Chanoch Wiggers <chanoch.wiggers@gmail.com>

Livetrack24 UDP protocol

Manolis Andreadakis <manolis@andreadakis.gr> Fri, Oct 19, 2012 at 6:38 PM
Reply-To: manolis@andreadakis.gr
To: email@chanoch.com

Hi Chanoch,

actually it is a bit more complicated than that:
have a look at the spec and the rar file with sample code:
i hope we can have UDP gaggle sometime :-)

```
Hi,

It says here that Gaggle supports UDP live tracking - did you get this from Kevin and do you think it's
aspirational or is it work already done?

Regardless, is there a spec I can have to see if I can implement this as there are a number of the Gaggle XC
pilots who would really like this. Doesn't look to difficult if it's just a matter of sending a GPS point per UDP
packet?

Chanoch
```

Here is the info the UDP/TPC session aware protocol :
(i am using the word midlet since my java app is a midlet)

the IP to use for testing is 188.40.246.133 port 995 UDP (both UDP / TCP)
and the tracks are displayed at test.livetrack24.com

in the zip file are the relevant classes and a sample code to send the packets is this

We can login into the server with user/name password. Logged in users have more functionality like triggering their predefined SMS alerts and more stuff. an account is not mandatory though , the user has 2 options:
a) create an account and then login once from midlet. his userID will be stored in the middlet and the sessionID that is send with each gps packet has his userID embedded in it. thus even if the first live packet that has login information is lost, the server still knows the userID of the packets.
b) dont use an account. the sessionID is a random 4byte long and the user/password pair is send with the first packet to the server.
all other packets are linked to the user via the unique random sessionID. if the first packet is lost, the user is displayed as guest.
the recommended way to send the first packet is this:
- use TCP instead.
- expect OK as an answer. if not retry at predefined intervals. not too many / not too frequent : TCP can be much more verbose, a TCP packet can be send back and forth many times before it timeouts, so a simple 100 bytes TCP packet can generate even 1-2 kb of GPRS traffic.
here is my intervals:

```
// send login intervals in seconds:
// 2 min,2 min,5 min,10 min,30 min,30 min,30 min
// total 6 retries ( the first one is not counted )
int[] sendLoginIntervals={120,120,300,600,1800,1800,1800};
```

I'll EXPLAIN METHOD B) , but A) is similar, you only set the sessionID in a format that the server knows it is a registered user

HERE IS a workflow:

Sending the first (login) packet
GpsTrackLive.sNet is an static instance of LiveNet, the first thing that does in the run() is send the login.

```
GpsTrackLive.sNet.open();
```

You app must monitor the success of the sending of the logjn packet with some code and retry like that:

```
if (!GpsTrackLive.sNet.loginHasBeenSent) {
    // send login intervals :
    // 2 min,2 min,5 min,10 min,30 min,30 min,30 min
    // total 6 retries ( the first one is not counted )
    int[] sendLoginIntervals={120,120,300,600,1800,1800,1800};

    //int[] sendLoginIntervals={20,10,300,600,1800,1800,1800};

    // now is a good change to resend  login info...
    if (GpsTrackLive.sNet.sendLoginTimes < sendLoginIntervals.length ) {
```

```
                        //GpsTrackLive.logMsg(MIDPLogger.INFO,"sendLoginTimes: "+GpsTrackLive.sNet.sendLoginTimes
                        //        + " resendLoginTimer: " +resendLoginTimer);

                        if ( resendLoginTimer > sendLoginIntervals[GpsTrackLive.sNet.sendLoginTimes]* 1000  ) {

                            GpsTrackLive.logMsg(MIDPLogger.INFO,"Will resend Login information #"+GpsTrackLive.sNet.
sendLoginTimes);
                            resendLoginTimer =0;
                            GpsTrackLive.sNet.sendLogin();
                        }
                    }
                }
```

Now on to gps point gathering and sending....

in the normal GPS/ TRACK loop we first STORE n gps points (n=1-5) with this :

```
                short cog;
                byte sog;
                byte[] netBytes=null;

                try {
                    sog=(byte)gps.getSpeed().toLong();
                    cog=(short)gps.getHeading().toLong();
                    netBytes=PrepareMsg.prepareMessage(tm,lon,lat,alt, sog, cog);
                } catch ( Exception ex01) {
                        GpsTrackLive.logMsg(MIDPLogger.ERROR,"Track.run: Exception 01:" + ex01.getMessage());
                }

                try {
                    // store this in the buffer
                    if (netBytes!=null){
                        netStore[netPointsStoredTmp]=netBytes;
                        netPointsStoredTmp++;
                        netPointsStored++;
                        GpsTrackLive.logMsg(MIDPLogger.DEBUG,"Track.run: netPointsStoredTmp: "+netPointsStoredTmp);

                    } else {
                        GpsTrackLive.logMsg(MIDPLogger.ERROR,"Track.run: netBytes is null ");
                        continue;
                    }

                } catch ( Exception ex02) {
                    GpsTrackLive.logMsg(MIDPLogger.ERROR,"Track.run: Exception 02:" + ex02.getMessage());
                }
```

when we gather the `mSendEveryN` packets it is time to send them ....
SPECIAL CASE: if it is the first point , we send it right away, we want the position of the user to be available on the web as soon as possible.

```
                // now decide if we store this record or send it
                // take special care for the first gps fix : send right way
                if ( netPointsStoredTmp == mSendEveryN   || packetsTransmitted==0  ) {
                    // the time has come, send the buffer and reset recordsStored
                    /*
                    header 1st byte                  DEFAULT VALUE of GPS points
                    1   1=gps 0= info                     1 // GPS point
                    2   1=request 0=signal                0 // signal
                    3   binary=1 text=0                   1 // binary
                    4   1=simple header 0 = extended      0 // extended
                    5   1= no ack required 0=> send ack +status    1 // no ack
                    6-8    0-7 gps packets num ( +1)      3 // means there are (3+1)= 4 gps points in this packet
                           so is 1-8 packets num // or type of text 1->login



                    Next 4 bytes:  sessionID
                    Next 2 bytes: packetID

                    So the header of each packet is  1 + 4 + 2 = 7 bytes

                    the gps data for each point is 16 bytes
                    # 4  4   4    2   1   1
                    # tm lat lon alt sog cog

                    */
                    //                                    isGPSdata,isRequest,isBinary,isSimpleHeader,noAck,packetNum
                    byte[] headerBytes=LiveProtocol.preparePacketHeader((byte)1,(byte) 0,(byte)1,(byte) 0 ,(byte) 1,(byte)
(netPointsStoredTmp-1)  );

                    byte[] msgBytes=PrepareMsg.concat(headerBytes,netStore,netPointsStoredTmp);

                    GpsTrackLive.logMsg(MIDPLogger.DEBUG,"Track.run: Send packet # "+packetsTransmitted );

                    // log the packetID and time
                    preferences.lastNotTerminatedSessionID=LiveProtocol.sessionID;
                    preferences.lastPacketID=LiveProtocol.packetID;
                    preferences.lastPacketTM=(int)(System.currentTimeMillis()/1000);
                    preferences.saveRunning();

                    latestNetBytes=msgBytes;


                    if ( GpsTrackLive.sNet.sendNetMsg(msgBytes)  ) {
                        packetsTransmitted++;
                    } else {
                        packetsError++;
                    }
                    netPointsStoredTmp=0;

                }
```

after the track finishes we need to also let the server know that and also ask the user if he had any problems...
(good for extreme sports etc....)

see **SendEndOfTrack** class for the code that does that....

i think this pretty much covers the basic functionality

cheers,
Manolis

protocol.rar
15K