FFI RAPPORT

A STUDY OF COMPUTER INTERFACES FOR SOLDIER SYSTEMS

(A look at Ethernet, FireWire and USB)

OLSEN Lars Erik, FLATHAGEN Joakim

FFI/RAPPORT-2005/03043

A STUDY OF COMPUTER INTERFACES FOR SOLDIER SYSTEMS
(A look at Ethernet, FireWire and USB)

OLSEN Lars Erik, FLATHAGEN Joakim

FFI/RAPPORT-2005/03043

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
P O Box 25, NO-2027 Kjeller, Norway

FORSVARETS FORSKNINGSINSTITUTT(FFI)
Norwegian Defence Research Establishment

P O BOX 25
NO-2027 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT-05/03043	UNCLASSIFIED	43
1a) PROJECT REFERENCE	**2a) DECLASSIFICATION/DOWNGRADING SCHEDULE**	
FFI-V/878/	-	

4) TITLE

A STUDY OF COMPUTER INTERFACES FOR SOLDIER SYSTEMS
(A look at Ethernet, FireWire and USB)

5) NAMES OF AUTHOR(S) IN FULL (surname first)

OLSEN Lars Erik, FLATHAGEN Joakim

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH	IN NORWEGIAN
a) Ethernet	a) Ethernet
b) FireWire	b) FireWire
c) USB	c) USB
d) Zero Configuration	d) Zero Configuration
e) Soldier System	e) Soldatsystem

THESAURUS REFERENCE: Tesaurus

8) ABSTRACT

This report looks at the challenges regarding computer communication and information sharing between the different NATO counteries soldier systems. The emphasis is on the wired interface for this data exchange. The report looks into the most common computer interface technologies (Ethernet, FireWire and USB) and discusses the usefullness of these technologies as wired communication interface between different countries sodier systems.

9) DATE	AUTHORIZED BY	POSITION
2005-10-10	This page only (Jan Ivar Botnan)	Director

ISBN 82-464-0967-0

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

CONTENTS

page

A STUDY OF COMPUTER INTERFACES FOR SOLDIER SYSTEMS

(A look at Ethernet, FireWire and USB)

1 INTRODUCTION

In the last decade we have seen an incredible development in all sectors of computer and information technology. Computers have become common property and information access and sharing on the internet is an everyday doing for many people. This has led to the introduction of personal computers and computer assistance in many new arenas, and the military is no exception. Todays soldiers equipped with modern weapons are very powerful and they are given greater missions and responsibilities in combat. For the soldier to be able to handle all these responsibilities, he needs the best possible situational awareness, and to help, many nations are looking into, or have already started to equip the soldiers with computers to improve the situational awareness and hence the effectiveness of the soldier.

Most of the conflicts involving NATO are resolved with a coalition of forces from many NATO countries. Therefore it is important that information and data can be shared between the different countries soldier computer systems within NATO. To ensure this, NATO has created TG/1 (soon to be LG/1) who are working on interoperability issues between the NATO nations future soldier systems. This reports is a study of the three most relevant interfaces for wired information and data sharing between the different nations soldier computer systems, namely Ethernet, FireWire and USB.

2 COMMUNICATION AND INFORMATION SHARING

Communication is “the process of exchanging information usually via a common system of symbols”[1]. For soldier systems, this can be narrowed down to “exchanging information via a common communication interface and protocol”. Its important to acknowledge the fact that both the communication interface and protocol must be agreed upon before any communication can take place. For instance, we could all agree on talking to each other on the phone, but if we didn’t agree on which language to speak in, the communication would fail.

We can divide the data communication needs for soldier systems in two parts after the size and nature of the data being communicated. Small data payloads could be information like GPS positions, text messages, target information etc. Since these data are small, (<1Kb), They can be communicated often and therefore would benefit from/require a long range wireless communication channel. Larger data payloads, like high resolution images, video files or map

[1] from Merriam-Webster Online Dictionary

data, are not easily communicated over a wireless channel because the communication will take up much of the bandwidth available in the wireless network and hence block all other communication. Wireless communication also have a trade off between range and bandwidth making long range Wireless communication considerably slower than wired communication. Large data payloads will therefor require a wired communication channel or a short range, high capacity wireless channel. This report will look at wired communication interfaces for large payload data communication. However, some of the configuration and communication protocols described in this report also apply to wireless communication devices.

There exists numerous interface standards that could be used in soldier systems, but this report will only look at the three most common which also are likely to be found in soldier systems or could easily be added to soldier system. These interfaces are Ethernet, FireWire and USB.

3 INTERFACES

The purpose of the report is to investigate different solutions for wired data communication between soldier systems.

In chapters 4 to 6 the different interfaces are presented together with their advantages and disadvantages and possible solutions for using the interface for soldier system communication. A more technical description of the mechanisms in the different interfaces are given in the appendixes.

An optimal solution for inter soldier system communication would be a solution where two ore more soldiers from different countries can plug a cable into a socket or hub and all data interchange is done automatically or by request from the soldiers. For this scenario to work, several mechanisms must be in place:

- The soldier computer systems must have the same link interface with compatible connectors.
- The link interface must have some sort of auto detection mechanism that can detect that it has been connected to something.
- The link interface must automatically be able to set up a communication using a common protocol.
- There must exist a mechanism to automatically detect who have connected to whom.
- There must exist a mechanism to detect what kind of data one soldier system can share, and the soldier system must also know what it wants to receive (the last could be done by human interaction)

Ethernet, FireWire and USB will be considered with regard to these mechanisms in the following chapters. It is not within the scope of this report to consider properties like power consumption and EMC (Electromagnetic Compatibility).

4 ETHERNET

The original Ethernet was developed as an experimental coaxial cable network in the 1970s by Xerox Corporation using a carrier sense multiple access collision detect (CSMA/CD) protocol. That project led to the 10Mbps Ethernet Version 1.0 specification by Digital Equipment Corporation (DEC), Intel Corporation, and Xerox Corporation. This work served as the basis for the IEEE 802.3 specification published as an official standard in 1985 (ANSI/IEEE Std. 802.3-1985). The standard specifies the physical and lower software layers. Since then, a number of supplements to the standard have been defined to take advantage of improvements in the technologies and to support additional network media and higher data rate capabilities, plus several new optional network access control features. Four data rates are currently defined for operation over optical fiber and twisted-pair cables:

- *10 Mbps* Ethernet
- *100 Mbps* Fast Ethernet (802.3u)
- *1000 Mbps* Gigabit Ethernet (802.3z and 802.3ab)
- *10Gbps* (802.3ae) - The latest addition to the standard. Formally ratified by the IEEE on 12 June, 2002.
- *802.3af* - Power over Ethernet is an extension to the standard, which transmits both power and data using the twisted pair cable.

Since the introduction, Ethernet has continuously gained in popularity, and it is currently used for more than 85 percent of the world's LAN-connected PCs and workstations[2]. There are a number of factors that have helped Ethernet to become so popular:

- Is easy to understand, implement, manage, and maintain
- Allows low-cost network implementations
- Provides extensive topological flexibility for network installation
- Guarantees successful interconnection and operation of standards-compliant products, regardless of manufacturer

4.1 Technology

Ethernet will, of course, not be covered in detail in this report. However, some background information about the technical aspects and building blocks in the Ethernet system are

[2]According to industry analyst International Data Corporation (IDC), Framingham, MA.

necessary in order to address the challenge of soldier-to-soldier interconnection. More details about Ethernet can be found in literature such as (13). For readers not familiar with the OSI model a short introduction to the seven layered concept is mentioned in Appendix A on page 28.

The Ethernet system includes four components that, when combined, make a working Ethernet:

- The *physical medium* is the signal-carrying portion of the shared Ethernet channel. This include fiber optics, twisted pair and coax.
- The *media access control protocol* uses the CSMA/CD protocol to allow multiple computers share the same Ethernet channel in a fair manner.
- The *signaling components* is the standardized electronic devices including Ethernet interfaces, repeaters, hubs etc.
- The *frame* which consists of certain fields including Destination address, Source Address, payload Data and more.

As mentioned above, the *frame* consists of source and destination address. The first part of the 48-bit address is a 24-bit Organizationally Unique Identifier unique for each hardware manufacturer. The remaining 24 bits is assigned by the vendor, being careful to ensure that each address is unique. The resulting 48-bit address is often called the hardware address, physical address or Media Access Control (MAC) address. This unique address avoids the problem of two or more Ethernet interfaces in a network having the same address. Since the address is specified during production, this eliminates the need to locally administer and manage Ethernet addresses.

The destination address may also identify a broadcast packet. If all of the 48 bits in the destination field is “1”, this will tell every recipient on the Ethernet to read the packet. This broadcast mechanism is used in the ARP protocol (among others).

The payload data in the Ethernet frame can carry different layer 3 protocols (referred to OSI model in Appendix A). These protocols can be one of NetBEUI, IPX, X25 or several others, but the most widely used protocol is the Internet Protocol (IP) as defined in (12).

Address resolution protocol

While Ethernet uses MAC to address the packets, IP uses the IP addresses. Fundamentally the layers are totally unaware of each other. Hence, the higher level protocols and the Ethernet system must interact in order to resolve the correct destination address for the Ethernet frame. The TCP/IP system accomplishes this task using the Address Resolution Protocol (ARP). The ARP protocol provides the glue, so to speak, between the Network layer and the Data link layer.

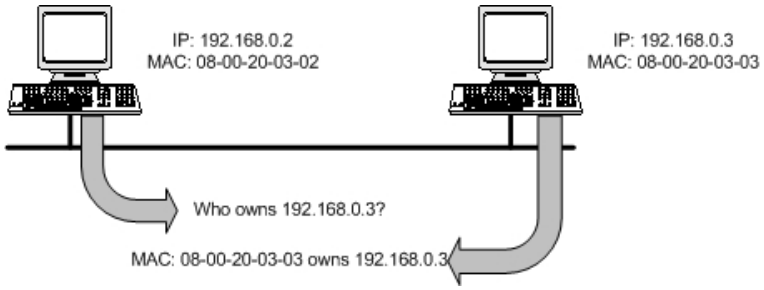

Figure 4.1 Using ARP over an Ethernet

Figure 4.1 shows two stations sending and receiving ARP packets over an Ethernet. Station A wants to send data to station B, which has the IP address 192.168.0.3. The ARP protocol sends a broadcast packet on the Ethernet to resolve station B's MAC address. Of course, Station B replies, and Station A is ready to send data as IP packets encapsulated in Ethernet frames. There are actually four types of ARP messages that may be sent by the ARP protocol. The types of message are: ARP request and ARP reply - as shown in Figure 4.1 and the reverse mechanism RARP request and RARP reply. The ARP protocol is described in detail in (11).

4.2 Autoconfiguration of the Ethernet network

When connecting two or more network-nodes (computers, soldiers) there is a need for some sort of configuration of the addresses to be able to interchange data between them. In a non-persistent ad-hoc network there is a matter of necessity that this configuration is done automatically. The configuration has to be done in several layers in the network implementation. Using the TCP/IP model (as described in Appendix A), one can show that connectivity must be done in the following layers:

- *Physical layer* - The host address at the physical interface is vendor specified as mentioned in 4. There is no need for any extra configuration. However, the ARP protocol is used to map the physical MAC addresses to the network addresses.
- *Network layer* - The IP address must be configured in some matter. Either manually or automatically. 4.3.
- *Application layer* - Even if the nodes got suitable addresses, it must be a way to tell the application that the nodes are successfully connected and ready to interchange information. Hence, an application layer protocol is needed.

We now know that the Ethernet address is fixed by the hardware vendor, and ARP is a simple solution to associate Ethernet address to the higher level IP address. But how is the IP address chosen? In IP networks, there exist a few ways to configure the address:

- *Manually* - To configure the address manually is always an option, but requires some

knowledge about IP-networks. In a soldier network, the actual address is of no interest for the user, and all the configuration issues should be hidden for him/her.

- *DHCP*[3] - Its purpose is to enable individual computers on an IP network to extract their configurations from a server (7). The most significant piece of information distributed in this manner is the IP address. The challenge in a soldier-to-soldier network is the fact that no central server can be delegated to distribute IP addresses. Hence, DHCP is not a suitable solution for a soldier-to-soldier network, but could be used if a central server can be placed in a building, vehicle etc.

- *ZeroConf* - Zero Configuration Networking is a set of techniques that automatically create a usable IP network without configuration or special servers.

In an Ad-Hoc network with no central server, ZeroConf is probably the best technique to administer the network. The next chapter gives an introduction to the ZeroConf concept and how to use it to configure both the network layer and the application layer.

4.3 Zero Configuration Networking

Even if DHCP is widely used in many Intranets, using it in a soldier-to-solder network gives a complicated and awkward solution. The best way is to let the connected network nodes configure itselves with no need for a central server or a server software. This is currently possible using protocols such as Appletalk or NETBIOS, but the concept behind ZeroConf or Zero Configuration Networking is to create a standardized, inter-operable way to make easy networking available for any host, independent of operating system and across platforms. ZeroConf allows unknowledgeable users to connect computers and all sorts of IP devices together and expect them to work. A typical usage scenario would involve two laptops (or in our case - two soldiers) connected with a cross-over Ethernet cable or ad-hoc wireless network.

However, keep in mind that it is not possible to completely automate the configuration of the Internet. Zero configuration protocols allow local communication on networks of limited scale. Some implementations of ZeroConf already exists, but the ZeroConf concept is still in the initial phase.

Three functions will benefit from ZeroConf :

- Network layer *address assignment*.

- *Name-to-address resolution* without a DNS server.

- *Service discovery*. Find services like printers, FTP, HTTP in the network.

[3] Dynamic Host Configuration Protocol

Address assignment

The address assignment routine creates a so called link-local address. The address space is in the range of 169.254.1.0 to 169.254.254.255 with a total of 65024 available addresses. The address is chosen randomly with the hardware address (MAC address) as seed to the random generator. Then, the host does an ARP probe for the address, and if the address is already taken by an other host, a new address is chosen randomly, and tries the ARP probe again (4).

Name-to-address resolution

In IP networks, nodes are typically identified using names, rather than IP addresses. With no use of a central DNS server, the ZeroConf name-to-address translation protocol must provide mechanisms for:

- obtaining the IP address for a certain name
- determine the name belonging to an IP address

To resolve an IP address from host name, the computer sends a request to a fixed multicast address and waits for a reply. Each host on the network will receive these requests, but only the host with the given name will respond.

There are some proposed implementations that works pretty much the same way, but they are not compatible. Apple Computer's Multicast DNS[4] (mDNS) (6) and Microsoft's Link-local Multicast Name Resolution (LLMNR) (1) is two of them.

Service discovery

The ZeroConf concept also includes a third protocol area of huge interest in a soldier-to-soldier network. *Service discovery* allows clients to be able to discover services on the network with no prior configuration. This include services such as scanners, backups, mail servers, web services, file servers, messaging etc. In normal IP networks a client will need the name or IP address of the service and the TCP/UDP port in order to use it, but the idea behind ZeroConf is to automatically resolve this address issue. For our purpose, this means that a soldier application has the ability to automatically detect and discover certain services on an other connected soldier computer, such as file sharing.

A few different protocols exists for service discovery over IP networks. The three most important is *Service Location Protocol (SLP)*, *DNS Service Discovery (DNS-SD)* and *Simple Service Discover Protocol (SSDP)*.

Service Location Protocol is described in (9) and is an IETF standard.

[4]The term mDNS is also used by other suppliers

Simple Service Discovery Protocol (SSDP) is an UPnP [5] protocol, used in Windows XP and several brands of network equipment (8). SSDP uses HTTP to all the transmissions.

DNS Service Discovery on the other hand is a way of using the existing DNS records to locate services as described in (5). Since a ZeroConf implementation must likely have a multicast DNS responder for the name-to-address translation, this design can be implemented in quite a lightweight manner. For that reason, DNS-SD is considered simpler than SSDP because it uses DNS rather than HTTP. The protocol can be used to obtain names, service type, port numbers and other attribute information.

Implementations

Both Microsoft and Apple are working with ZeroConf solutions, and both have supported link-local addresses since 1998 which conforms to the RFC 3927 standard (4). But unfortunately Microsoft and Apple support different - and non compatible protocols for address resolution and service discovery. The two main systems are:

- UPnP - Universial Plug And Play - supported by Microsoft
- Bonjour - Created by Apple

UPnP is built around the Simple Service Discovery Protocol which uses HTTP and XML. The UPnP Forum consists of almost 800 industry companies. Apple is not a part of them, hence it's very unlikely that they will include UPnP in a future version of the MAC operating system. Some brands of network devices has included the UPnP protocols, which gives a zero configuration solution if used in a network with MS Windows computers.

Apples *Bonjour* (formerly Rendevouz) is included in MAC OS X and is used by Apple software such as iPhoto and iTunes. The system uses a combination of link-local address choosing, mDNS, and DNS-SD, altogether a plain and simple approach to ZeroConf. Since Apple first launched Bonjour in 2002, every major maker of network printers has adopted Bonjour. Apple is working hard to get acceptance in the marked, and recently released an implementation for Windows, free to download. The Bonjour SDK is available with platform specific code for Mac OS X, Windows, Windows CE, Linux, Solaris, FreeBSD and VxWorks.

Besides from UPnP and Bonjour, there is also several other implementations out there, listed in appendix B.

The Zeroconf technology is embraced by the market, and both Bonjour and UPnP are out ready for use. Based on either UPnP or Bonjour, you can now easily create a pair of applications for auto configurable service sharing. But true out-of-the-box zero configuration between computers with different operating systems is not yet possible. We still have to wait for whatever becomes the de-facto zeroconf standard.

[5] The UPnP Forum is an industry initiative designed to enable simple and robust connectivity among stand-alone devices and PCs from many different vendors.

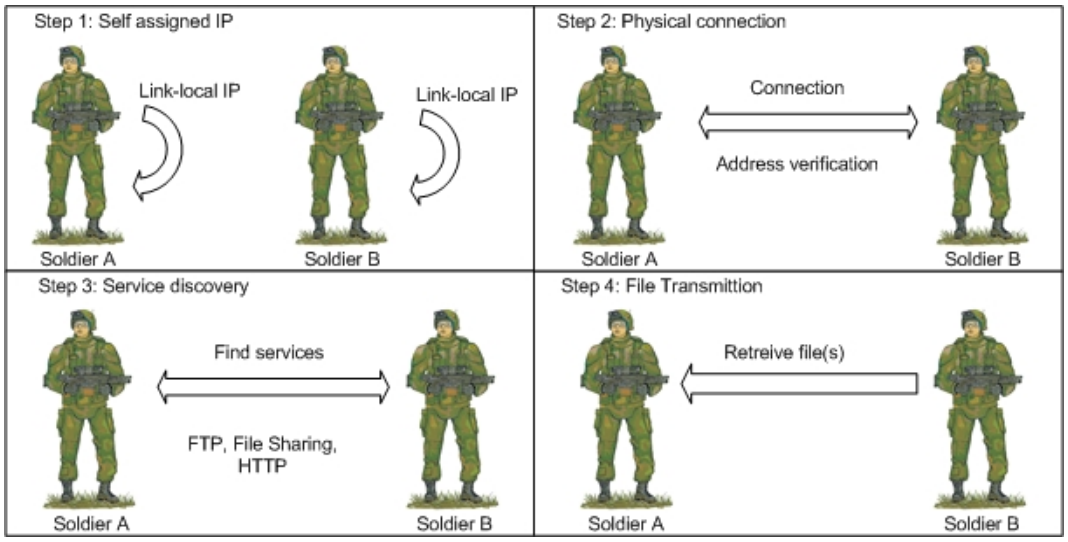

Figure 4.2 The steps using ZeroConf in a soldier-to-soldier file exchange system

4.4 Ethernet as soldier system communication interface

Lets sum up and figure out how Ethernets fits as a communication interface between soldiers systems. Because of its popularity, the Ethernet hardware is relatively cheap, and many processor boards supports Ethernet out-of-the-box. Ethernet is among the fastest wired standards (up to 10Gbps), making it the first choice when hight bandwidth is required. Ethernet will probably be supported and extended in years to come. Ethernet gives an element of flexibility and scalability that you cannot find in any other interconnection standard and together with power over Ethernet (IEEE 802.3af) you can supply power to peripheral Ethernet devices. However, there is a configuration issue to overcome. Ethernet is not a complete system and you need several layers of protocols on top of it (usually the TCP/IP family) to make a usable communication system.

By using TCP/IP and ZeroConf technologies, one can make an auto configurable system on top of Ethernet. This means that one soldier can connect to an other soldier using a crossover Ethernet cable, and the systems will automatically negotiate and initiate communication.

One could imagine a system based on the following standards:

- Ethernet *802.3*
- *IP* on the network layer
- *TCP* and *UDP* as transmission protocols
- *FTP* as application layer protocol for file transfer
- *ZeroConf* Technology for service discovery etc.

These standards combined gives a system as shown on Figure 4.2. The usage of the ZeroConf implementation is as follows. Step 1 includes the self assigned IP chosen from the link-local address range. Step 2 shows the physical connection between the two soldiers. The network is now running and prepared for Service Discovery - Step 3. When an appropriate service is found in the network, one soldier computer can utilize the service, for example file transmission

Please keep in mind that the ZeroConf concept does not require the use of Ethernet as communication carrier. You might as well use 802.11 (WLAN), FireWire, USB, radio modem, or another standard supporting the TCP/IP protocol family.

5 FIREWIRE

FireWire was developed by Apple in the mid 80's as a high-speed method of transferring data to and from the hard drives inside Macintosh desktops while simplifying the internal cabling. The technology was adopted by IEEE and this resulted in the IEEE 1394-1995 standard in 1995. FireWire is also known as IEEE 1394 and iLink but this report will use the term FireWire. The system is commonly used for connection of data storage devices and digital video cameras, but is also popular in industrial systems for machine vision and professional audio systems. It is used instead of the more common USB due to its faster speed, higher power distribution capabilities, and because it does not need a computer host. It also has native support for isochronous[6] data transport (data that must be delivered with deterministic latency, such as audio or video).

Figure 5.1 FireWire logo

5.1 Technology

FireWire is a serial bus, meaning that only one data bit is transmitted at a time.

FireWire can connect together up to 63 peripherals in an acyclic network structure. It allows peer-to-peer device communication, such as direct communication between a scanner and a printer without using a computers system memory or CPU. FireWire also supports multiple hosts per bus, and IP networks, ref chapter 4, can be formed through software between FireWire-linked computers. It is designed to support plug-and-play and hot swapping. Its six-wire cable can provide 8-40 VDC and up to 1.5 A (system dependent), allowing devices to operate without a separate power cord.

FireWire 400 can transfer data between devices at 100, 200, or 400 Mbit/s data rates (actually 98.304, 196.608, or 393.216 Mbit/s, but commonly referred to as S100, S200, and S400). Cable length is limited to 4.5 meters but up to 16 cables can be daisy chained yielding a total length of 72 meters under the specification.

FireWire 800 (Apple's name for the 9-pin "S800 bilingual" version of the IEEE1394b standard) was introduced by Apple in 2003. It allows an increase to 786.432 Mbit/s with backwards compatibility to the slower rates and 6-pin connectors of FireWire 400.

The IEEE 1394b specification supports optical connections up to 100 meters in length and data rates up to 3.2 Gbit/s. The original 1394 and 1394a standards used data/strobe (D/S) encoding, see appendix C.1, (called legacy mode) on the signal wires, while 1394b adds a data encoding scheme called 8B10B (also called beta mode). With this technology, FireWire, which was arguably already slightly faster, is now substantially faster than Hi-Speed USB.

[6] Iso (same) chronous (time):Uniform in time. Having equal duration. Recurring at regular intervals

A FireWire devices is identified by an IEEE EUI-64 unique identifier (this is an extension of the 48-bit MAC address used in Ethernet). In addition to this, it uses codes to indicate the type of device it is and the protocols it supports.

For a more technical description of FireWire, see appendix C on page 31

5.2 Security Issue

Devices on a FireWire bus can communicate by direct memory access, where a device can use hardware to map internal memory to FireWire's "Physical Memory Space". The SBP (serial bus protocol) used by FireWire disk drives use this capability to minimize interrupts and buffer copies. In SBP, the initiator (controlling device) sends a request by remotely writing a command into a specified area of the target's FireWire address space. This command usually includes buffer addresses in the initiator's FireWire "Physical Address Space", which the target is supposed to use for moving I/O data to and from the initiator.

According to (17), for many implementations (particularly those like PCs and Mackintoshes who are using the popular OHCI [7] interface) the mapping between the FireWire "Physical Memory Space" and device physical memory is done in hardware, without operating-system intervention. While this enables extremely high-speed and low-latency communication between data sources and sinks without unnecessary copying (such as between a video camera and a software video recording application, or between a disk drive and the application buffers), this can also be a security risk if untrustworthy devices are attached to the bus. For example, a malicious FireWire device can get direct access to host memory, bypassing operating system limitations, and read and modify sensitive memory, causing privilege escalation, information leakage and system compromise. For this reason, high-security installations will typically either purchase machines that map a virtual memory space to the FireWire "Physical Memory Space" (such as a G5 Macintosh, or any Sun workstation), disable the OHCI hardware mapping between FireWire and device memory, physically disable the entire FireWire interface, or do not have FireWire at all.

5.3 FireWire as soldier system communication interface

Lets investigate how good FireWire fits as a communication interface for soldier computer system. First of all, FireWire has plug and play capability so it will automatically be detected by an operating system when connected. It also has the performance needed to transfer large amount of data in a short time (400 Mbit/s), and it allows for host-to-host communication. The only thing that needs to be added is a communication protocol. This could be a completely proprietary communication protocol or based on a collection of other standard protocols. A common way to use FireWire for computer to computer communication is to use a virtual

[7] Open Host Controller Interface, OHCI, is an open standard that allows a computer host to interact with FireWire and USB devices.

network driver that treats the FireWire connection as a standard network interface. This allows all standard networking protocols to be used over the FireWire connection. For soldier systems, this could be a good solution, and the network configuration could be done either by the ZeroConf and service discovery mechanisms described in chapter 4, or by other standard network protocols.

It is also a possibility to develop proprietary FireWire communication protocols that handles all the identifications and communication.

FireWire may look as the ideal interface for soldier system communication but there are a couple of concerns. First of all, the security issues mentioned in chapter 5.2. Secondly, FireWire is not as common as USB and Ethernet. This means that there are fewer peripherals available with a FireWire connection and hence fewer systems that has a FireWire host controller. Equipping system with FireWire can prove difficult and expensive.

6 USB

Universal Serial Bus (USB) provides a serial bus standard for connecting devices, usually to a computer, but it also is in use on other systems such as game consoles and PDAs.

Figure 6.1 A Collection of different USB logos

USB was invented as a easy to use, fast and reliable and cheap interface to replace the many different PC interfaces like different COM ports and parallel ports. To achieve all this the USB standard was designed completely from scratch. It was first introduced as USB 1.0 in 1996. It has since then seen two updates:

- USB 1.0 was introduced in 1996 and supports data rates of 1.5 Mbit/s and 12 Mbit/s
- USB 1.1 was introduced in 1998. It was written to provide further clarification and additional features for USB devices and hubs. It supports the same data rates as 1.0, and are backwards compatible with version 1.0.
- USB 2.0 was introduced in 2000. This version introduces a data rate of 480 Mbit/s and makes USB match the bandwidth of FireWire400. USB 2.0 are backwards compatible with USB 1.1

The USB specifications are maintained by the USB Implementers Forum (USB IF).

USB has a host-device structure. A device connects to USB host through a chain of hubs. There always exists one hub known as the root hub, which is attached directly to the host controller (usually a PC). A Hub works like a bridge. It has a single upstream connection (that is going to the root hub, or the next hub closer to the root), and one to many downstream connections.

Hubs are themselves USB devices, and may incorporate some amount of intelligence. Once a hub detects a new device (or the removal of one), it reports the new information to the host, and enables communications with the newly inserted device. Though physically configured as a tiered star, logically (to the application code) a direct connection exists between the host and each device. A total of 127 USB devices can be connected to a host controller.

Fig USB tired star network.

In a USB network of hubs and devices, all communication is controlled by the host. No device can send data without the host requesting data. This means that the USB devices can only send

data to the host and the host can only send data to one device at a time. Devices cannot speak to each other and they are also unaware of each other. This is a major difference between USB and FireWire where FireWire allows for direct peripheral-to-peripheral communication. However, since the host controls all network traffic, the USB device hardware can be made much simpler and cheaper.

USB uses a 4 wired cable, one wire pair is used for data communication, and one pair for power. A cable can be maximum 5 meters long. Any longer distances require hubs. A maximum of four hubs can be chained yielding a total length of 20 meters. The bus voltage is nominally +5V, however, the specification (15) allows for as low as 4.4V. A device that draws up to 100mA can extract all of its power from the bus all the time. If the device requires less than 500mA, and if the upstream host or hub can provide that much power (which is optional), the device can be bus-powered if at power-up time, during system configuration, it consumes less than 100ma. If the device needs more than 500mA, it must have its own power supply.

The USB host and hubs can manage power by enabling and disabling power to individual devices by electrically remove peripherals from the system. Further, they can instruct devices to enter a suspended state, which reduces maximum power consumption to 500 micro amps (for low-power, 1.5Mbps peripherals) or 2.5ma for 12Mbps devices.

When a device is attached to the USB network, the host communicates with the device to learn its identity and to discover which device driver is required. This process is called *enumeration*.

During enumeration, the host requests a number of descriptors from the device. Based on these descriptors, the host controllers operating system loads the correct driver for the device. For more information on the enumeration process, see appendix D. If the operating system cant find any suitable device driver, it needs user interaction to tell it what driver to use. The next time the USB device is plugged in, it will in most cases use this driver. In some cases however, for instance if the USB device is plugged into another USB port than last used, the operating system may need to ask for the device driver again. This has to do with the way the USB device identify itself to the USB host. If the USB device has a serial number (which is optional for most types of USB devices) the operating system will recognise the device on whatever port, but if the device doesn't have a serial number, it may ask for the device driver the first time for each port.

Many USB devices share a set of common functionality. For example, All mice send information about mouse movement and button clicks, and all disk drives transfer files between the host and the device. This has led the USB Implementers Forum (USB IF) to define a set of classes that share common functionality so that a common device drivers can be implemented. Windows XP and many flavors of Linux comes with standard drivers that work with the most common USB devices like keyboards, mouse, storage units like USB pens and USB hard disks. However, special USB devices require their own drivers. For a list of the device classes specified by USB IF, see (14).

6.1 USB On The Go

Due to its widespread acceptance, USB is becoming the de facto industry standard for connecting peripherals to PC's and laptops. Many of the new devices now using USB are also portable devices.

As these portable devices increase in popularity, there is a growing need for them to communicate directly with each other when a PC is not available. The On-The-Go Supplement addresses this need for mobile interconnectivity by allowing a USB peripheral to have the following enhancements:

- Limited host capability to communicate with selected other USB peripherals
- A small USB connector to fit the mobile form factor
- Low power features to preserve battery life

For more information, see (16).

6.2 USB as soldier system communication interface

Lets investigate how good USB fits as a communication interface for soldier computer system. USB is a very common interface, so its very likely that all soldier systems will have this interface. USB is also plug-and-play compatible so it will automatically be detected when connected. However, as described earlier, USB has a host-device type architecture. This means that USB can only be used as a communication interface between a host computer and a USB device. This makes the interface unusable to use for host-to-host communication as one would expect the soldier system communication to be. If we abandon this idea of direct host-to-host connection, there are a number of ways to make USB work.

6.2.1 USB device to host

One could imagine that one soldier system could behave like a USB device and and use another soldier system as the host. The USB device must identify itself as something to the USB host. This could be as a proprietary military USB device that required proprietary drivers, or it could be as something as simple as a portable storage unit (like a USB pen). The Host is responsible for all communication, so all data must be either pushed or pulled by the host. If the USB device would act as a storage unit, all data could simply be copied as files between the systems. If the USB device would identify itself having a more proprietary format, the data would be sent as byte streams between applications on the soldier systems. Even though USB will take care of transmission of the bytes or the files, there must exist a common interpretation

of these data. This interpretation could for instance be a part of the data exchange API created by the NATO TG/1 C4I group. A drawback for this solution is that all soldier systems must have one USB port working as a USB device or the ability to make a port switch from host to device. In addition, the soldiers who are exchanging data must agree on who should be the device and who should be the host before initiating the communication, and possibly also choose a cable accordingly.

6.2.2 USB standalone device

A very simple solution would be to use a stand alone USB device like a USB pen drive. First plug it into one system, put all your data into it. Pass it to the next system. This could have some security issues if/when the USB pen drive was lost. There also exists USB cables with two integrated devices in them. These devices has a shared memory area so that they can be connected to each their host and share data between the two, see figure 6.2. These cables are common “of the shelve products” and can be used in stead of a Ethernet network connection between two computers. They require proprietary drivers. If they are used as “standard” network connections, one could use the mechanisms described in chapter 4 for data exchange. It is also be possible to develop drivers that would allow for direct application to application communication through the USB devices.

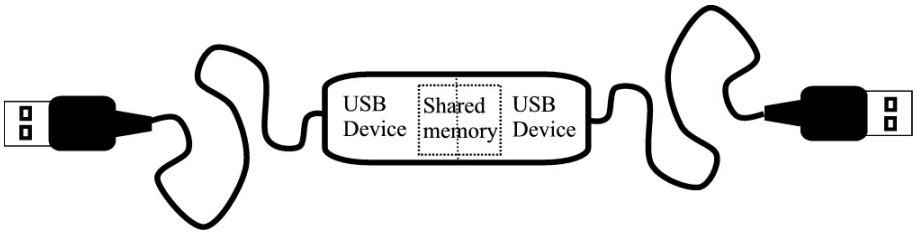

Figure 6.2 A cable that can connect to two hosts and enable them to communicate

6.2.3 USB to Ethernet

Soldier system with no available Ethernet connection could acquire a USB to Ethernet adapter and use the mechanisms described in chapter 4 for data exchange.

6.2.4 USB On-The-Go

USB On-The-Go was designed so that USB devices could act as hosts for certain types of other USB devices. This doesn’t fit well with the idea that we want two host to communicate. If a soldier system didn’t have any USB host ports at all but only a USB device port, it would make sense to make this port On-The-Go enabled for certain other USB device equipment, but

if there exist a USB host on the system, On-The-Go functionality will only be a subset of the USB host's functionality and hence, of no use.

6.3 Summing up USB

USB has many of the features we want for soldier system communication: It can be connected and disconnected without rebooting, it has plug-and-play support meaning that it will be detected by the operating system and it has high bandwidth (480 Mbit/s for USB 2.0). However, the host-device architecture makes it impossible to use as a "out-of-the-box" communication solution for host-to-host communication. Here follows an evaluation of the different solutions given in chapter 6.2:

- **USB device to host**. This solution is probably not very realistic since it depends the two communicating systems to behave differently. It requires all soldier systems to be able to be both a USB device or a USB host in the data communication. If the USB device would identify itself as an USB storage unit (like a USB pen drive) it could still be a quite elegant solution. In either case, the USB host needs a suitable device driver for the USB device.
- **USB standalone device** Both the USB pen-dive solution and the "host-to-host" bridge cable are realistic solutions. USB 2.0 has a high bandwidth (480Mbit/s) and a lot of data could be communicated in a short time. By copying all data as files on a pen-drive, we relieve ourselves from many of the complexities with streaming bytes from an application and most of the data being transfered are likely images and videos that already are files.

  For the "host-to-host" bridge cable, one could use a virtual network driver so that the connection would look like any other network connection. This allows use of any standard network protocol, for instance TCP/IP, ZeroConf and Service discovery as described in chapter 4. Data could then be transfered for instance as files over FTP. All USB devices needs device drivers and for USB pen-drives these are available for most operating systems. The USB host-to-host bridge cables comes with drivers for windows and there exists drivers for these types of cables for linux.
- **USB to Ethernet** This would work exactly like what is explained in chapter 4. Drivers for these adapters are available for the most common operating systems.
- **USB On-The-Go** USB on-the-go is not suitable for communication between soldier systems, see 6.2.4.

7 CONCLUSION

The goal of this report has been to find a suitable interface for exchanging large amounts of data over a wired communication channel. The interface must be as user friendly as possible for the soldier, preferably plug and play, or “plug and exchange data”. None of the interfaces discussed in this report offers this functionality out-of-the-box but they can be adapted to suit our needs.

- Ethernet is a popular interface used in for more than 85 percent of the worlds LAN-connected PCs. TCP/IP is the most used protocol in Ethernet networks and also the best suited for our needs. The drawback with the TCP/IP protocol is that each node in the network needs to be assigned an IP address. This is commonly solved by a DHCP server that hands out IP addresses to new nodes in a network. In a direct soldier to soldier network this will not be possible and we need mechanisms for choosing suitable IP addresses and communicating these to the other network nodes. Chapter 4 presents mechanisms for solving this together with ways of detecting who are on the net and what services they offer. FTP can be used for the actual data exchange since the data most likely will be stored as files. This solution will give the soldier a “plug and exchange data” solution that requires a minimum of human interaction. All the protocols at play are public standards that are available for the most common operating systems.
- FireWire has plug and play capability so it will automatically be detected by an operating system when connected. It also has the performance needed to transfer large amount of data in a short time (400 Mbit/s), and it allows for host-to-host communication. The only thing that needs to be added is a communication protocol. There is no FireWire communication protocol that is designed for host to host file sharing but FireWire can be used as a standard network connection. This allows the same network configuration mechanisms to be used as for Ethernet, see chapter 4. It is also possible to develop a proprietary communication protocol for FireWire. This will require a very detailed specification of the communication protocol so that drivers for the communication protocol can be developed for each different soldier system platform and operating system. This will require a lot of time and effort.
- USB has many of the features we want for soldier system communication: It can be connected and disconnected without rebooting, it has plug-and-play support meaning that it will be detected by the operating system and it has high bandwidth (480 Mbit/s for USB 2.0). However, the host-device architecture makes it unusable for host-to-host communication. We can use USB devices:
  - USB to Ethernet: This would work exactly like what is explained for Ethernet above and in chapter 4.
  - USB device: Use a USB pen-dive. USB 2.0 has a high bandwidth (480Mbit/s) and a lot of data could be communicated in a short time. By copying all data as files on a pen-drive, we relieve ourselves from many of the complexities with streaming bytes from an application and most of the data being transfered are likely already

files. We could also use a USB Host-to-Host bridge cable, see chapter 6, together with a virtual network driver so that the connection would look like any other network connection. This allows the use of the same network configuration mechanisms to be used as for Ethernet, see chapter 4.

Of all the possible solutions discussed above, USB pen-drive is probably the easiest. USB pen-drives have become very popular and they are probably already available and in use in many soldier systems. The Ethernet solution is also quite straight forward with well defined communication protocols. But getting these protocols to work on all platforms could be a lot of work. Any solution that requires proprietary drivers will need a lot of specification and development work on each different soldier system platform and they are therefore not very desirable.

APPENDIX

A OSI MODEL AND THE TCP/IP MODEL

The ISO (International Standards Organization) has created a layered model, called the OSI (Open Systems Interconnect) model, to describe defined layers in a network operating system. Think of the seven layers as the assembly line in the computer. At each layer, certain things happen to the data that prepare it for the next layer. The seven layers are:

- *Layer 1: Physical*
  Standardizes the electrical, mechanical and functional interface that connects the entity to the physical transmission media.
- *Layer 2: Datalink*
  Provides addressing and error control between adjacent nodes.
- *Layer 3: Network*
  Establishes communication from station to station across an internetwork and provides higher level addressing and routing. Protocols at this layer and above are independent of the Ethernet standard or any other underlaying protocol.
- *Layer 4: Transport*
  Provides reliable end to end communication control, recovery and flow control in the higher level networking software.
- *Layer 5: Session*
  The session layer provides mechanisms to establish reliable communications between applications.
- *Layer 6: Presentation*
  Provides mechanisms for dealing with data representation in applications.
- *Layer 7: Application*
  Provides mechanisms to support end user applications such as mail, file transfer etc.

A thorough description of the functionality in each layer can be found in the ISO literature. Please keep in mind that the OSI Reference Model is really just a guideline. Actual protocol stacks often combine one or more of the OSI layers into a single layer. The TCP/IP protocol stack is a good example. It uses four layers that map to the OSI model as follows:

- *Layer 1: Link level*
  This layer combines the Physical and Data link layers and routes the data between devices on the same network. It also manages the exchange of data between the network and other devices.

- *Layer 2: Network level*
  This layer corresponds to the Network layer. The Internet Protocol (IP) uses the IP address, consisting of a Network Identifier and a Host Identifier, to determine the address of the device it is communicating with.

- *Layer 3: Transport level*
  Corresponding to the OSI Transport layer, this is the part of the protocol stack where the Transport Control Protocol (TCP) can be found. TCP works by asking another device on the network if it is willing to accept information from the local device.

- *Layer 4: Application*
  Layer 4 combines the Session, Presentation and Application layers of the OSI model. Protocols for specific functions such as e-mail (Simple Mail Transfer Protocol, SMTP) and file transfer (File Transfer Protocol, FTP) reside at this level.

As you can see, it is not necessary to develop a separate layer for each and every function outlined in the OSI Reference Model. But developers are able to ensure that a certain level of compatibility is maintained by following the general guidelines provided by the model.

B ZEROCONF IMPLEMENTATIONS

There are several ZeroConf implementations available for download. Some of them are mostly applicable for test purpose, however, altogether they show the growing interest for ZeroConf solutions. Most of the implementations are based on and are compatible with Bonjour and consist of the source code and API for a combined Multicast DNS Responder and DNS Service discovery. They usually differ in the Software API, platform, and programming languages supported.

- Bonjour - Available from Apple.
- JmDNS - a Java implementation of the mDNS.
- pyZeroConf - Service discovery API for Python programs.
- Howl - An open source implementation from Porchdog software compatible with Apple Bonjour and available for most platforms.
- Avahi - Bonjour compatible implementation for Linux.
- Liaison - Zeroconf for Linux and the PocketPC platform. Bonjour compatible.
- mdnsd - Embeddable Multicast DNS Daemon.

It is worth to mention some applications and devices that take use of ZeroConf Technology.

- *Spike* - A file sharing software (Network Clipboard) available for MAC OS X and Windows XP. You don't have to install and run any ZeroConf programs first, as Spike is shipped with its own mDNSResponder.
- The *Phillips Micro Audiosystem MCW770/22* is designed to stream and play music from your computer or an Internet radio station using UPnP over IEEE 802.11b.
- An other interesting Audio Streamer is the *Roku SoundBridge*. It lets the user stream music from the computer over Ethernet and play it using regular stereo equipment. The network configuration can be done both with UPnP and Bonjour, making it fully compatible with both Apples iTunes program and Windows Media Connect.
- *D-Link DSM-604H* is a 40GB Ethernet Media Storage using UPnP configuration.

C FIREWIRE TECHNOLOGY

The features and functionality described in this chapter are referring to the IEEE 1395-1995 standard (10).

FireWire is a serial bus, meaning that only one bit at a time is transmitted over the wire. In comparison, a parallel bus is transmitting many bits simultaneously. One would think that a parallel bus would be faster than a serial bus but this is only true in theory. the problem with a parallel bus is the synchronization of the signals together with distortions created by the other wires of the bus. this has caused high speed parallel bus systems like Fast 32 bit Wide SCSI to use very high quality cables with many more signal wires than bits in the parallel bus (Fast 32 bit Wide SCSI uses a 110-pin cable). This again makes the cables very expensive, and it also limits maximum length of the cable. Hence, a serial bus is the way of the future.

FireWire 400 can transfer data between devices at 100, 200, or 400 Mbit/s data rates (actually 98.304, 196.608, or 393.216 Mbit/s, but commonly referred to as S100, S200, and S400). FireWire 800 (Apple's name for the 9-pin "S800 bilingual" version of the IEEE1394b standard) was introduced by Apple in 2003. It allows an increase to 786.432 Mbit/s with backwards compatibility to the slower rates and 6-pin connectors of FireWire 400. The IEEE 1394b specification supports optical connections up to 100 meters in length and data rates up to 3.2 Gbit/s. The original 1394 and 1394a standards used data/strobe (D/S) encoding (called legacy mode) on the signal wires, while 1394b adds a data encoding scheme called 8B10B (also called beta mode).

A FireWire devices is identified by an IEEE EUI-64 unique identifier (this is an extension of the 48-bit MAC address used in Ethernet). In addition to this, it uses codes to indicate the type of device it is and the protocols it supports.

C.1 Cable and Signals

The FireWire cable comes in two flavors: 4-pins or 6-pins. The 4-pins uses all four wires for data transmission and the 6-pins has 2 additional wires for power distribution (8-40 VDC, up to 1.5 A). The four data wires are arranged in two twisted pairs that are separately shielded to minimize distortion. In addition, the whole cable is shielded, resulting in a very good protection which makes the high transfer rates possible, see figure C.1. The data wires uses differential data transmission which means that one wire in a pair has the negative voltage of the other. If one wire at D+ (e.g. 1V), the other wire is at D- (-1V). This makes any outside noise affect both wires the same way, leaving the difference between the nearly unchanged. The receiver calculates the logical signal from the difference of the two wires. Although FireWire has two twisted pair for data transmission, it is not a parallel bus. This is because it while one pairs carry the raw data, the other one indirectly sets the clock cycle. "indirectly sets the clock cycle" means that the wire pair changes signal when the raw data pair does not. This means that one of the two pairs changes state every clock cycle, allowing the receiver to

reconstruct the clock pulse by simply XOR the signals. This technique is called NRZ (Non-Return-Zero) with Data-Strobe (DS) encoding.

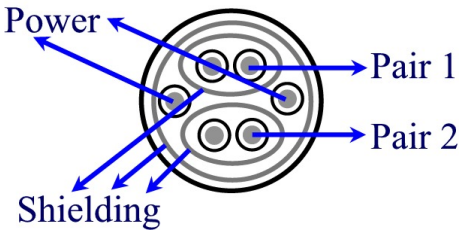

Figure C.1 Cross section of a standard 6-pins FireWire cable

A FireWire cable (S100, S200 or S400) can be up to 4.5 meters long.

C.2 Devices and nodes

Before giving an introduction the FireWire network configuration, it is important to define a few terms: A *device* is a physical thing like a video camera or a hard drive, a *node* is a point in a virtual network tree. A device can contain several nodes, however it usually contains only one. A *port* enables a node to connect to other nodes (this is a FireWire jack in the physical implementation). A node can have more than one port, allowing other nodes (child nodes) to connect “down the chain”. In this case, the node works as a repeater, passing all data to its child nodes that is not intended for itself. Nodes that only have one port will always be on the end of a chain and is called leaf nodes. In this way, the nodes can form trees and chains, but loops are not allowed. One node will work as the root-node which sits on the top (or at the bottom if you like) of the tree.

C.3 Network configuration and transactions

The 1394 protocol supports both asynchronous and isochronous data transfers.

Isochronous transfers are broadcast in a one-to-one or one-to-many fashion. No error correction or retransmission is available for isochronous transfers. Up to 80% of the available bus bandwidth can be allocated for isochronous transfers. A node in the network is given the role as isochronous resource manager and is responsible for the delegation of bandwidth. This may or may not be the root node or the bus manager. The maximum amount of bandwidth an isochronous device can obtain is only limited by the number of other isochronous devices that have already obtained bandwidth from the isochronous resource manager.

Asynchronous transfers are targeted to a specific node with an explicit address. They are not guaranteed a specific amount of bandwidth on the bus, but they are guaranteed a fair shot at gaining access to the bus when asynchronous transfers are permitted. The maximum data

block size for an asynchronous packet is determined by the transfer rate of the device as shown in table C.3

Table C.1 Maximum data block size for an asynchronous packet

Cable Speed	Maximum Data Size
100Mbps	512 bytes
200Mbps	1,024 bytes
400Mbps	2,048 bytes

Asynchronous transfers are acknowledged and responded to. This allows error-checking and retransmissions.

Whenever the topology of the bus changes, like a node is connected or disconnected, the bus needs to reconfigure itself. This is done by a bus-reset and initialization, tree identification and self identification. The process takes place as follows:

C.3.1 Reset

A reset is signaled by a node by driving both data-wire-pairs to logic1. This will always be detected by a port. When a node detects a reset signal, it propagates the signal to all the other ports the node has. The node then enters an idle state for a given period of time to wait for the reset signal to propagate to all the other nodes. The reset signal will clear all topology information within a node.

C.3.2 Tree identification

The tree identification process enables the nodes to agree on a common virtual tree structure. Here follows and example on how a tree identification process work. Lets say we have a network topology like shown in figure C.2.

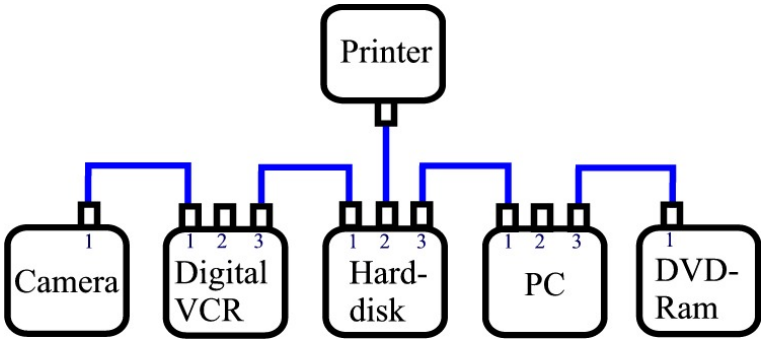

Figure C.2 Network of FireWire devices

After the reset, all leaf nodes present a Parent_Notify signal on their data and strobe pairs. This is a signal state, not a transmitted packet. In the example in figure C.2, the digital camera will signal the digital VCR, the printer will signal the hard disk, and the DVD-RAM will signal the PC. When a branch node receives the Parent_Notify signal on one of its ports, it marks that port as containing a child, and outputs a Child_Notify signaling state on that port's data and strobe pairs. Upon detecting this state, the leaf node marks its port as a parent port and removes the signaling, thereby confirming that the leaf node has accepted the child designation. After the leaf nodes have identified themselves, the hard disk still has two ports that have not received a Parent_Notify, while the digital VCR and the PC branch node both have only one port with an attached device that has not received a Parent_Notify. Therefore, both the Digital VCR and the PC start to signal a Parent_Notify on the one port that has not yet received one. In this case, the hard disk the receives Parent_Notify on both of its remaining ports, which it acknowledges with a Child_Notify condition. Because the Hard Disk has marked all of its ports as children, the Hard Disk becomes the root node. If two nodes are in contention for root node status at the end of the process, a random back-off timer is used to eventually settle on a root node. A node can also force itself to become root node by delaying its participation in the tree identification process for a while. See (10) and (2) for more details.

C.3.3 Self identification

After the network topology is defined, the self identification can begin. This consists of assigning ID's to all nodes on the bus, having the neighboring nodes exchange transmission speed capabilities and making all the nodes aware of each other. The self identification process begins with the root node sending an arbitration grant (permission to use the bus) to its lowest numbered port. For the network in figure C.2, the hard disk (being the root node) will send the signal to the digital VCR. Since the digital VCR isn't a leaf node, it will propagate the signal to its lowest numbered port with a child node which will be to the camera. Since the camera is a leaf node, it cannot propagate the signal any further. The camera then assigns itself physical ID 0 and transmit a self ID packet upstream to the digital VCR. The digital VCR repeats this packet to all its other ports with attached devices who again repeats the packet on all other ports and so on. In this manner, all attached devices will receive the self ID packet from the digital camera. Upon receiving this packet, all of the other devices will increment their self ID counter. The camera the signals a self ID done signal to its parent. The digital VCR will not propagate this signal to any other nodes because itself has not gone through the self identification process yet. The root node (the hard disk) then continues to send an arbitration grant signal to its lowest numbered port that hasn't finished self identification which still is the port where the digital VCR is connected. Because all the port with nodes connected to it on the digital VCR has gone through the self identification process, it assigns itself ID 1 and sends out a self identification packet to all its neighbors, and so on. This process continues until all nodes has indicated a self ID done signal condition. The root node will always be the highest numbered device on the bus. The devices in figure C.2 will get the following ID's: camera = 0, digital VCR = 1, printer = 2, DVD-RAM = 3, PC = 4 and hard disk = 5.

Also during the self ID process, all nodes wishing to become the isochronous resource

manager will indicate this in their self ID packet. The highest numbered node that wishes to become resource manager will receive the honor.

C.4 Arbitration

Once the configuration process is complete, normal bus operations can begin. To fully understand arbitration, a knowledge of the cycle structure of 1394 is necessary.

A 1394 cycle is a time slice with a nominal 125μs period. The 8kHz cycle clock is kept by the cycle master, which is also the root node. To begin a cycle, the cycle master broadcasts a cycle start packet, which all other devices on the bus use to synchronize their timebase.

Immediately following the cycle start packet, devices that wish to broadcast their isochronous data may arbitrate for the bus. Arbitration consists of signaling your parent node that you wish to gain access to the bus. The parent nodes in turn signal their parents and so on, until the request reaches the root node. For the case in figure C.2, lets say the digital camera and the PC wish to stream data over the bus. They both signal their parents that they wish to gain access to the bus. Since the PC's parent is the root node, its request is received first and it is granted the bus. From this scenario, it is evident that the closest device to the root node wins the arbitration.

Because isochronous channels can only be used once per cycle, when the next isochronous gap occurs, the PC will no longer participate in the arbitration. This condition allows the digital camera to win the next arbitration. Note that the PC could have more than one isochronous channel, in which case it would win the arbitration until it had no more channels left. This points out the important role of the isochronous resource manager: it will not allow the allotted isochronous channels to require more bandwidth than available.

When the last isochronous channel has transmitted its data, the bus becomes idle waiting for another isochronous channel to begin arbitration. Because there are no more isochronous devices left waiting to transmit, the idle time extends longer than the isochronous gap until it reaches the duration defined as the subaction (or asynchronous) gap. At this time, asynchronous devices may begin to arbitrate for the bus. Arbitration proceeds in the same manner, with the closest device to the root node winning arbitration.

Because asynchronous devices can send more than one packet per cycle, the device closest to the root node (or the root node itself) might be able to hog the bus by always winning the arbitration. This is dealt with using what is called the fairness interval and the arbitration rest gap. The concept is: once a node wins the asynchronous arbitration and delivers its packet, it clears its arbitration enable bit. When this bit is cleared, the physical layer (defined in C.5) no longer participates in the arbitration process, giving devices farther away from the root node a fair shot at gaining access to the bus. When all devices wishing to gain access to the bus have had their fair shot, they all wind up having their arbitration enable bits cleared, meaning no one is trying to gain access to the bus. This causes the idle time on the bus to go longer than the

10μs subaction gap until it finally reaches 20μs, which is called the arbitration reset gap. When the idle time reaches this point, all devices may reset their arbitration enable bits and arbitration can begin all over again.

For more details on the arbitration process, see (10) and (2).

C.5 Protocol Layers

The IEEE 1394 standard defines three layers in every device that serves different purposes: The Physical Layer, the Link Layer and the Transaction Layer. The first two are implemented in hardware and the Transaction Layer are often both hardware and software. The three layers correspond to the lowest three layers of ISO's Open Systems Interconnection (OSI) model (appendix A). To implement a specific device, additional protocol and application layers must be placed on top of these layers to provide the functionality of the particular device. The three layers (illustrated in fig C.3) perform the following functions:

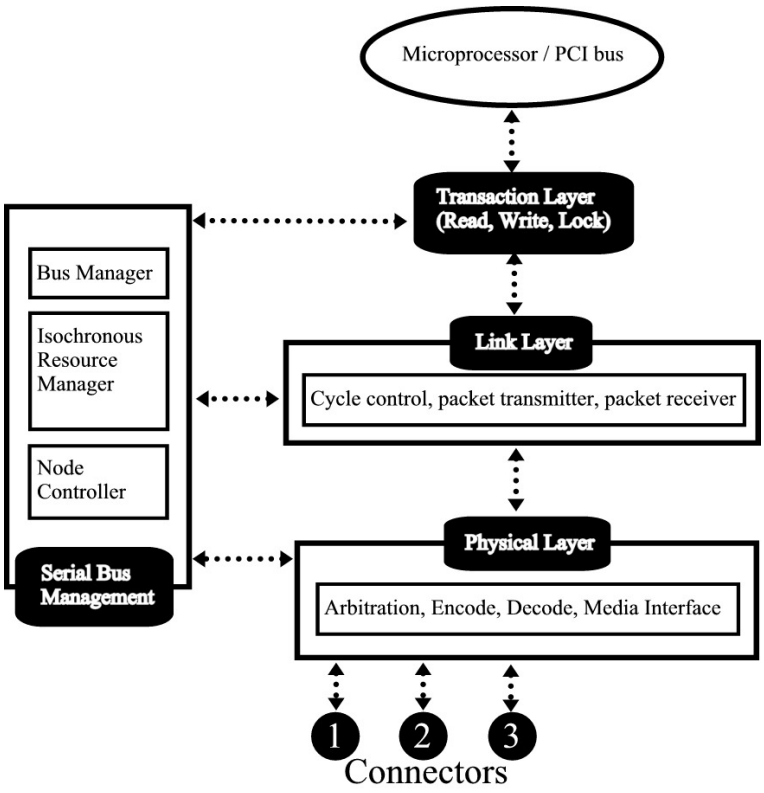

Figure C.3 1394 Layers

- The physical layer (PHY) provides the electrical and mechanical connections between the 1394 device and the 1394 cable. In addition to actual data transmission and reception, the physical layer provides arbitration mechanisms to ensure that all devices have appropriate access to the bus.

- The link layer (Link) provides data packet delivery service for both asynchronous and isochronous packets to nodes. Isochronous data packets are formatted and transferred directly to the application.
- The transaction layer supports the asynchronous protocol write, read and lock commands. A write sends data from the originator to the receiver, and a read returns the data to the originator. Lock combines the function of the write and read commands by producing a round-trip routing of data between sender and receiver, including processing by the receiver.

The 1394 serial bus management provides overall configuration control of the serial bus by optimizing arbitration timing, guaranteeing adequate electrical power for all devices on the bus, assigning which 1394 device is to be the cycle master, assigning isochronous channel IDs, and providing basic notification of errors. This bus management process connects to all three layers in the 1394 layer structure.

The IEEE 1394 architecture is consistent with the IEEE 1212 Control and Status Register Architecture Specification, which defines bus functions, address space and registers. Each node support up to 48 bits of address space (256 TeraBytes) in addition, each bus can support up to 63 nodes and up to 1023 buses can be connected together with bridges. This gives a total memory space of 16 ExaBytes.

D USB ENUMERATION PROCESS

USB supports four types of data transfer: control, isochronous, bulk, and interrupt:

- **Control** transfers exchange configuration, setup, and command information between the device and the host. CRCs validates the data.
- **Bulk** transfers move large amounts of data when timely delivery isn't critical. Typical applications include printers and scanners. Bulk transfers are fillers, claiming unused USB bandwidth when nothing more important is going on. CRCs validates the data.
- **Interrupt** transfers are not interrupts in the CPU-diverting sense. The host poll devices to see if they need service. The device can send small amounts of data that need immediate attention. Mice and keyboards use interrupt transfers. CRCs validates the data.
- **Isochronous** transfers handle streaming data like that from an audio or video devices. It is time sensitive information so, within limitations, it has guaranteed access to the USB bus. No error checking occurs so the system must tolerate occasional scrambled bytes.

All USB communication is takes place through virtual *pipes* that are connected to a logical entities in the devices called *endpoints*. An endpoint is a uniquely addressable part of a device that is the source or receiver of data. Four bits define an endpoint address. Codes also indicate transfer direction and the transaction type of the transfer. Endpoint 0 is reserved for control transfers, leaving up to 15 bi-directional destinations or sources of data within each device.

During the enumeration process, the host requests a number of descriptors from the device. These requests are done by control transfers to endpoint 0.

The following description of the enumeration process is from Jan Axelson's book "USB Complete", Third Edition (3) reproduced at `http://www.lvr.com/usbcenum.htm`. It describes the enumeration process for a computer running Microsoft Windows.

> One of the duties of a hub is to detect the attachment and removal of devices. Each hub has an interrupt IN endpoint for reporting these events to the host. On system boot-up, the host polls its root hub to learn if any devices are attached, including additional hubs and devices attached to those hubs. After boot-up, the host continues to poll periodically to learn of any newly attached or removed devices.
>
> On learning of a new device, the host sends a series of requests to the device's hub, causing the hub to establish a communications path between the host and the device. The host then attempts to enumerate the device by sending control transfers containing standard USB requests to the device's Endpoint 0. All USB devices must support control transfers, the standard requests, and Endpoint 0. For

a successful enumeration, the device must respond to each request by returning requested information and taking other requested actions.

From the user's perspective, enumeration is invisible and automatic except for possibly a message that announces the detection of a new device and whether the attempt to configure it succeeded. Sometimes on first use, the user needs to assist in selecting a driver or specifying where the host should look for driver files.

When enumeration is complete, Windows adds the new device to the Device Manager's display in the Control Panel. When a user removes a device from the bus, Windows removes the device from the Device Manager.

In a typical device, firmware contains the information the host will request, and a combination of hardware and firmware decodes and responds to requests for the information. Some controllers can manage the enumeration entirely in hardware, with no firmware support. On the host side, under Windows there's no need to write code for enumerating because the operating system handles the process.

D.1 Enumeration Steps

The USB specification defines six device states. During enumeration, a device moves through four of the states: Powered, Default, Address, and Configured. (The other states are Attached and Suspend.) In each state, the device has defined capabilities and behavior.

The steps below are a typical sequence of events that occurs during enumeration under Windows. But device firmware must not assume that the enumeration requests and events will occur in a particular order. To function successfully, a device must detect and respond to any control request or other bus event at any time.

1. The user attaches a device to a USB port. Or the system powers up with a device already attached. The port may be on the root hub at the host or a hub that connects downstream from the host. The hub provides power to the port, and the device is in the Powered state.
2. The hub detects the device. The hub monitors the voltages on the signal lines of each of its ports. The hub has a pull-down resistor of 14.25 to 24.8 kilohm on each of the port's two signal lines (D+ and D-). A device has a pull-up resistor of 900 to 1575 ohms on either D+ for a full-speed device or D- for a low-speed device. High-speed-capable devices attach at full speed. When a device plugs into a port, the device's pull-up brings its line high, enabling the hub to detect that a device is attached. Chapter 15 has more on how hubs detect devices. On detecting a device, the hub continues to provide power but doesn't yet transmit USB traffic to the device.
3. The host learns of the new device. Each hub uses its interrupt endpoint to report events at the hub. The report indicates only whether the hub or a port (and if so, which port) has experienced an event. On learning of an event, the host sends the hub a Get_Port_Status request to find out more.

Get_Port_Status and the other requests described here are standard hub-class requests that all hubs support. The information returned tells the host when a device is newly attached.

4. The hub detects whether a device is low or full speed. Just before the hub resets the device, the hub determines whether the device is low or full speed by examining the voltages on the two signal lines. The hub detects the speed of a device by determining which line has the higher voltage when idle. The hub sends the information to the host in response to the next Get_Port_Status request. A 1.x hub may instead detect the device's speed just after a bus reset. USB 2.0 requires speed detection to occur before the reset so the hub knows whether to check for a high-speed-capable device during reset, as described below.

5. The hub resets the device. When a host learns of a new device, the host controller sends the hub a Set_Port_Feature request that asks the hub to reset the port. The hub places the device's USB data lines in the Reset condition for at least 10 milliseconds. Reset is a special condition where both D+ and D- are a logic low. (Normally, the lines have opposite logic states.) The hub sends the reset only to the new device. Other hubs and devices on the bus don't see the reset.

6. The host learns if a full-speed device supports high speed. Detecting whether a device supports high speed uses two special signal states. In the Chirp J state, only the D+ line is driven and in the Chirp K state, only the D- line is driven.

   During the reset, a device that supports high speed sends a Chirp K. A high-speed-capable hub detects the chirp and responds with a series of alternating Chirp Ks and Chirp Js. On detecting the pattern KJKJKJ, the device removes its full-speed pull up and performs all further communications at high speed. If the hub doesn't respond to the device's Chirp K, the device knows it must continue to communicate at full speed. All high-speed devices must be capable of responding to enumeration requests at full speed.

7. The hub establishes a signal path between the device and the bus. The host verifies that the device has exited the reset state by sending a Get_Port_Status request. A bit in the returned data indicates whether the device is still in the reset state. If necessary, the host repeats the request until the device has exited the reset state.

   When the hub removes the reset, the device is in the Default state. The device's USB registers are in their reset states and the device is ready to respond to control transfers at Endpoint 0. The device communicates with the host using the default address of 00h. The device can draw up to 100 milliamperes from the bus.

8. The host sends a Get_Descriptor request to learn the maximum packet size of the default pipe. The host sends the request to device address 0, Endpoint 0. Because the host enumerates only one device at a time, only one device will

respond to communications addressed to device address 0, even if several devices attach at once.

The eighth byte of the device descriptor contains the maximum packet size supported by Endpoint 0. A Windows host requests 64 bytes, but after receiving just one packet (whether or not it has 64 bytes), the host begins the Status stage of the transfer. On completion of the Status stage, a Windows host requests the hub to reset the device, as in Step 5 above. The USB specification doesn't require a reset here. Resetting is a precaution that ensures that the device will be in a known state when the reset ends.

9. The host assigns an address. The host controller assigns a unique address to the device by sending a Set_Address request. The device completes the Status stage of the request using the default address and then implements the new address. The device is now in the Address state. All communications from this point on use the new address. The address is valid until the device is detached, the port is reset, or the system reboots. On the next enumeration, the host may assign a different address to the device.

10. The host learns about the device's abilities. The host sends a Get_Descriptor request to the new address to read the device descriptor. This time the host retrieves the entire descriptor. The descriptor is a data structure containing the maximum packet size for Endpoint 0, the number of configurations the device supports, and other basic information about the device. The host uses this information in the communications that follow.

    The host continues to learn about the device by requesting the one or more configuration descriptors specified in the device descriptor. A request for a configuration descriptor is actually a request for the configuration descriptor followed by all of that descriptor's subordinate descriptors. A Windows host begins by requesting just the configuration descriptor's nine bytes. Included in these bytes is the total length of the configuration descriptor and its subordinate descriptors.

    Windows then requests the configuration descriptor again, this time using the retrieved total length. The device responds by sending the configuration descriptor followed by the configuration's interface descriptor(s), with each interface descriptor followed by any endpoint descriptors for the interface. Some configurations also include class- or vendor-specific descriptors that extend or modify another descriptor. These descriptors follow the descriptor being extended or modified. Each descriptor begins with its length and type. The Descriptors section in this chapter has more on what each descriptor contains.

11. The host assigns and loads a device driver (except for composite devices). After learning about a device from its descriptors, the host looks for the best match in a device driver to manage communications with the device. In selecting a driver, Windows tries to match the information in the PC's INF files with the Vendor ID, Product ID, and (optional) release number retrieved from the device. If there is no match, Windows looks for a match with any

class, subclass, and protocol values retrieved from the device. If the device has been enumerated previously, Windows can use information in the system registry instead of searching the INF files. After the operating system assigns and loads the driver, the driver may request the device to resend descriptors or send other class-specific descriptors.

An exception to this sequence is composite devices, which can have different drivers assigned to different interfaces in a configuration. The host can assign these drivers only after the interfaces are enabled, which requires the device to be configured (as described in the next step).

12. The host's device driver selects a configuration. After learning about a device from the descriptors, the device driver requests a configuration by sending a Set_Configuration request with the desired configuration number. Some devices support only one configuration. If a device supports multiple configurations, the driver can decide which configuration to request based on information the driver has about how the device will be used, or the driver can ask the user what to do or just select the first configuration. The device reads the request and enables the requested configuration. The device is now in the Configured state and the device's interface(s) are enabled.

For composite devices, the host assigns drivers at this point. As with other devices, the host uses the information retrieved from the device to find a matching driver for each active interface in the configuration. The device is now ready for use.

References

(1) Aboba B, Thaler D and Esibov L (2005): Link-Local Multicast Name Resolution (LLMNR). (The Internet Society).

(2) Anderson D (1998): *FireWire system architecture : IEEE 1394a. - 2nd ed.* Addison-Wesley Professional.

(3) Axelson J (2005): *USB Complete, Third Edition*. Lakeview Research.

(4) Cheshire S, Aboba B and Guttman E (2005): Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927 (Proposed Standard).

(5) Cheshire S and Krochmal M (2005): DNS-Based Service Discovery. (Apple Computer, Inc.).

(6) Cheshire S and Krochmal M (2005): Multicast DNS. (Apple Computer, Inc.).

(7) Droms R (1997): Dynamic Host Configuration Protocol. RFC 2131 (Draft Standard). Updated by RFC 3396.

(8) Fout T (2001): Universal Plug and Play in Windows XP.

(9) Guttman E, Perkins C, Veizades J and Day M (1999): Service Location Protocol, Version 2 . RFC 2608 (Proposed Standard). Updated by RFC 3224.

(10) IEEE (1998): *IEEE 1394-1995: Standard for a High Performance Serial Bus*.

(11) Plummer D (1982): Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. RFC 826 (Standard).

(12) Postel J (1981): Internet Protocol. RFC 791 (Standard). Updated by RFC 1349.

(13) Spurgeon C E (2000): *Ethernet, The definitive guide*. O'Reilly.

(14) USB (1998): Approved USB Device Class Specifications. (`http://www.usb.org/developers/devclass/`).

(15) USB (2000): Universal Serial Bus specification.

(16) USB (2003): On-The-Go Supplement to the USB 2.0 specification.

(17) Wikipedia (2005): FireWire. (`http://en.wikipedia.org/wiki/Firewire`).