Data Transfer Over GPRS: Palm OS

By: Prashant Batra
Mindfire Solutions
http://www.mindfiresolutions.com

Table of Contents

Overview

Transfer of files from a Palm device (aka. Client) to a server running on a PC (aka. Server) can be achieved in different ways: FTP, HTTP, and TCP/IP. In all these cases, one thing remains common, the channel of transfer, that is "The Cellular Network". Though it may sound as easy as opening a socket and sending the file data across to server, when you are dealing with cellular networks, things are a bit different.

The goal of the document is to bridge the gap between academics and experience. This document delves into our experience with file transfer over the GPRS network using socket programming for Palm OS based devices, so that anyone else facing similar issues can handle similar problems quickly and effectively, or better, design the system keeping these in mind.

Abstract

We highlight the issues that software architects should consider when designing applications depending on GPRS service.

Introduction

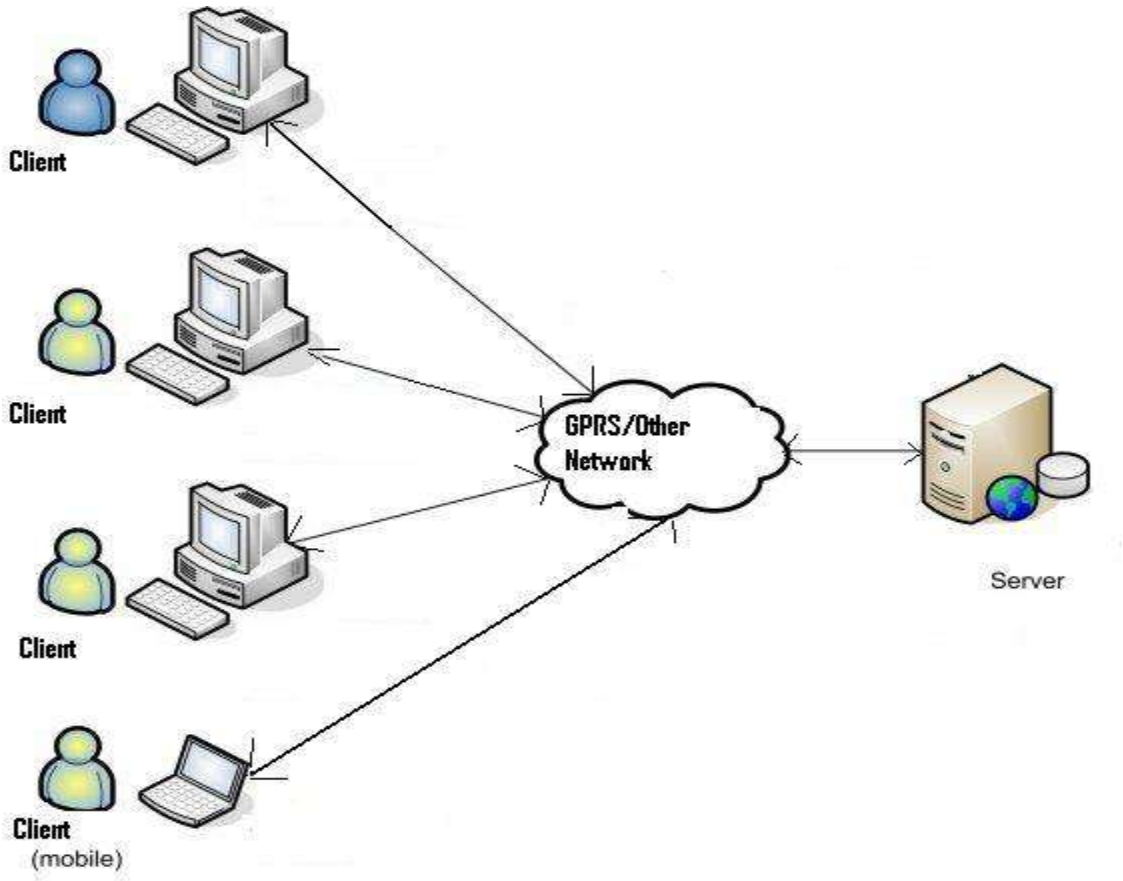
The past few years have witnessed substantial growth in cellular telephony, which has also lead to an increased demand for wireless data services. Although now an agreed standard, so-called 3rd generation (3G) UMTS (Universal Mobile Telecommunications System) mobile networks are still some time away from wide-scale deployment. In the meantime, the GPRS (General Packet Radio Service) extension to current 2G GSM networks looks set to provide a widely deployed solution for wireless data access.

In particular, we examine the behavior of TCP, the dominant protocol in the wired Internet, and one that is likely to be similarly important for mobile users too.

We suggest few measures that may be taken and what things need to be considered while working with GPRS network.

Challenge

Our application implements two-way communication, designed to transfer files between client and server applications, in a "behind the scenes" mode, over the cellular network.



General Client – Server Architecture

The must have features for such applications are:

Reliability

Both ends of the application should be reliable. Client should send what it is supposed to or notify the user in case any error occurs. Server should receive the data and should be able to re-create the same exact file at its end. Situations may arise when server gets garbage values from the network transmission. This should be handled appropriately by validating the received data. Also, reliability requires a solution that is based on established standards.

Security

Transferring the file from one system to another is only half battle won. At a time when businesses and government agencies alike are increasing their focus on security; protecting the information as it is transferred has become paramount.

Organizations face a challenge to promote the sharing of data across the enterprise by providing their staff with a method for transferring files via the Internet that is secure, reliable and cost-effective.

A secure method guarantees that sensitive data is protected at every point of the transfer, from start to finish.

Speed

It is a well-known fact that when it comes to cellular networks, speed has always been on the slower side. The application should be designed to run at an optimal speed as any unwanted / unexpected delays can cause random problems.

Communications Integrity

Suppose you had to send large files, e.g. recording files (WAV and other formats). It is critical that the file transferred is good one and that the application provides a reliable integrity.

Common Solution

Let us now look at a possible answer to our requirements stated above.

- Robustness: Implement Handshaking algorithm that will enable the applications to realize if other party is down or there is a problem in communication channel, hence enable us to handle the error conditions appropriately.

- Reliability: Send the checksum with each packet. Adding a header to each packet having custom data also ensures security and reliability to some extent, as after looking at the header, server can decide whether the arrived packet is valid or not, and recreate the file appropriately by managing the packet sequencing.

- Speed: Do not put any intentional delay in sending various data packets.

- Security: Encrypt the sensitive data, if required.

Putting it all together, a common solution that we may plan out is:

On the client application, read the file, break it into packets and send the packets with appropriate checksum and header. Before sending the next packet, wait to receive the ACK (acknowledgment) message of last packet from Server, to ensure that packet was successfully delivered.

On the server side, take large buffer; keep on adding the received packets to it. Process the buffer, validated each packet using the checksum. Arrange the packet by taking information from header. Send ACK to client informing the successful packet received and save the packets to a file. That's all there is to it.

Great!! We are half way through, as we have conceptualized the logic in our mind and it should work…

But wait! This is not the end. There are many pitfalls and unknowns, learnt through experience, which we would like to share at this point.

Things To Consider

Let us start from the beginning; In Palm, all the operations related to sending/receiving data are done using Network Library. We must know its architecture to understand its functionality.

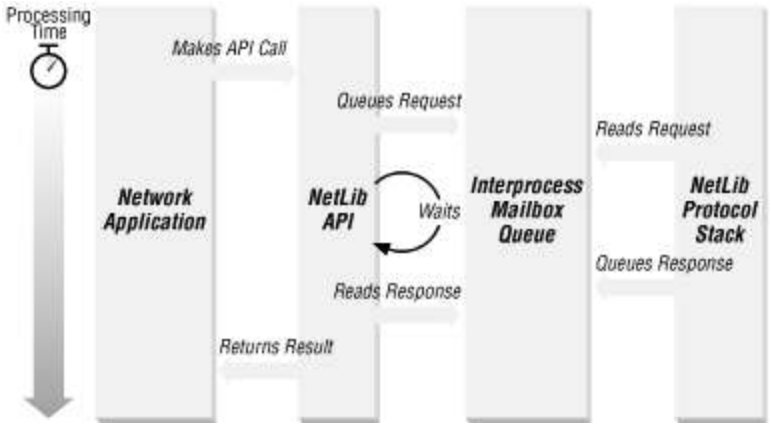
The Design of the Net Library

The Net Library provides networking services for the Palm OS. The standard implementation supports ***TCP*** and ***UDP*** over an IP network.

Architecture

The Network Library can be divided into two parts: the API and the protocol stack. The API is a set of functions that an application invokes to access the Net Library's functionality. The protocol stack handles all communication with the network: connecting, sending, and receiving.

These components are connected by a mailbox queue. Following figure illustrates these relationships.



In this diagram, we see the client application invoking functions in the API. The API translates these into requests that are placed into the mailbox queue. The protocol stack reads the requests and talks to the network. The protocol stack then places a response into the mailbox queue. The API reads the mailbox queue and returns a result to the client application.

Lets dig deep…

Sending Data

The most common way of sending data in Palm is to call NetLibSend. When you open a socket, specify whether it is to be a stream or datagram socket.

For datagram sockets that were not connected to a specific address, you need to pass the destination address as a parameter. For all other types of sockets, this parameter is ignored. When you send data, the protocol stack stores the data in buffers to be sent out later, and returns. **If there is insufficient room in these buffers for the amount of data you send, the protocol stack stores as much of the data as it can fit.**

For stream-based sockets, NetLibSend returns the amount of data accepted by the protocol stack to be sent. **It is the application's responsibility to keep track of how much data was sent.** For datagram sockets, all the data must be sent. **In either case, if there is no buffer available for the data, the function call waits until the space is available.**

For a stream-based socket, a single send operation can generate multiple packets. The remote host sends an acknowledgment for each packet sent. If the acknowledgment is not received within a certain amount of time, the protocol stack resends the data. Until the protocol stack receives this acknowledgment, it will not send the next packet.

Datagram-based sockets generate exactly one packet for each send operation. Once the packet is sent out, the protocol stack releases the buffers used to store the data; the UDP protocol does not support acknowledgments of packets. If the data from a send operation is too big for a UDP packet, no data is sent and an error is returned.

NOTE: It is very much essential that you check the return value of a send operation. This value indicates the number of bytes actually sent, or, more accurately, the number of bytes buffered to be sent by this operation. For stream-based sockets, it is possible that the protocol stack was unable to buffer all this data. In the case of large amounts of data being sent, it is even likely.

Developers must take the responsibility to resend the unsent / unwritten data, until all of it has been sent or an error is encountered.

In order to make our application we decided to send the next packet after receiving the ACK for the last one. No doubt this is a good option for ensuring the robustness but not a clever one.
Getting ACK after each packet ensure reliability it has to be understood that waiting for ACK after each packet affects the speed of whole process adversely.

By our experience, **TCP has proved to be quite reliable connection even on GPRS**. The chances of data loss are minute.

Stream sockets in Palm use the Transmission Control Protocol (TCP) to provide a "virtual circuit" between the two connected hosts. Packets are guaranteed to arrive in order, exactly once. This protocol is geared towards a world where the network is relatively fast but not necessarily reliable. TCP is a "chatty" protocol, in which the chattiness is designed to ensure reliability.

Datagram sockets use the User Datagram Protocol (UDP). This protocol is connectionless. Packets may arrive in any random order. They may arrive once, more than once, or not at all. This protocol is good for situations where the network reliability is not a question or where the protocol that sits on top of UDP ensures the correct delivery of packets. UDP is common in wireless applications because of its low overhead.

As we want robust application so we should target Stream socket and as the TCP ensures reliability, there is no need to check ACK after each packet.

A much better approach is that the server keeps validating the received packets and stores the serial numbers. When all the packets are received, it can return the serial numbers of bad packets back to the client.

Even if a packet goes missing, server can look into the serial numbers received and request the client to resend the lost packets.

So if you want to increase the speed of transfer **you can drop the idea of ACK after each packet using TCP but can use after full file transfer, to ensure that no loss occurs in any case.**

Be ready to face this too

Devices can process data much faster than the networks. Hence, it will happen that your program is continuously writing data on to the stream but the network is not able to process it at same pace.
This problem shall show up frequently when we try to blindingly send the data without waiting for an ACK for each packet, which we had to do in order to increase the transfer speed.

Generally before writing data to the socket, always check that it is available for writing. If not, then wait for a fixed interval, say 1 second, repeatedly, and recheck whether it is free, until it is free, or you reach your timeout limit.
This technique is widely used in case of sending file via FTP as in that case you wont receive any ACK for the packet sent.

In palm we can do this as follows:

```
Uint16 SocketReady2Write(AFTP_DATA *aftp, UInt16 sRef)
{
  Err err = 0;      // Return value.
  NetFDSetType readFDs, writeFDs, exceptFDs;
  UInt16 rc, width, numFDs;

  netFDZero (&readFDs);          // setting the value of readFDs to 0
  netFDZero (&writeFDs);         // setting the value of writeFDs to 0
  netFDZero (&exceptFDs);        // setting the value of exceptFDs to 0

  netFDSet (sRef, &writeFDs); // setting that write descriptor should be checked

  // Calculate max width.
  width = sRef + 1;

  // Wait for one of the descriptors to be ready.
  numFDs = NetLibSelect (aftp->netRef, width,&readFDs, &writeFDs,
                                  &exceptFDs,
                                 0, &err);
if (numFDs == 1) {
     rc = 1;
  } else {
     rc = 0;
  }
  return(rc);
}
```

This function returns true if the socket is free for writing.

In a similar fashion, we can use "**netFDSet** (sRef, &readFDs);" to check whether socket is ready for reading.

Slow GPRS

GPRS is agonizingly slow sometimes. It may also be delayed due to some other higher priority application in need of data usage, like in the case of circuit-switched calls. There may be cases when function may take quite a long time to complete its task. If ***timeout*** value is reached, you shall get **netErrTimout** error.

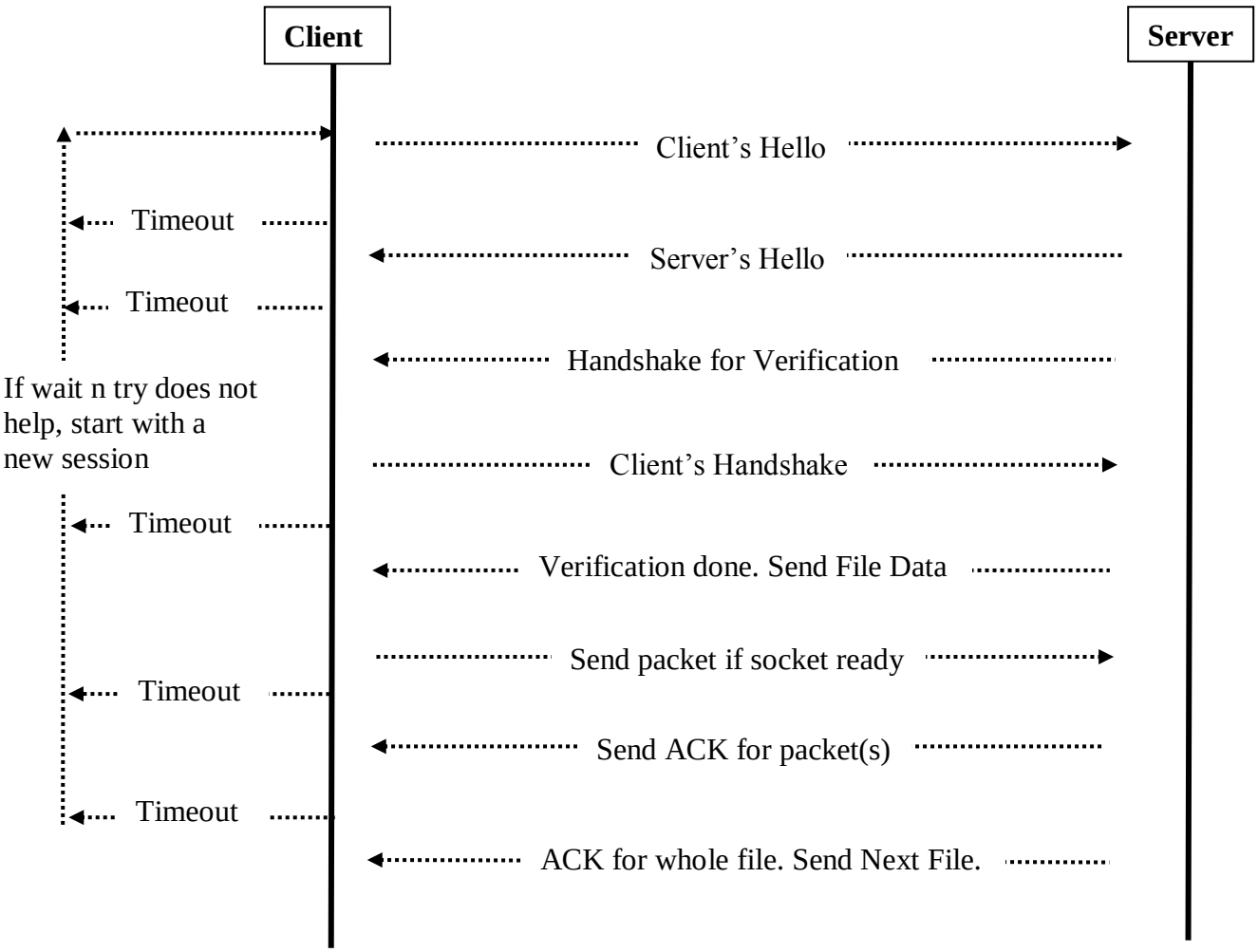
What is a timeout?
Most Net Library functions take two parameters - a timeout value and an error parameter. In the Palm environment, there is only a single user interface task. As long as a network function is blocking, no other user interface code can run. There is no smooth way to kill any running / blocking process. If a network function takes too long to execute and return, the device appears to be in a hung state. The timeout parameter specifies a limit within which the network function MUST return. If it takes more time, then it returns the netErrTimeout error.

Hence, the timeout value must be aptly chosen to suit your application needs. Be sure that its not so small that you get error a lot of times, and also that its not so high that your application has to wait for very long durations.

Even after doing all the calculations and making all the estimates, there are chances you will still encounter netErrTimeout. In that case you can wait for some time and try again. If that does not help, then consider closing the socket and using a fresh connection.

Summary

The solution can be summarized as follows:



Conclusion

In this document, we have discussed various things that developers should consider while developing data transfer applications using GPRS.

GPRS being a slow network, we discussed how to take care of the problematic areas.

Hoping that many developers who are working or are going to work on similar applications, shall benefit from our experience and knowledge sharing.

For any queries or help, you can always contact the Palm developer team at Mindfire Solutions.