbee.log,"XBee: Samples: %d\n",d[offset + 2]);
      }
      i = offset + 5;

      /* each sample is split into its own packet here, for simplicity */
      for (o = samples; o > 0; o--) {
        if (xbee.logfd) {
          fprintf(xbee.log,"XBee: --- Sample %3d ------------\n", o - samples + 1);
        }
        /* if we arent still using the origional packet */
        if (o < samples) {
          /* make a new one and link it up! */
          q = Xcalloc(sizeof(xbee_pkt));
          p->next = q;
          p = q;
        }

        /* never returns data */
        p->datalen = 0;

        p->dataPkt = FALSE;
        p->txStatusPkt = FALSE;
        p->modemStatusPkt = FALSE;
        p->remoteATPkt = FALSE;
        p->IOPkt = TRUE;

        if (t == 0x82) { /* 64bit */
          p->type = xbee_64bitIO;

          p->sAddr64 = TRUE;

          p->Addr64[0] = d[0];
          p->Addr64[1] = d[1];
          p->Addr64[2] = d[2];
          p->Addr64[3] = d[3];
          p->Addr64[4] = d[4];
          p->Addr64[5] = d[5];
          p->Addr64[6] = d[6];
          p->Addr64[7] = d[7];
        } else { /* 16bit */
          p->type = xbee_16bitIO;

          p->sAddr64 = FALSE;

          p->Addr16[0] = d[0];
          p->Addr16[1] = d[1];
        }

        /* save the RSSI / signal strength
           this can be used with printf as:
           printf("-%ddB\n",p->RSSI); */
        p->RSSI = d[offset];

        p->status = d[offset + 1];

        /* parse the io data */
        i = xbee_parse_io(p, d, offset + 3, i);
      }
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: ----------------------------\n");
      }

    /* ########################################## */
    /* if: Unknown */
    } else {
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Packet type: Unknown (0x%02X)\n",t);
      }
      p->type = xbee_unknown;
    }
    p->next = NULL;

    /* lock the connection mutex */
    pthread_mutex_lock(&xbee.conmutex);

    con = xbee.conlist;
    hasCon = 0;
    while (con) {
      if (xbee_matchpktcon(p,con)) {
        hasCon = 1;
        break;
      }
      con = con->next;
    }

    /* unlock the connection mutex */
    pthread_mutex_unlock(&xbee.conmutex);

    /* if the packet doesn't have a connection, don't add it! */
    if (!hasCon) {
      Xfree(p);
      if (xbee.logfd) {
        fprintf(xbee.log,"XBee: Connectionless packet... discarding!\n");
      }
      continue;
    }

    /* lock the packet mutex, so we can safely add the packet to the list */
    pthread_mutex_lock(&xbee.pktmutex);
    i = 1;
    /* if: the list is empty */
    if (!xbee.pktlist) {
      /* start the list! */
      xbee.pktlist = po;
    } else {
      /* add the packet to the end */
      q = xbee.pktlist;
      while (q->next) {
        q = q->next;
        i++;
      }
      q->next = po;
    }

    if (xbee.logfd) {
      while (q && q->next) {
        q = q->next;
        i++;
      }
      fprintf(xbee.log,"XBee: --========================--\n");
      fprintf(xbee.log,"XBee: Packets: %d\n",i);
    }

    po = p = q = NULL;

    /* unlock the packet mutex */
    pthread_mutex_unlock(&xbee.pktmutex);
  }
}

/* #################################################################
   xbee_getByte - INTERNAL
   waits for an escaped byte of data */
static unsigned char xbee_getByte(void) {
  unsigned char c;

  ISREADY;

  /* take a byte */
  c = xbee_getRawByte();
  /* if its escaped, take another and un-escape */
  if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;

  return (c & 0xFF);
}

/* #################################################################
   xbee_getRawByte - INTERNAL
   waits for a raw byte of data */
```

```
static unsigned char xbee_getRawByte(void) {
  unsigned char c;
  fd_set fds;

  ISREADY;

  /* wait for a read to be possible */
  FD_ZERO(&fds);
  FD_SET(xbee.ttyfd,&fds);
  if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
    perror("xbee:xbee_listen():xbee_getRawByte()");
    exit(1);
  }

  /* read 1 character
     the loop is just incase there actually isnt a byte there to be read... */
  do {
    if (read(xbee.ttyfd,&c,1) == 0) {
      usleep(10);
      continue;
    }
  } while (0);

  return (c & 0xFF);
}

/* #################################################################
   xbee_send_pkt - INTERNAL
   sends a complete packet of data */
static void xbee_send_pkt(t_data *pkt) {
  ISREADY;


  /* lock the send mutex */
  pthread_mutex_lock(&xbee.sendmutex);

  /* write and flush the data */
  fwrite(pkt->data,pkt->length,1,xbee.tty);
  fflush(xbee.tty);

  /* unlock the mutex */
  pthread_mutex_unlock(&xbee.sendmutex);

  if (xbee.logfd) {
    int i;
    /* prints packet in hex byte-by-byte */
    fprintf(xbee.log,"XBee: TX Packet - ");
    for (i=0;i<pkt->length;i++) {
      fprintf(xbee.log,"0x%02X ",pkt->data[i]);
    }
    fprintf(xbee.log,"\n");
  }

  /* free the packet */
  Xfree(pkt);
}

/* #################################################################
   xbee_make_pkt - INTERNAL
   adds delimiter field
   calculates length and checksum
   escapes bytes */
static t_data *xbee_make_pkt(unsigned char *data, int length) {
  t_data *pkt;
  unsigned int l, i, o, t, x, m;
  char d = 0;

  ISREADY;

  /* check the data given isnt too long
     100 bytes maximum payload + 12 bytes header information */
  if (length > 100 + 12) return NULL;

  /* calculate the length of the whole packet
     start, length (MSB), length (LSB), DATA, checksum */
  l = 3 + length + 1;

  /* prepare memory */
  pkt = Xcalloc(sizeof(t_data));

  /* put start byte on */
  pkt->data[0] = 0x7E;

  /* copy data into packet */
  for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
```

```c
    /* if: its time for the checksum */
    if (i == length) d = M8((0xFF - M8(t)));
    /* if: its time for the high length byte */
    else if (m == 1) d = M8(length >> 8);
    /* if: its time for the low length byte */
    else if (m == 2) d = M8(length);
    /* if: its time for the normal data */
    else if (m > 2) d = data[i];

    x = 0;
    /* check for any escapes needed */
    if ((d == 0x11) || /* XON */
        (d == 0x13) || /* XOFF */
        (d == 0x7D) || /* Escape */
        (d == 0x7E)) { /* Frame Delimiter */
      l++;
      pkt->data[o++] = 0x7D;
      x = 1;
    }

    /* move data in */
    pkt->data[o] = ((!x)?d:d^0x20);
    if (m > 2) {
      i++;
      t += d;
    }
  }

  /* remember the length */
  pkt->length = l;

  return pkt;
}
```