Scatternet - Part 1

Baseband vs. Host Stack Implementation

White paper

Ericsson Technology Licensing
June 2004

Contents

1. ....Abstract 3

2. ....Introduction 4

3. .... Scatternet user case 5

4. .... Interoperability issues 9

5. ....Architectural considerations 11

6. .... Conclusion 13

1. Abstract

When two or more independent, non-synchronized Bluetooth piconets overlap, a scatternet is formed in a seamless, ad-hoc fashion allowing inter-piconet communication. While the Bluetooth specification stipulates the use of time-division multiplexing (TDM) for enabling concurrent participation by a device in multiple piconets, it leaves the choice of actual mechanisms and algorithms for achieving this functionality open to developers. Some implementation proposals suggest that the host stack handle the complex inter-piconet communication of a scatternet. On the surface this may appear to be a simple and clean solution, but what dangers, if any, does this entail?

The purpose of this white paper is to illustrate the risks associated with solutions that pass scatternet communication operation on to the host stack. By doing so, this paper further intends to show why scatternet functionality is best implemented in the Bluetooth Control layers only, below the HCI interface.

2. Introduction

As with piconets, where multiple Bluetooth devices are able to connect with each other in an ad-hoc manner, so too can multiple piconets join together to form a larger network known as a scatternet. Bluetooth devices must have point-to-multipoint capability in order to engage in scatternet communication, and several piconets can be connected to each other through one scatternet. Furthermore, a single Bluetooth device may participate as a slave in several piconets, but can only be a master in one piconet.

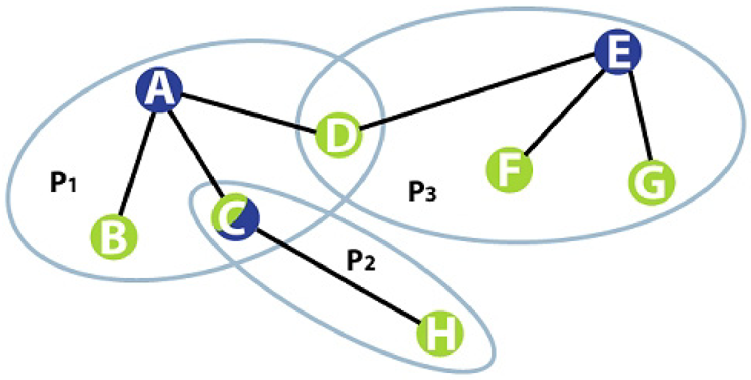

Figure 2.1: Example Bluetooth topology

Figure 2.1 shows an example of a scatternet consisting of three separate piconets, P_1, P_2 and P_3. Each piconet is controlled by a separate master (devices A, C and E) and contains one or more slaves. Note how device C, which connects P_1 and P_2, is a slave in one piconet (P_1) and a master in the other (P_2).

Giving rise to new complications
Realizing true scatternet functionality introduces complications concerning interoperability and timing that in fact can jeopardize a product's ability to remain fully compatible with the Bluetooth standard. In the following chapters we will examine these issues as illustrated through a common user scenario, as well as question the viability of relying on the host stack rather than the baseband for dealing with them.

The aim of this analysis is to demonstrate that solutions where scatternet communication operation is passed on to the host stack may be likened to icebergs – seemingly harmless on the surface, but indeed quite treacherous due to perils hidden in the depths below. The only way to navigate safely through these dangers is to avoid them altogether, i.e. by steering clear of solutions that support scatternet by allowing the host stack to control scatternet communication.

3. Scatternet user case

An example user case demonstrating the need for a clean, high quality implementation of scatternet is shown in Figure 3.1. Here we see a user synchronizing calendars between a phone and a notebook PC while at the same time carrying on a phone conversation using a Bluetooth headset. In this scenario the mobile phone functions as both a master (towards the headset) and a slave (towards the computer). In order for this to work, regardless of data speed, an effective scatternet implementation is required.

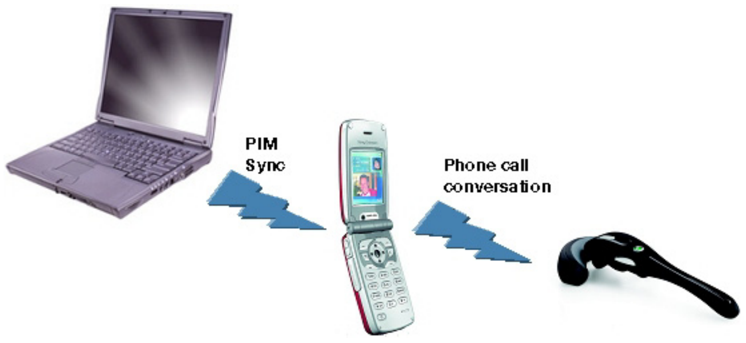

Figure 3.1: Example scatternet user case

Complex scenario
In the user case example above, the notebook PC could alternatively be a PDA. In either case, this is a complex situation where two applications that are intended for two different logical links are running concurrently. One involves data transmission and the other a voice link. Here we are immediately confronted with some difficult timing issues that arise because, in addition to the critical timing of the SCO-link where the phone is a master to the headset, there is also an ACL-link where the phone is a slave to the PC or PDA.

Limitations on phone as slave
To evaluate this user case, one must first look at what transpires when the mobile phone operates in active mode and sniff mode respectively. At first glance, the most logical choice for the phone to operate in while performing calendar synchronization with the computer is active mode. This means that there is an active link over which data is exchanged during scheduled periods of time.

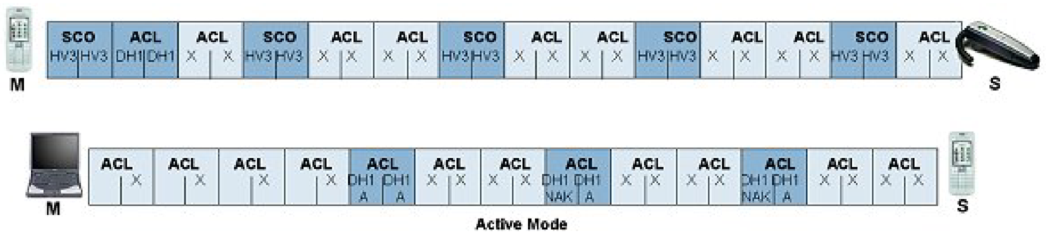

Figure 3.2: Packets and timing between scatternet connections in active mode

As illustrated in Figure 3.2, the voice link between the mobile phone and the headset is an SCO connection, meaning that the scheduled packets between the mobile phone and the headset are fixed in time. Due to potential clock drift overlap between the two master clocks, a phenomenon explained in the next section, the mobile phone only has time to communicate with the computer when there are at least two open slots between the phone and the headset.

When the phone (slave) attempts calendar synchronization in active mode, it must wait for acknowledgement from the computer (master) confirming receipt of a sent data packet (packet A). When acknowledgement is not received, the phone gets caught up in a loop of resending the same packet during every available communication window between the slave and the master.

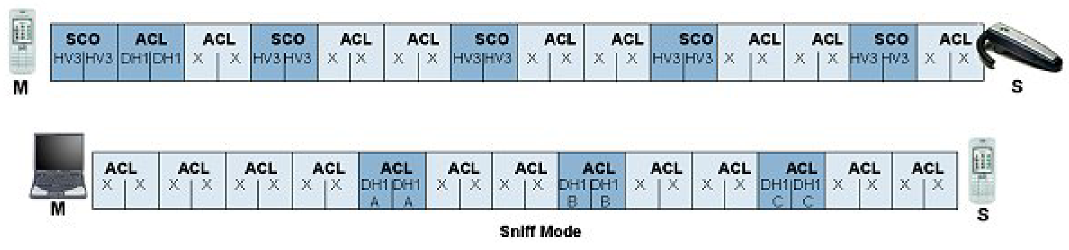

Figure 3.3: Packets and timing between scatternet connections in sniff mode

In sniff mode, shown in Figure 3.3, the computer sends out data while the phone performs regular checks to see whether or not there is anything being sent to it. When the answer to this question is yes, acknowledgement is duly received and the exchange of data (packets A, B, C, etc.) takes place.

Because of the above-described restrictions that are placed on the phone when operating as a slave in active mode, it is clear that calendar synchronization between the mobile phone and the computer can only be handled through implementation of scatternet using sniff mode.

Adjusting for clock drift
"Since the clocks of two masters of different piconets are not synchronized, a slave device participating in two piconets shall maintain two offsets that, added to its own native clock, create the two master clocks. Since the two master clocks drift independently, the slave must regularly update the offsets in order to keep synchronization to both masters."[1]

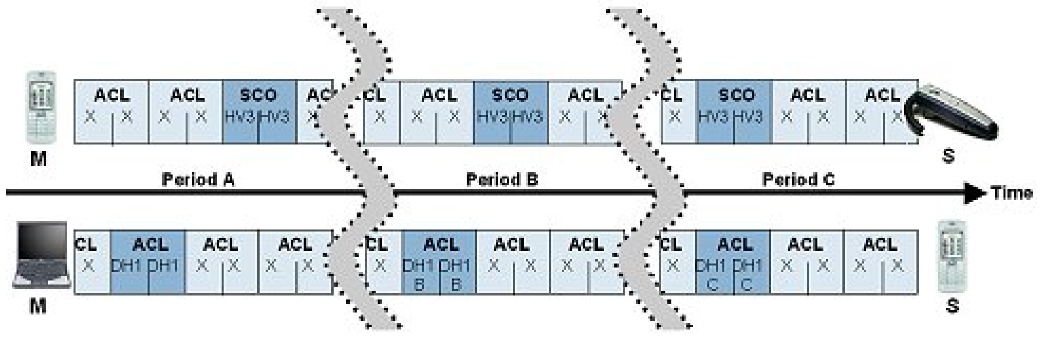

Figure 3.4: Clock drift overlap between the two master clocks

By observing snapshots of our user case scatternet communication taken at three different times (Periods A, B and C), we can see in Figure 3.4 how clock drift overlap between the phone call and calendar synchronization occurs over a period of time. In order for the phone and headset to maintain the necessary timing demands for a good audio link over Bluetooth, clock drift of the calendar synchronization link must not be allowed to interfere with the fixed and scheduled SCO-link slots as it does in Period C.

As a slave to the computer, the mobile phone must immediately adjust for the clock drift in the computer. In order for the phone to be able to do this, its host stack must pause the SCO-link with the headset to perform un-sniff followed by a sniff function towards the computer. Thereafter, the SCO-link can be re-established with the headset. Alternatively, the phone could renegotiate the link with the headset rather than with the computer, though in the either case this activity will be heard as a slight 'click' in the headset every time that it occurs.

The choice of which link to prioritize and which to renegotiate could actually differ from one scatternet implementation to another, but the important thing to remember here is that conflicts resulting from clock drift overlap are unavoidable. In a real user situation this type of pause-and-renegotiate activity is likely to occur fairly frequently, perhaps as often as two to three times per minute, which makes this a relatively challenging issue when attempting to pass scatternet communication operation to the host stack.

[1] Bluetooth Specification 1.2, Core System Package: Controller Volume, Baseband Specification, Chapter 8.6.6.1 Inter-Piconet Communications

4. Interoperability issues

The complexity of the user scenario introduced in the previous chapter stems in part from a primary issue with scatternet communication, i.e. that there are more than two master devices involved in the inter-piconet communication relationships. As described above, this means that there are two independent piconet clocks with different clock drifts, causing the piconets to overlap. Specifically, this problem becomes apparent because the host stack may know the sniff interval, but it does not know the sniff offset, and thus cannot know when to adjust for sniff window overlaps between the two master clocks.

The baseband, on the other hand, is fully aware of exactly when the two master clocks overlap. This is because all scatternet scheduling is handled below the HCI level by various components of the baseband, something that is discussed in more detail in the next chapter.

Possible work-around
In theory, there is a simple work-around to the problem of the host stack not having enough information to be able to perform scatternet operations. All that is required is to implement a number of manufacturer specific HCI commands that would enable the host stack to retrieve necessary timing information from the baseband controller. Indeed, it is this possibility that makes scatternet implementations via the host stack so tempting.

In practice, however, this is may prove to be more difficult than believed. Retrieving the necessary information is one matter, but now even the timing of the communication between the host stack and baseband comes into question. As we are talking about SCO links, the host stack scatternet implementation would have to be very fast. In fact, every time clock drift overlap occurs the host stack only has approximately 3-4 ms to receive the information over the HCI and to make the necessary adjustments. Anything longer than that would result in a much more noticeable disturbance in the voice link than just a slight click. To accomplish this, even the physical HCI transport between the host stack and the baseband would therefore also have to be exceptionally fast.

Interoperability breakdown
The greatest risk when passing the scatternet communication operation to the host stack, however, is the inherent breakdown of interoperability that this entails. Interoperability is, of course, one of the cornerstones of the Bluetooth Specification. and it is held in the highest regard by the Bluetooth SIG:

“The Bluetooth specification enables inter-operability between independent Bluetooth systems by defining the protocol messages exchanged between equivalent layers, and also interoperability between independent Bluetooth sub-systems by defining a common interface between Bluetooth controllers and Bluetooth hosts.” [2]

By design, the Bluetooth Specification permits the seamless use of a Bluetooth host stack from one manufacturer together with a Bluetooth baseband from another, without having to worry about interoperability. The baseband and the host stack are two of the sub-systems referred to in the Bluetooth Specification.

With the work-around describe above, interoperability between these two independent sub-systems is suddenly put at risk. Introducing special commands from the host stack towards the baseband would lockout the sub-system interoperability and prevent the use of independent sub-systems. In short, the baseband and host stack would need to be delivered together. This would be the only way to ensure that both sides use and understand the same manufacturer specific HCI commands needed to get important clock drift information up into the host stack so that a scatternet operation can be achieved.

Taking this somewhat easier road to scatternet via the higher levels may seem enticing on the surface, but it would in effect completely compromise sub-system interoperability by making the baseband and host stack dependent upon each other.

[2] Bluetooth Specification 1.2, Architecture & Terminology Overview, Chapter 2 Core System Architecture

5. Architectural considerations

Sub-system interoperability is unquestionably a major consideration when deciding upon an approach to a scatternet implementation, as is the overall intention of the Bluetooth Specification. With the latest release of the specification, Bluetooth 1.2, the Bluetooth SIG has made one of its most important contributions in the way of clarity. As a result of the Bluetooth SIG's efforts, the specification has now reached a very stable and clear status.

Among other improvements, the Bluetooth 1.2 Specification features an extremely well described architecture overview, with excellent core system architecture descriptions that explain the intentions and purposes of the various architectural blocks. The following excerpts from the specification illustrate these intentions as well as the clarity with which the Bluetooth SIG has addressed key elements of the core system architecture:

Device manager
"The device manager is the functional block in the baseband [and] is responsible for all operation of the Bluetooth system that is not directly related to data transport, such as inquiring for the presence of other nearby Bluetooth devices, connecting to other Bluetooth devices, or making the local Bluetooth device discoverable or connectable by other devices."[3]

Link Manager
"The link manager is responsible for the creation, modification and release of logical links, as well as the update of parameters related to physical links between devices. The LM protocol allows the creation of new logical links and logical transports between devices when required."[4]

Baseband Resource Manager
"The baseband resource manager is responsible for all access to the radio medium. It has two main functions: a scheduler that grants time on the physical channels and to negotiate access contacts. The access contact and scheduling function must take account of any behavior that requires use of the Bluetooth radio. In some cases the scheduling of a logical link results in changing to a different physical channel from the one that was previously used. This may be due to involvement in scatternet. When physical channels are not time slot aligned, then the resource manager also accounts for the realignment time between slots on the original physical channel and slots on the new physical channel."[5]

[3] Bluetooth Specification 1.2, Architecture & Terminology Overview, Chapter 2.1.3 Device Manager
[4] Bluetooth Specification 1.2, Architecture & Terminology Overview, Chapter 2.1.4 Link Manager
[5] Bluetooth Specification 1.2, Architecture & Terminology Overview, Chapter 2.1.5 Baseband Resource Manager

Clear intentions
As described above, the Device Manager of the baseband is responsible for inquiring after and connecting with other Bluetooth devices within the network. It is also responsible for handling multiple applications and must make decisions that directly affect the behavior of these applications. Therefore, in order to fulfill its tasks, it is imperative that the Device Manager is fully aware of the existence of a scatternet.

The Baseband Resource Manager, which makes sure that the radio communicates with only one entity at a time, together with the Link Manager, are both involved in various scheduling tasks between the different piconets. For this reason both of these must work very closely in unison.

Clearly the descriptions provided in the specification imply that scatternet is handled at the most fundamental level, with these blocks carrying much of the responsibility. In Chapter 3 we saw how the mobile phone in our user case needed to adjust for clock drift due to the overlap that can occur over time. Naturally, this requires an awareness of clock drift that is only available at the lower levels, in the Bluetooth baseband. And with the type of descriptions like those from the Architecture & Terminology Overview included above, the Bluetooth Specification 1.2 describes how functionality for controlling links is intended for the baseband, while the host software stack is better suited for controlling the flow of applications and their usage of the links. To move link control away from the baseband and up into the host stack would only serve to blur the intended division of responsibility.

6. Conclusion

As we have shown through our analysis, interoperability between independent Bluetooth sub-systems simply cannot be upheld when control of scatternet communication operation is handed over to the higher levels, i.e. the host stack. With such solutions, all Bluetooth entities immediately become dependent upon one another. This in turn defeats one of the primary goals of Bluetooth – to facilitate seamless, ad hoc wireless networks between Bluetooth devices originating from any manufacturer.

It is also clear that the authors of the Bluetooth 1.2 Specification are intent on making sure that developers focus on maintaining good link quality even in situations like the example user case above. Scenarios like these, where users expect multiple, concurrent applications not only to function but to function well, require close-knit cooperation at the most fundamental levels. This is evident through the clean-cut descriptions and explanations of the core system architecture formulated by the Bluetooth SIG.

The quality of the scatternet links is directly related to how well the scatternet implementation is able to deal with critical timing issues such as clock drift overlap. In this regard, the baseband, with its built-in timing and scheduling capabilities, holds a significant advantage over the host stack and thus provides higher link quality.

In light of these considerations, the only way to ensure that scatternet implementations remain true to the Bluetooth ideal is by implementing them below the HCI level, in the core Bluetooth control layers.