

The value of a packet logger in wireless development – a story from real life

Scope

The scope of this document is to explain why it is important to have access to a packet logger during development of wireless solutions. The intended audience is technical personnel and technical decision makers who are responsible for the tools supporting the development process.

Introduction

Development of wireless solutions has many commonalities with client-server development. In both kinds of systems, events originate on one system and propagate to another system. Further packing and unpacking of commands and data to different formats or structures takes place on every event. Another common factor is that both kinds of development involve dealing with the intended receiver of the event.

Some functionality is implemented directly by the developers, in addition to the core functionality, typically involving translation between your own data structures and the structures used by the API. This is done in the application layer or in other layers high up in the protocol stack. The structures are then again handled by the stacks and operating systems that prepare them for transmission over a network, either wirelessly or through cables. Again this involves moving data between data structures and formats. On the receiving site the whole process is carried out again, but in reverse order. The packing and unpacking on several layers decreases visibility of what is happening with our original data and commands and makes it more difficult to debug the system if the intended response is not generated.

Message flow in wireless systems

Let's have a look of the message flow and the visibility of this flow in a wireless system. Figure 1 illustrates the message flow and which part of the flow can be monitored by the developer.

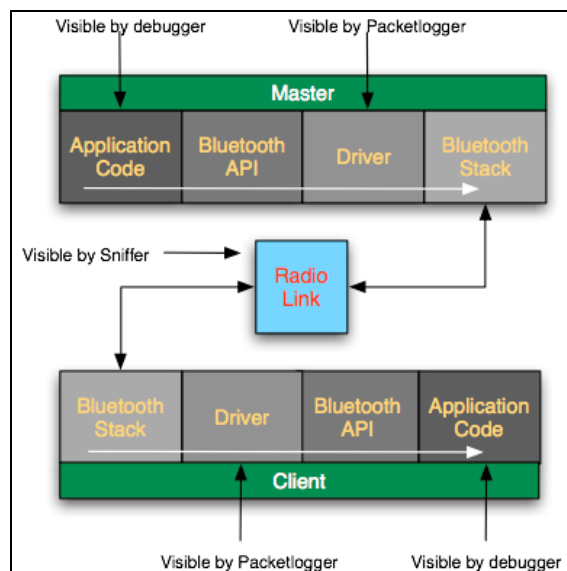


Figure 1

Monitoring what is happening in the code is essential for an efficient development. Usually the developer turns to the debugger to monitor the message flow when the application is not behaving as expected. When we look at Figure 1 again, we can see that debugging tools can be used on the ends of the system, but as soon as a message is received by the API to the stack or operating system we usually lose the possibility to monitor the processing of the events in a meaningful way because the stack is often a precompiled library for which we only seldom get access to the source code.

This might turn out to be a problem in cases where nothing is happening on the receiving side. Soon questions arise as to whether data has been sent and what data has been sent. The reasons for these 92 problems could be in the users' code on each side but it could just as well be hidden in the stack on one the side. If the data flow between the client and server cannot be monitored a lot of time may be wasted because we are basically seeking something unknown without the ability to see what is going on.

A packet logger is the tool we need to use to be able to watch what is happening in the entire system. The following is a practical example of how a problem can be solved by using a packet logger.

The Agenda Pusher

The agenda pusher is a system that is able to push a text file to a Bluetooth enabled device when it comes in range.

The application works by registering devices in range. These devices are filtered and a connection is established to each device that has not yet received the text file. As soon as the connection is established the text file is transferred to the mobile device and the result of the operation is registered. The application is quite simple and features some common functions of wireless applications such as detecting devices in range, establishing a connection, transmitting data and closing the connection.

Problem

During the development of the agenda pusher I discovered that the first connection to a device was taking place as expected. The mobile phone presented me with a message box where I had the options to accept or reject the message. If I rejected the message for some reason the application would give me another option to receive the message. This second attempt would fail in most cases. The symptoms was that nothing was happening but through further inspection of the mobile phone I could see that some sort of connection actually was established, as the Bluetooth indicator changed to connected, but the message box offering the option to receive or reject was not presented.

These kinds of problems are known by most to be quite frustrating. When the application works everything is right. This is clearly an indication that something is right, but it fails at the second attempt so obviously something is still wrong, but where?

Expected Result

Let us have a look at what we would expect to take place when we initiate a Bluetooth connection.

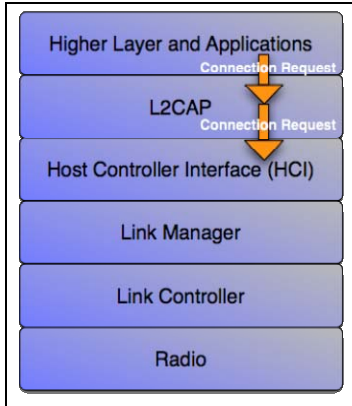


Figure 2

The connect message from the application is sent to the L2CAP layer. The L2CAP layer checks if there is an existing ACL connection and if not it will request the HCI layer to establish a connection.

This is done on both the master and the client side and ends up with a L2CAP connection request from the master to the L2CAP layer on the client. These messages are the key points in establishing a connection and it is easy to verify if they are exchanged.

Actual Result

In order to find out why I did not get a full connection when I connected to the phone the second time, I used a Bluetooth packet logger to see the actual data exchange that was taking place. I wanted to capture the Bluetooth data packages when the application worked successfully during the first connection and when it failed during the second connection in order to compare these to logs to see if something unexpected was happening. My expectations would be to see a similar package structure because I performed a full disconnect between each transmission.

Figure 3 shows the communication that was taking place. The unmasked part is difference of interest.



Figure 3

As can be seen the first three lines are identical but in line four there is a difference compared to Figure 7. The log of the successful transmission contains a connection request on the L2CAP layer that is never sent when the application fails.

The reason for the problem is now identified and we can actually start solving it.

Solution

The application makes a new connection from scratch every time it tries to send the text file because the connection is closed down after each transmission, as can be seen in the log of the disconnection in figure 4.

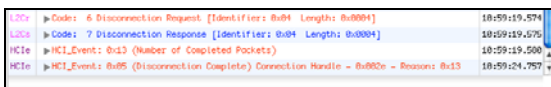


Figure 4

When we look at the logged data we can see the disconnection is complete and for some reason something either in the phone or in the Bluetooth stack fails to establish a full connection again.

```

NSLog(@"Connecting %@", [[device valueForKey:@"bluetoothDevice"] getAddressString]);
if ([[obex connectToObjectPushService] != KOBEXSuccess)
{
    NSLog(@"OBEX Connection error");
}
}

```

Figure 5

The code in Figure 5 makes the connection and the API call reports back that the connection attempt is successfully registered by the Bluetooth stack. According to the documentation the call connectToObjectPushService on the obex object should establish the connection if it is not currently established. This includes everything all the way down from the OBEX session to the Bluetooth connection.

In order to ensure that the connection is fully established on the L2CAP level too I tried to force the connection on the Bluetooth object with the call openConnection as seen in figure 6.

```

[[btDevice openConnection];
NSLog(@"Connecting %@", [[device valueForKey:@"bluetoothDevice"] getAddressString]);
if ([[obex connectToObexPushService] != KOBEXSuccess)
{
    NSLog(@"OBEX Connection error");
}
}

```

Figure 6

As a result the connection is established every time and the problem was solved as seen in Figure 7.



Figure 7

Conclusions

During the development of an application incorporating wireless technology, in this case a Bluetooth connection between computer and a mobile phone, I encountered unexpected behavior.

The debugging of the code I developed did not show any errors and the status messages I got back through the API from the Bluetooth stack did not report any problem, but still I could see that some action was happening but not as expected.

I used a packet logger to monitor the communication on the Bluetooth connection and discovered that a connection package was not sent for some reason, which I did not investigate.

Forcing a connection on the Bluetooth object in use solved the problem.

The packet logger provided me with the information I needed to solve the problem and without it a problem like this would have been very hard to solve and would have taken a lot of time. Ultimately I would have to involve the vendor of the Bluetooth stack or of the mobile phone. In that case I would have been very dependent of the priority they would assign to my problem.

I conclude that the packet logger is a very important tool to have access to. If a company does carry out a wireless project it must be ensured that a packet logger is delivered with the stack or can be acquired separately.

If a proprietary solution is chosen for the communication protocol, time should be set aside to create a packet logger for the rest of the development team to use.

Any questions or comments relating to this article? Kindly contact Thomas Sørensen, ths@delta.dk.