

IP over IEEE-1394: Using the Internet Protocol over a high-performance serial bus

There is a wide variety of communication systems currently available on the market, and each one offers its own advantages and inconveniences when compared to its counterparts. Powerful IEEE-1394 connectivity implemented with the IPv4 protocol is ideal for the straightforward integration of multiple heterogeneous computer platforms, which are often necessary to provide sufficient horsepower for today's demanding applications.

This article will begin by providing background information on need for a flexible protocol and the IEEE-1394 High-Speed Serial Bus. We will then give an overview of its two communication schemes and explain how the IP over 1394 protocol works.

NOT ANOTHER PROTOCOL!

In the past, system integrators were forced to rewrite their system's communication code for compatibility every time the communication platform was changed. To resolve this never-ending incompatibility dilemma, most integrators have opted to use a higher-level protocol, Internet Protocol (IP), for inter-node communication. IP has become today's standard communication protocol: regardless of the communication system used, if the communication aspect of the application has been written using standard IP, no modifications are required on the software side when porting to a new communication platform.

Developing inter-system communication presents several challenges. These systems are typically heterogeneous environments with different buses, and communication is based on the IP protocol. They require high-priority, collision-free communication between selected equipment, reserved bandwidth exceeding 100Mbits per second and low, predictable latency. In order to reduce total system costs, inter-equipment communication requires a cost-effective solution that is based on a well-established standard for longevity and hassle-free replacement of system components or vendors.

Until recently, system integrators felt it was nearly impossible to develop an inter-equipment communication system without compromising on any of the above requirements. Even if all these requirements were met, most engineers were still faced with the prohibitive costs of the inter-equipment communication technologies currently available on the market. Most are simply not flexible enough to cover the range from modest price-sensitive systems to expansive high-performance developments.

THE 1394 EXPLAINED

Over the past few years, the IEEE-1394 high-speed serial bus has transformed the communication systems in mass market devices such as computers, printers,

scanners, camcorders, digital TVs and DVD. Today, 1394 is making its mark in the embedded and industrial world. A prime example is companies that manufacture and integrate their own equipment with third-party systems using standard commercial buses such as PCI (PCI, CPCI and PMC) and VME.

The 1394 is a high-speed serial bus, and not a network. By viewing the communication system as a bus, the developer can map the entire memory resources of all nodes into all other nodes, which can be considered as a vast system-wide shared memory pool (distributed shared memory.) Memory locations can be accessed through quadlet-read, quadlet-write, block-read, block-write and lock functions. A form of communication based on channels has also been implemented.

Using a bus structure means that 1394 is a collision-free medium managed by a full arbiter. The 1394 arbiter respects a fairness protocol, which prevents the bus from being monopolized by the fastest nodes (all nodes have an equal opportunity to send.)

ASYNCHRONOUS VERSUS ISOCRONOUS COMMUNICATION

The 1394 communication scheme is based on two basic methods of communication: asynchronous and isochronous.

The *asynchronous communication* method uses quadlet-read, quadlet-write, block-read, block-write and lock functions, as well as an asynchronous channel type of communication called asynchronous streams. In asynchronous mode, communication is directed to either a specific node (point-to-point or unicast) or to all nodes (broadcast.) For read, write and lock functions, the internal address of the node where the action is to take place is specified within the packet header. All transactions are sent along the bus using the fairness protocol, and only when no isochronous cycle is in progress.

Asynchronous communication uses a best-effort com-

munication strategy; communication is secure when using quadlet-read, quadlet-write, block-read, block-write and lock functions because an acknowledge status is received for every packet that is sent. A time-out is indicated if no acknowledge is received after a certain period of time has elapsed. If an error occurs, the correction or retry is managed in the lower protocol layers (usually in the hardware.) When using asynchronous streams, however, no acknowledge is received, leaving error handling to the higher-layer protocols.

The second method is known as isochronous communication. Every 125µs, an isochronous cycle start is broadcast by the root node. Once all nodes have received this packet, an isochronous communication window opens. All nodes waiting with pending isochronous packets will then send. Once either all isochronous packets have been transferred or 100µs (80% of the bandwidth) has elapsed since the isochronous cycle start was broadcast, the isochronous communication window closes and asynchronous communications are allowed to take place until the next 125µs cycle begins.

Isochronous communication uses a channel scheme where nodes transmit over a numbered channel and receive by listening on the receiver channels. This approach allows unicast, multicast and broadcast communication. The isochronous method is a time-controlled manner of communication and has priority over asynchronous communication. Every 125µs, an instance of every channel in use (maximum of 64 channels) may be sent. This type of communication is referred to as a quality service because of its reserved bandwidth. In contrast to asynchronous communication, isochronous communication does not implement

acknowledge handshaking, and error handling is left to higher-layer protocols.

IP OVER 1394

IPv4 is a software layer that interfaces between the operating system's IP stack and the 1394 stack. This layer manages data structures and the encapsulation process for the transportation of IP datagrams, as well as the Address Resolution Protocol (ARP).

In order to communicate between nodes supporting IPv4, each node must build a translation table using ARP to translate each IP address to the corresponding 1394 address (the memory-mapped address of the receiver or the receiving channel number.) An ARP request broadcast packet is sent to request the 1394 address or channel of the IP node in question. All nodes supporting IPv4 will then compare the IP address given in the ARP request with their own IP address. If the node's IP address matches the requested IP address, it responds by sending an ARP response packet to the ARP requester containing the responder's 1394 address or channel number.

Once the sender node has resolved the link between the IP address and the target node's 1394 address, the sender can begin transmitting IP datagrams to the target.

IP datagrams can be sent using isochronous streams, asynchronous streams or asynchronous block-write requests.

When sending IP datagrams via isochronous streams, the datagrams must be transmitted in channels. The advantage of this method is that the communication path can have reserved bandwidth. The disadvantage is that a channel must be reserved for every path, and channels are a precious and limited resource in 1394 (maximum of 64 channels.) This method is recommended for specific communications that require fixed bandwidth.

When sending IP datagrams via asynchronous streams, the datagrams must also be transmitted in channels. The advantage of this method is the possibility to complete multicast-type transactions. The disadvantage is that a channel must be reserved for every path, and channels are a precious and limited resource in 1394 (maximum of 64 channels.) Packets sent using this method must employ the best-effort communication strategy. This transaction type is only recommended for multicast communications.

When sending IP datagrams via asynchronous block-write requests, the datagrams must be transmitted in memory maps. The advantage of this method is that the sender does not use channels. Communication errors or retries can most often be resolved in a lower layer (or even in the hardware.) The disadvantage is that only broadcast and unicast transactions can be completed in this manner. When using this method, packets are sent using the best-effort communication strategy. This method is recommended for unicast and broadcast communications when fixed bandwidth is not required.

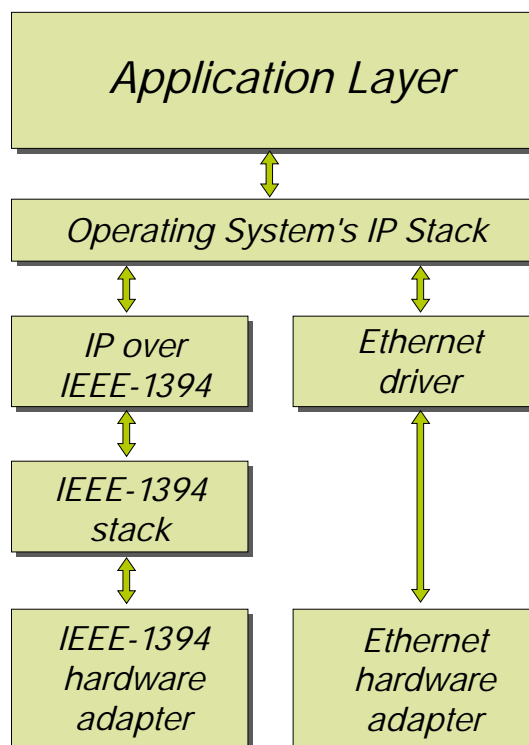


Figure 1. IP Implementation.

IP DATAGRAM FORMAT

The IP datagrams are encapsulated in 1394-formatted packets (the format depends on the communication scheme being used.) The IP datagram plus an extra data quadlet are encapsulated in the 1394 packet payload.

An extra quadlet prefixes the IP datagram to indicate if the IP datagram is complete or fragmented. If the packet is fragmented, this prefix indicates whether the received packet is the first, middle or last part of the IP datagram.

Unfragmented encapsulation prefix.

The following section explains the prefix format.

<i>Lf</i>	<i>Reserved</i>	<i>ether_type</i>
Bit 0-1	Bit 2-15	Bit 16-31

Table 1.

lf: This field is zero for unfragmented IP datagrams.

ether_type: This field indicates the nature of the upcoming datagram.

Note: Other network protocols, identified by different *ether_type* values, may use the encapsulation formats defined here; however, such use is outside of the scope of this document.

Fragmented datagrams have a 2-quadlet prefix.

<i>ether_type</i>	<i>Datagram</i>
0x800	IPv4
0x806	ARP
0x8861	MCAP

Table 2.

First fragment of the datagram

<i>Lf</i>	<i>rsv</i>	<i>buffer_size</i>	<i>Ether_type</i>
Bit 0-1	Bit 2-3	Bit 4-15	Bit 16-31

Table 3. First quadlet for the first fragment of the datagram.

<i>Dql</i>	<i>Reserved</i>
Bit 0-15	Bit 16-31

Table 4. Second quadlet for the first fragment of the datagram.

Subsequent fragments of the datagram

<i>Dql</i>	<i>Reserved</i>
Bit 0-15	Bit 16-31

Table 5. First quadlet for the subsequent fragments of the datagram.

<i>Dql</i>	<i>Reserved</i>
Bit 0-15	Bit 16-31

Table 6. Second quadlet for the subsequent fragments of the datagram.

lf: This field specifies the relative position of the link fragment.

<i>Lf</i>	<i>Position</i>
0	Unfragmented datagram
1	First fragment
2	Last fragment
3	Interior fragments

Table 7.

Buffer_size: The size of the buffer, expressed as *buffer_size* + 1 octet, necessary for the reception or reassembly of the link fragments.

ether_type: This field is only present in the first link fragment and contains the value 0x800, indicating the IPv4 datagram.

fragment_offset: This field is only present in the second and subsequent link fragments and specifies the offset of the fragment, in octets, from the beginning of the IP datagram.

dgl: The value of *dgl* (datagram label) is the same for all link fragments of the IP datagram. The sender increments the *dgl* for successive, fragmented datagrams; the incremented value of *dgl* will wrap from 65,535 back to 0.

All IP datagrams, regardless of the transmission method, are preceded by one of the encapsulation headers given above. This approach allows all datagrams from all transmission methods to be processed equally.

CONCLUSION

IPv4 is proving to be an ideal solution to inter-system communications complexities with its scalable architecture, flexible peer-to-peer topology, low cost implementation, high-speed deterministic communication, and built-in synchronization of communications and processes between devices. In addition, 1394 is a well-established standard in the mass market. Adopting this standard for industrial applications has brought new possibilities for real-time inter-equipment communication ■

As Chief Technologist Santamaria is responsible for the development of new markets as well as the technological directions of the company. Santamaria is one of the founders of Sederta Inc. as executive Vice President in charge of Research and Development, Santamaria was responsible for the design and development of the company's hardware and software products. He has played a key role in the development of the enterprise.

Prior to joining Sederta, Santamaria served as Hardware Engineer at Alex Informatique, where he participated in the development of the company's second generation parallel computers as well as projects in speech and image recognition. His technical skills and expertise were honed during his tenure as Technician for ITC Inc. and Compugraphique Canada. Santamaria holds a degree in Electronic Engineering from L'Ecole de Technologie Superieure de Montreal, as well as a Technician Degree in Electronics from L'Institut Teccart de Montreal.

AD CCII