

Cobain Architectural Specification

Copyright (c) 2002-2003

Matti Dahlbom, `mdahlbom@cc.hut.fi`
Matti Kokkola, `matti.kokkola@iki.fi`

29th March 2004

Contents

1	Introduction	1
1.1	Symbian OS	1
2	Motivation	1
3	Cobain	1
3.1	Goals of the project	2
4	Architecture	2
4.1	Object-oriented design	2
4.2	Cobain and Symbian's client-server model	2
4.3	Underlying technology abstractions	3
4.3.1	Cobain Bluetooth implementation	4
5	Performance considerations	4
6	NanoIP	5
6.1	nanoUDP	5
6.2	nanoTCP	6
7	Ad-hoc routing	6
7.1	Packet routing	7

7.2	Joining a scatternet	7
7.3	Parting from a scatternet	8
7.4	Ad-hoc routing over the Internet	8
8	Conclusions and future work	8

1 Introduction

Cobain (originally *Collaborative Bluetooth Ad-hoc Integration Network*) is a lightweight ad-hoc networking framework (or “platform” in the sense that it offers a communications API and services) for Symbian OS environments. It has been designed performance and a small memory footprint in mind. The framework offers normal UNIX-like socket API, hiding the underlying low level data transportation layer. Currently, network interface implementation for Bluetooth exists, and IrDA interface is planned.

1.1 Symbian OS

Symbian OS is an operating system designed specifically for the demands of mobile devices. The most known Symbian based devices are the latest (by the time of this writing) mobile phones of Nokia, for example Nokia 7650/3650/9210i/6600/N-Gage and Siemens SX1.

2 Motivation

Development of Bluetooth aware Symbian applications is often unnecessarily difficult as the current API is way too complex for simple applications. Complexity of the API causes developers not to adapt Bluetooth features to their applications and thus availability of Bluetooth capable applications is not increasing as is expected.

Cobain aims to offer a solution that allows developers to develop Bluetooth aware applications without prior knowledge about Bluetooth protocol stack and its complexity.

3 Cobain

Cobain provides UDP-like unreliable datagram protocol. Unreliability is a known design decision: reliability features are left outside the scope of Cobain as all of the desired bearer layers provide reliability features. The Cobain platform consists of multiple layers; the lowest layer is basically just an efficient, easy-to-use networking API that hides the complexity of Bluetooth/IrDA/GPRS connectivity. Above that there is the conceptual nanoIP (see section 6) layer, providing addressing functionality for the packets. Above that there is the ad-hoc routing (see section 7) layer that adds routing capabilities between devices in different piconets. Additional layers may be inserted, given their design does not break down the functionality of the layer above them. Future plans include a multiplayer gaming API layer that would offer a basic framework for building a common client-server networking environment for gaming or similar use. The client software may reside above any of these layers.

For more technical design information concerning Cobain, see the Section 4.1.

3.1 Goals of the project

In the first phase of Cobain, the following deliverables will be ready:

- Bluetooth driver
- Proprietary datagram server
- An example application

All the source code will be written with Symbian C++.

4 Architecture

This section discusses the architectural design of the framework. While section 4.1 describes the object-oriented design of the code, section 4.2 explains how the system fits into the client-server model of the Symbian OS operating system.

4.1 Object-oriented design

One of the cornerstones of Cobain development has been ease of application development with it. For example, a minimal socket application wishing to connect to another device running a given service requires no more than a few lines of code and no prior knowledge about Bluetooth stack in order to perform the following steps:

- discover the devices available for Bluetooth connection
- discover the services available on the selected device
- connect to the given service
- send and receive arbitrary data to and from the remote service
- close the connection

Below is an UML diagram describing Cobain's class structure:

4.2 Cobain and Symbian's client-server model

Symbian OS is based on interprocess client-server relationships framework. In the framework, each server is responsible for handling a system resource and sharing it to the clients. In the basic configuration of Symbian OS, for example, access to the file system is done through the file server.

Each server is run in its own process in its own thread. The communication between client and server processes is done by passing simple messages in a synchronous or in an asynchronous way. The passed message itself can be data, or it can be a pointer pointing to the actual data block.

pared to the others and the fact that does not have existing support for TCP/UDP packaging like GPRS and is not limited to a single peer-to-peer connection like IrDA, it makes the most interesting subject for this research. Also as the concept of *scatternetting* is already well-known [1], no additional research effort is required to prove the possibility of building ad-hoc routing on the top of it, making it possible to directly focus on the design and optimization of such a system.

As protocols such as IrDA and Bluetooth support only peer-to-peer connections with no specific addressing mechanisms – the connection is merely established between two devices using their 48bit hardware Bluetooth addresses – some kind of an addressing mechanism has to be added in order to be able to do ad-hoc routing between devices and between piconets (TODO: cite). For this, *NanoIP* techniques are used. For discussion about *NanoIP*, see section 6.

4.3.1 Cobain Bluetooth implementation

The Bluetooth implementation of Cobain is built on the top of the existing Bluetooth transport layer protocols L2CAP and RFCOMM (see [2]). When an application built on top of Cobain wishes to communicate with a non-Cobain peer, either L2CAP or RFCOMM may be selected, depending on the implementation of the other peer. When two Cobain peers communicate, L2CAP should be selected as the one with better performance. With native Cobain applications (such as the built-in ad-hoc routing), L2CAP is always used. Figure 3 shows the Cobain protocol stack running on L2CAP.

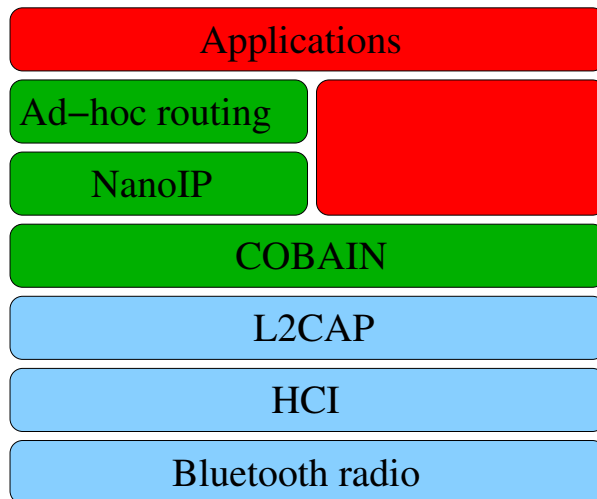


Figure 3: Cobain Bluetooth implementation on top of L2CAP. Bluetooth services offered by the host platform (Symbian OS) have a blue background, while the layers belonging to Cobain are on a green background. Services on top of Cobain are on a red background.

5 Performance considerations

One of the down-sides of the client-server framework is that all the communication between clients and servers is inter-process communicating (IPC) causing performance

penalty as it involves context switches and data transfers between independent address spaces.

To tackle this problem, related servers can be configured to run in the same process or to use shared memory for IPC.

6 NanoIP

Discussion in this section is mainly based on [3].

NanoIP is a protocol suite designed for the most minimal devices (usually embedded devices). In embedded environments, the use real-IP can be impossible as it takes too much bandwidth, power or memory.

In nanoIP, there is no separated network-layer nor network-layer addresses. Instead of them, 48-bit MAC addresses are used. As a consequence, nanoIP can only be used in subnetworks where all network nodes are accessible by MAC address. However, separate nanoIP networks can be attached together through a protocol bridge. Bridges can also be used to attach nanoIP network with a real-IP network. See Figure As no gateway or coordinating device is necessary, nanoIP is ideal for infrastructureless peer-to-peer communication.

nanoIP is called a protocol suite as it consists of two protocols, namely nanoTCP and nanoUDP, which are introduced in the following sections. Basically these protocols combine the most important features of network- and transport-layers.

6.1 nanoUDP

As its full-featured big brother, nanoUDP is a connectionless transport protocol. It provides basic encapsulation and an application multiplexing with ports. nanoUDP protocol frame can be seen in Figure 4.

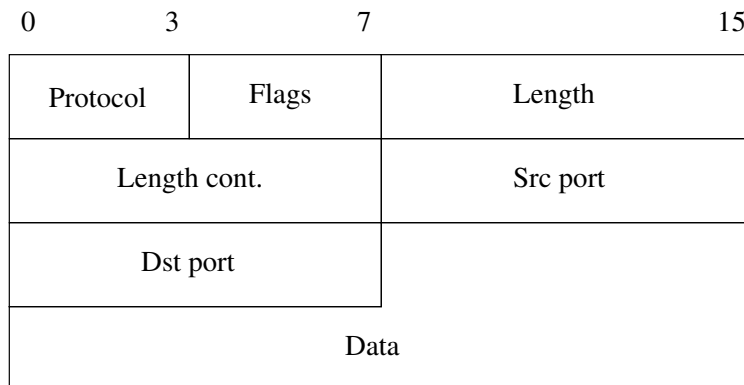


Figure 4: nanoUDP protocol frame

6.2 nanoTCP

nanoTCP is light-weight version of real-TCP, it provides connection oriented and reliable transport of datagrams. The protocol frame on nanoTCP can be seen in Figure 5.

0	3	7	15
Protocol	Flags	Length	
Length cont.		Src port	
Dst port		Sequence	
Sequence cont		Ack	
Ack cont		Data	
Data			

Figure 5: nanoTCP protocol frame

nanoTCP offers the same set of features as real-TCP, except the following features:

- smaller port number space
- no push functionality
- fixed window size, i.e. it does not support adaptive flow control
- no checksum
- round trip time-out calculation can be done in an implementation-specific manner

7 Ad-hoc routing

This section describes the planned ad-hoc packet routing. This approach requires that Bluetooth devices can simultaneously act as a master in one piconet and as a slave in another piconet. This kind of behavior is called *scatternetting*, and by the time of this writing is not supported by the Symbian OS devices on the market. For more on scatternets, see [1]. The routing system is based on the same idea that IP routing: every node taking part in the routing maintains a route list and an address list for every route. Unlike in IP routing, the address list may contain every destination address reachable via that route; this is possible due to the rather small number of hosts participating in the scatternet. In the future as the range of Bluetooth signal grows (and possible integration of ad-hoc routing over TCP/IP, see section 7.4) it may be required to change

the algorithm to prevent the destination address lists of growing too large. The construction of the scatternet requires a finetuned master selection algorithm as each of the routing tree nodes contains the complete address list of *all* of the reachable nodes in its subtrees.

7.1 Packet routing

When a host participating in a scatternet receives a packet, it performs routing for the packet as follows:

1. Checks whether it is the destination; if is, consume the packet
2. Checks whether the destination is included in the address lists of any of its children; if is, sends the packet to that child
3. If connected to a parent, sends the packet to the parent. If not, drops the packet.

The process of sending a packet is the same.

7.2 Joining a scatternet

Joining a scatternet is a rather strenuous operation as it involves introducing a new address list to the whole scatternet routing tree. When a host wishes to join a scatternet, the following steps need to be taken:

1. The point of connection in the scatternet must be decided. This means selecting a host already taking part in the scatternet to connect to. A selection algorithm should be used to pick a host with as large destination address list as possible; a large address list depicts a position near the root of the tree. Connecting to a node near the root of the tree makes the route shorter, thus offering better performance in the routing. This requires establishing connection to all of the nodes in the reach and requesting the child list, which may be a slow operation. If the connection to a host cannot be established (it already has the maximum number of Bluetooth connections in use), the host is ignored. If a connection is refused to all of the in-range devices in the scatternet, the connection to the scatternet itself is considered to be refused.
2. The connecting host creates a list of addresses containing all of the addresses in the destination address lists of its children and inserts its own address as the first entry in the list.
3. The connecting host sends the created address list to the host it connects to; its new parent.
4. The parent node creates a new routing table entry, assuming the destination address list supplied by the connecting host to the route.
5. The parent node then continues to step 2 in the role of the connecting node; this process is recursive and continues until a parentless node (tree's root) is met.

7.3 Parting from a scatternet

Parting from a scatternet is a straightforward task, although it is almost as time consuming than joining one; a node parting (disconnecting) from its parent does not have to do anything; its subnet remains as it was. The parent node whose child disconnects deletes the route to the child and the destination address list belonging to it. It then informs its own parent about the removal of a set of addresses. The parent then removes the list of given addresses from that route's destination address list, and informs its own parent in the same fashion. This, again, is a recursive process that is repeated until a parentless node (tree's root) is met.

7.4 Ad-hoc routing over the Internet

Networking over Bluetooth is – even with the *scatternet* support – very limited as the devices must be in close range with one other. IrDA on the other hand is limited to a single peer-to-peer connection, and GPRS is very slow. This is why it is desirable to have an alternative way of communication between Bluetooth piconets. One approach is to have a Bluetooth - internet - Bluetooth *bridge* to connect the piconets. This is accomplished by having stations capable of both Bluetooth and internet connectivity. Examples of such stations are PCs and special Bluetooth access points, running a programmable operating system and having an ethernet connection. The idea is to write software to such stations to handle both Bluetooth and internet connections and to interconnect these. Two stations share an internet connection and each shares a Bluetooth connection to a Bluetooth peer. The stations maintain a mapping between the Bluetooth connections and the internet connections. NanoIP or similar addressing technique is applied to the packet exchange by the end peers to address the Bluetooth network beyond the other station. This traffic passes through the bridge stations transparently.

8 Conclusions and future work

Limitations of close range information transmission technologies such as Bluetooth or IrDA make it impossible to flexibly build a communications system with an arbitrary number of peers. As the number connections per host is limited compared to the internet world, sharing data with many hosts at once is impossible, and manual connection handling is required.

This is what Cobain was designed to address; to offer a robust, scalable platform capable of constructing networks with unlimited number of participants on-the-fly and to handle packet traffic routing between the hosts in that network transparently to the user application.

While the *scatternetting* operations and packet routing require some additional memory in the form of maintaining the route lists, their functionality is efficient and does not take up much CPU resources.

At its simplest, Cobain acts as a simplified communications API; it saves the trouble of writing hundreds of lines of code required to discover the devices and their services, handle the connections and the low-level sending and receiving of the packets.

The basic functionality of Cobain has been completed and tested; the design and experimental implementation for the ad-hoc routing layer has been established. When the devices have the support for *scatternetting*, the functionality may be tested and improved.

Meanwhile, the focus of further development will be on the internet bridge, as it would be enormous asset in a multiplayer gaming environment, for example. It remains seen what kinds of other development interests Cobain will receive after its release under the GNU Public Licence.

References

- [1] Simon Baatz, Matthias Frank, Carmen Kühl, Peter Martini, and Christoph Scholz, “Adaptive scatternet support for bluetooth using sniff mode,” in *Proceedings of the 26th Annual Conference on Local Computer Networks*, November 2001, pp. 112–120.
- [2] Bluetooth Special Interest Group, “Bluetooth standard,” .
- [3] Shelby Z. D., Riihijärvi J., Raivio O., and Mähönen O., “NanoIP,” Internet-draft draft-shelby-nanoIP-00, IETF, May 2003.