PVC: An Application of Embedded Linux

Alfred Reynolds
Defence Science and Technology Organisation, Communications Division

ABSTRACT
The Australian Army currently fields two main information technology and telecommunication systems to commanders in the field, a telephone system (Parakeet) and an IT system (BCSS). Currently, these two systems are disparate with interconnection only at the WAN. In the commercial world there is a trend to integrate voice and data systems to share a common transport mechanism. This convergence can be seen with applications such as Voice over IP.

This paper details the design and construction of the Parakeet Virtual Cable (PVC), a device that enables Parakeet telephone traffic to be carried across the BCSS local area infrastructure. The device consists of several custom printed circuit boards and an embedded Linux PC in the form of a uCSimm from Lineo. The paper will cover the various sub-systems comprising the device and a discussion of some novel solutions to the problems encountered due to the low processing power of the embedded Linux PC.

1. Introduction

When an army is in the field, it needs to be able to communicate with its units and other forces. There are currently two types of communication systems used for this purpose, the combat net radio and the trunk communications system. For the Australian Army, the trunk communications system is Parakeet. Parakeet is a military standard circuit switched system similar in concept to civilian Narrow-band Integrated Services Digital Network. Parakeet offers 16 kbps (kilobits per second) channels to the end user, which can be used for either voice (using a military standard waveform encoder called Continuously Variable Slope Delta Modulation, (CVSD)) or data.

As technology advances it is possible to automate business processes, and for the Army this involves deploying PCs with command and control software. For the Australian Army this system is called the Battlefield Command Support System (BCSS). It provides notebook PCs and an ethernet local area network (LAN) to connect them together in the local area (within a headquarters). Wide area connections are via routers that run over the Parakeet or via gateway PCs connected to combat net radio.

There is a problem with these two disparate systems within the headquarters. Both require communications assets to be deployed around headquarters, requiring two separate systems to do this (field telephone cable or fibre optic for Parakeet and independent LAN cabling for BCSS). This dual cabling requires more materials and man-hours than an integrated system.

Technologies such as VoIP allow voice and data services to share a common transport network. The commercial world is quickly adopting this converged concept due to the economics of maintaining one physical network, the flexibility and control it allows and the new services it can provide. The military also sees benefit in having one common infrastructure to carry traffic in the local area, with an important outcome being a single network to lay, maintain and manage.

While there is currently no formal converged network in place in the Australian Defence Force there does exist a system that could provide those features. The BCSS LAN (mentioned in the introduction) provides a physical network that could provide converged services.

The goal for this project was to facilitate the integration of the current Parakeet telephone over the BCSS LAN to progress toward the concept of a converged transport network. The final system must be able to interoperate with the existing trunk voice communications systems. It must also use the existing BCSS LAN infrastructure as the transport layer and when deployed it must be manpower-neutral.

An important consideration in creating a converged network is the functionality that is required by a military telephone system. Value added services offered by the Parakeet circuit switch do not exist on commercial telephony switches yet they are required in a military environment. The final solution must have these military features, which suggests keeping the existing system.

This paper will cover the design and implementation of a Linux based device that solves these requirements. It starts with a discussion on the concept used and its components. The implementation of the system is then examined with reference to the 3 subsystems implemented. An in depth look at the subsystems is then described with a view to the hardware and software sides of the systems. Finally the outcomes of the system are briefly discussed.

2. Concept

With the requirements in mind a solution involving bridging between the digital interface on a military telephone and the BCSS LAN was designed.

The picture below shows how the concept would be used operationally.

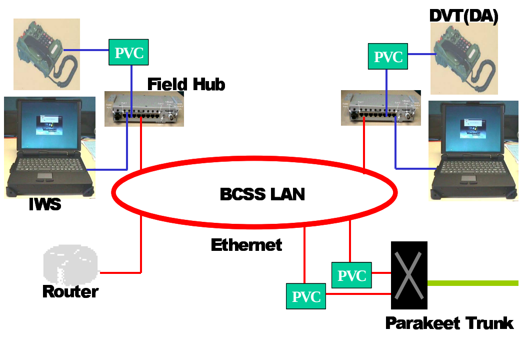

Figure 1 - PVC operational concept

The PVC will use the BCSS ethernet LAN to transport the Parakeet telephone data. It will also comprise circuitry to convert the physical data format from ethernet packets into the signals expected by Parakeet telephone terminals. The Parakeet telephone terminals are known as Digital Voice Terminal/Data Adaptor or DVT(DA)'s. By using the current Parakeet telephony terminals and switches no functionality will be lost but it will still be possible to operate a converged network in the LAN.

There are two distinct problems to be solved in the design of the PVC:

- Connection of the Parakeet elements to a microprocessor. In itself, this requires:

- conversion of the two wire signal (Conditioned Di-phase [CDP]) used by Parakeet into a four-wire[1] synchronous bitstream of the correct voltage levels (TTL) and
- getting the bitstream into a processor (typically designed to manipulate bytes rather than bits).

- The packetisation and transmission of the data over an ethernet LAN with processing to handle latency, jitter and packet loss.

The device requires small data packets to be sent regularly across an ethernet network. This network will add jitter (variation in delay) and loss into the data stream. The receiving PVC must be able to react to the conditions of the ethernet to keep a continuous data stream to the DVT(DA).

An accurate clock source is required so that appropriate decisions can be made about packets arriving from the network in order to de-jitter the incoming packet stream. This clock does not need to keep absolute time, it only needs to be accurate relative to the data stream.

The device must also be able to interface the military standard synchronous interface of a DVT(DA) to an asynchronous ethernet network. It must also achieve this within a minimum amount of space and electrical power.

[1] Four wire in this context means one receive signal and its clock, and one transmit signal and its clock.

3. System Components

3.1 System Overview

The final design for the device consists of three major system elements that process the: conversion of the DVT(DA) data waveform to a synchronous TTL compatible data stream, synchronous to asynchronous data conversion and asynchronous data to ethernet.

The DVT(DA) to synchronous system will convert the military standard waveform emerging from the DVT(DA) telephone into synchronous TTL data streams that microprocessor based circuits can utilise. This system will be called the "two-wire to four-wire converter".

The next component is the synchronous to asynchronous converter. This section is called the "4-wire to RS232 converter". It provides the buffering logic to convert from a continuous, synchronous data stream to an asynchronous, byte-based stream with jitter between packets.

The final component is the asynchronous to ethernet stage that is called the "RS232 to ethernet stage". This system collects the asynchronous bytes it receives into packets to be transmitted over the ethernet LAN.

The data flows and interfaces used in this design are shown below:

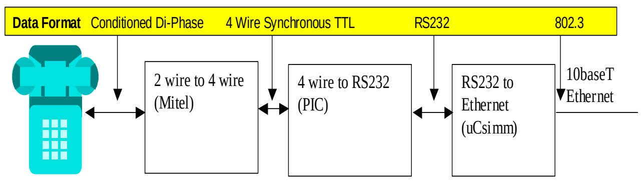

Figure 2 - Major system components

The design of the three subsystems will now be discussed.

3.2 2-wire to 4-wire (Mitel)

The Mitel chip [1] acts as a modem, converting the 16 kbps digital data from the telephone system into an analog waveform for transmission over wires and vice versa. The Mitel chip uses a Conditioned Di-Phase signal with smoothing as the analog waveform. The current DVT(DA) uses this Mitel chip for its modem interface so using this chip ensured the correct decoding of the DVT(DA) data waveform. The chip is deliberately underclocked to produce the 16kbps data stream rather than the 160 kbps that the chip was designed for.

3.3 4-wire to RS232 (PIC)

This sub system converts the constant bit rate, bit oriented data stream from the Mitel device into an asynchronous, byte based RS232 data stream for the uCsimm microprocessor. We accomplish this by using a PIC-Microchip[2] to provide the interface and logic between the two types of data flows.

The PIC chip provides an interface between the asynchronous serial port on a RS232 capable device and the 16 kbps synchronous stream that the Mitel chip requires. The PIC has two internal buffers, one for incoming and one for outgoing data. The incoming buffer (i.e. data from the Mitel chip) is 90 bytes long. The outgoing buffer (i.e. data to the Mitel chip) consists of two separate 70 byte buffers (see the cadence concept in Appendix A.1 for details).

3.4 The uCsimm

The uCsimm is a Linux based PC on a ram stick. It consists of a 16MHz 68EZ328 DragonBall Microcontroller, 8MB of RAM, 2MB of flash ROM, a RS232 serial port, a 10baseT Ethernet port and 18 I/O lines in a small footprint. For this application, the important parts are the RS232 serial port and the 10baseT Ethernet connection. The uCsimm receives bytes from the PIC via its RS232 port and then packetises them and sends them out of the Ethernet port. It also does the reverse to create full duplex communications. The uCsimm was chosen as the ethernet interface device for two reasons. The first is its flexibility, the uCsimm runs the Linux OS so programs can be written quickly and efficiently due to its open nature. The second reason was the small footprint of the device.

4. Implementation Details

4.1 Buffering

Due to the low processing power in the system a novel way to handle de-jittering of the incoming (ethernet) data stream was required. Details of the solution are described in Appendix A but a quick overview will be provided here.

The arrival packets from the ethernet interface are jittered due to the nature of ethernet. This jitter must be handled and removed before the data can be sent to the synchronous components of the system. The clock present on the uCSimm has a granularity of the same order as the packet arrival times so another mechanism must be used to handle the timing aspects of the stream.

The solution implemented consists of two components, a dual buffer situated in the PIC chip and a handshake line between the PIC and the uCSimm. This handshake line is used to co-ordinate packet delivery to the PIC's dual buffer. A diagram of the buffer concept can be seen in Appendix A.1.

[2] Microchip, http://www.microchip.com, produces a range of low cost programmable integrated circuits.

4.2 Linux

The operating system of choice for this device is Linux. Linux provides the soft real-time performance needed for the system while providing simple, high level API's for system access. The access to the source code of the linux kernel also provided the ability to adapt the system to any specific needs if required (such as altering the RS232 code to handle raw data, see 5.3.3).

5. System Details

5.1 2-wire to 4-wire (Mitel)

The Mitel chip interface is derived from the standard design shown in the Mitel MT9172 specification [1]. The only issue in adapting the design was locating a source for the special transformers required to create the unbalanced input signal. The standard circuit design (as per the specifications sheet for the product) calls for a "2 to (1/2+1/2)" balanced transformer. No source for this component was found so two "2 to (1+1)" transformers (RS-Components 210-6346 [2] pulse transformers) were used instead. The Mitel chip then demodulates the signal back into a 4-wire baseband TTL signal that is fed into the synchronous TTL to RS232 circuit.

5.2 4-wire to RS232 (PIC)

5.2.1 Hardware

A logical diagram of the implemented system can be seen below.

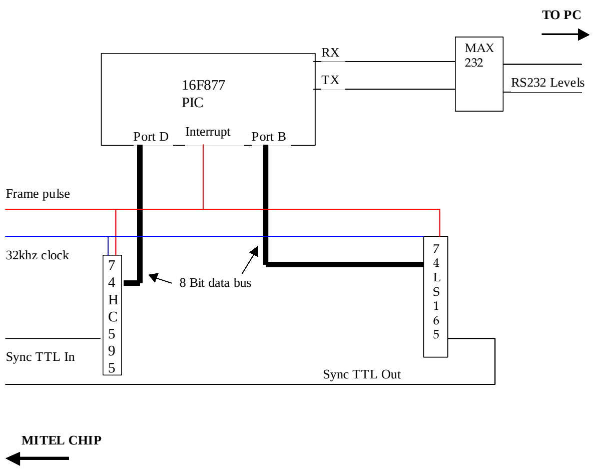

Figure 3 – Block diagram of the Synchronous TTL to RS232 sub-system

The components of this subsystem are now discussed with the next section providing a focus on the logic contained within the PIC chip.

5.2.1.1 TTL Input to TX on RS232

The 16 kbps TTL level stream is clocked into a shift register with latches (in this case, a 74HC595) by the 32 kHz clock produced by the Mitel chip. When the frame pulse is triggered (see below for a description of the frame pulse) the 74HC595 loads the contents of its internal shift registers onto its 8 output pins. The frame pulse also interrupts the PIC chip and triggers it to read the value of the port D pins. This data is then put into an internal buffer and is sent out of the RS232 port of the PIC chip in an asynchronous manner. The MAX232 is used to shift the voltage levels to the RS232 standard from the TTL levels produced by the PIC chip.

5.2.1.2 RX on RS232 to TTL Output

A character is transmitted by the uCsimm down the RS232 line into the MAX232 where the voltage levels are shifted to the TTL levels expected by the PIC chip. The character is then read and copied to an internal circular buffer. When an interrupt is received from port D (through the interrupt pin triggered by the TX side) the current character in the circular buffer is presented onto port B and the circular buffer is incremented to point to the next character. This character (comprising 8 bits) is read by an 8 bit parallel to serial converter (for this circuit a 74LS165 is used) and copied to its internal shift registers. These registers are then clocked out of the chip by the 32 kHz clock and onto the TTL out line.

5.2.1.3 Frame Pulse

The frame pulse is a signal that indicates the byte boundaries in the Mitel data stream. It is pulled low half way through each 8th bit. The PIC is a byte based machine, so this signal is used to convert the synchronous incoming bit stream into a byte orientated data stream.

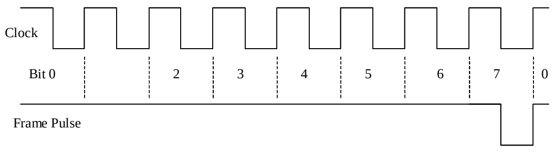

Figure 4 – The Frame Pulse Signal

The Mitel chip currently generates this frame pulse, but it can be created by discrete logic if required.

5.2.2 Software

The program created for the PIC implements the dual buffer detailed above. Its two main procedures will also now be discussed.

5.2.2.1 Buffering

The PIC has a unique outgoing buffering design. Writes into the buffer (from the RS232 port) occur in packet size chunks of 70 bytes. Reads from this buffer are byte based, with one byte being read at a rate of 2 kHz (1/8 of 16 kHz).

The PIC chip has two 70 byte buffers allocated on start-up for data transmission from the ethernet to the DVT(DA) (as seen in Figure 5). It also allocates one 90 byte circular buffer for data going from the DVT(DA) out onto the ethernet.

5.2.3 Interrupt Service Routine

This procedure services port D (the parallel port of the PIC) and the receive line for the RS232 data. If a parallel port interrupt is received then it copies the data from port D to the current input buffer (to uCsimm/PC) and increments the read out pointer. If the read out pointer is at the boundary of its current buffer (see Appendix A.1 for a diagram of the boundaries) the following algorithm is followed:

```
   if (write out pointer is
at start of next buffer) {
       reset read pointer to
start of current buffer;
   } else {
       switch read pointer to
new buffer;
   }
```

Pseudo Code 1 - PIC algorithm for controlling the read pointers bounce back effect

If the RS232 controller causes the interrupt however, it copies the incoming character into the output buffer ready for transmission to the Mitel circuit.

5.2.3.1 Main Loop

The main loop scans the circular input buffer and sends new data from it to the RS232 TX port. Therefore the buffer will be emptied whenever the PIC is not servicing the interrupt routine. The PIC has enough processing power that this buffer should never have more than 10 bytes waiting to be transmitted. The 56 kbps data rate to the RS232 port means that this buffer can be emptied 3 times faster than it can be filled (from the Mitel chip) so a backlog of data will not build up.

5.2.3.2 Extra Features

The PIC has an internal watchdog timer that detects program errors (crashes) and automatically resets the chip on such conditions. This makes the PIC robust to any possible error conditions, as it will just reset to the initial startup state that the uCSimm can react to.

5.3 The uCsimm

The main logic of the program is best described by the pseudo code in Appendix B.

The program uses the heartbeat from the PIC to throttle the delivery of packets to the PIC. This method was chosen because of accuracy of the clock with respect to the events in the data stream and because of the very low processing power needed to implement this design.

The loop has two main sections, the section for handling the arrival of packets from the Ethernet and the code dealing with reading of bytes from the serial port. The main loop starts with a **select** function call. This function waits for a file descriptor to become ready due to data arriving. When one or more file descriptors is available the call returns and execution continues.

5.3.1 Ethernet Packet Arrival

The first section handles the arrival of Ethernet packets. When a packet arrives, it is read into a memory buffer and a check is performed to determine whether it should be written out of the serial port. If the state of the heartbeat pin on the PIC has changed from last time a packet was written, then the packet is written to the serial port if no other packet is currently buffered. If one is buffered then the buffered packet is written to the serial port and the new packet is put into the buffer in its place. However, if the heartbeat pin hasn't changed since the last packet the new packet is buffered. The packet buffer is only one packet deep, so when a new packet is buffered the packet currently in the buffer is effectively discarded. There is one final check, if the heartbeat pin has changed and if there is a buffered packet, that packet is written to the serial port. This check is performed because it is possible for packets to be delayed for long periods, so the writing of any buffered data still needs to be performed if a new packet hasn't arrived.

5.3.2 DVT(DA) Data

The second section of the code deals with packetising the data arriving from the DVT(DA). The first **if** statement copies any available data from the serial port into a linear buffer. Next, a statement checks if there is one packet worth of data currently in the buffer. If there is enough data a packet is constructed and sent out of the Ethernet port. The number of bytes of data in the incoming buffer is also checked. If this buffer gets larger than two packets then the extra data is dropped. If this didn't happen it would be possible for the two ends of the connection to slowly slip out of time with each other as more and more data is buffered.

5.3.3 Kernel

There was one issue with the Linux kernel running on this device. The serial port code had a section that reacted to byte values of 0x10 and 0x12 being sent down the line. The problem was that the DVT(DA) would produce this character under normal operating conditions. So, the source code of the kernel was altered to remove this functionality.

6. Conclusions

This paper detailed the construction of the PVC device that enables the use of Parakeet telephones over the BCSS LAN. This allows for the LAN to act as a convergence layer for voice and data without loosing any of the functionality present in the Parakeet telephony system.

The device consists of an uCsimm (an embedded Linux PC), a PIC chip and a Mitel modem device. The device provides a transparent link between the two endpoints by tunnelling the voice data in ethernet packets. Various novel algorithms were used to achieve the buffer control needed due to the unreliable nature of ethernet packet delivery and the low processing power of the components used. Linux was used due to its ability to handle the soft real-time data stream and due to its open architecture and source code availability.

7. Acknowledgements

The device described in this paper is the outcome of work done by W.D. Blair and myself. This paper borrows heavily from the earlier work, further details about the project and its outcomes can be found in the paper "*Parakeet Virtual Cable Concept Demonstrator*" DSTO-TR-1113, March 2001, http://www.dsto.defence.gov.au/corporate/reports/DSTO-TR-1113.pdf.

8. References

[1] Mitel MT9172 Data Sheet, http://assets.zarlink.com/products/data/datasheets/zarlink_MT9172_Jan_2001.pdf.

[2] RS Components product 210-6346, http://www.rs-components.com.au.

Appendix A: Buffer Concepts

The PIC contains a two-packet buffer, with one side being written into from the uCsimm while the other is being read out of to the Mitel/DVT(DA). This system of writing a packet down into a separate buffer removes two problems:

- The prime purpose of this buffer is to provide about one packet worth of de-jitter buffer within the PIC.
- The system also assists in avoiding a clipping effect caused by the read pointer over-taking the write point and then the write pointer over-taking the read pointer. The bounce-back effect described later ensures that this cannot happen.

The design is visualised in Figure 5:

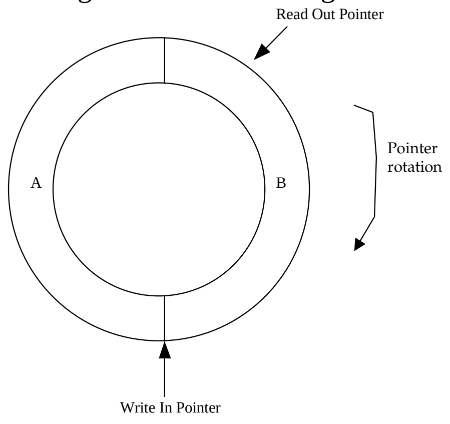

Figure 5 - Dual buffers

The role of the uCsimm in this buffering is to provide another level of packet buffering. With assistance from the PIC (and packet sequence numbering), the uCsimm makes decisions to discard packets that have arrived from the ethernet too late.

A.1. Cadence Concept

The clock available to software on the uCsimm is only accurate to +/- 10 msec. When the packet lengths are within this order of magnitude the internal clock cannot be used in deciding whether to drop a packet when it is too late. Since the PIC has direct access to the data stream and its clock, the concept of using a heartbeat from the PIC was created. A pin on the PIC is used to indicate which half of the buffer is providing the packet currently being sent to the Mitel/DVT(DA). This information can then be used by the uCsimm to control the rate and timing of when packets are sent to the PIC. Each packet should be sent from the uCsimm into alternate PIC buffers, i.e. the heartbeat must change state between each packet transmission.

Referring to Figure 5. When the read out pointer is in buffer 'A', the cadence pin on the PIC will be at the zero level. When the read out pointer is in buffer 'B', the cadence pin will be at the logical one level.

A.2. Reset PIN

The ability to reset the PIC to a known state is vital for the operation of the cadence concept. By asserting this pin on the PIC the read pointer resets to the top of the 'B' buffer and the write pointer is reset to the top of the 'A' pointer (the bottom of the circle as seen in Figure 5). On powerup the uCsimm uses this reset pin to reset the PIC chip to a known state.

A.3. Read Pointer Bounce Back

As a byte of data is read out of the buffer, the value of that position is reset to a default value, in this case a "10101010" bit pattern. This pattern is chosen as it is the pattern produced out of a CVSD coder when the speaker is silent.

The read pointer must only swap buffers when a new packet has been successfully written from the uCsimm. To achieve this, the read pointer has the "bounce back" concept.

The write in (from the uCsimm) pointer essentially moves in packet length bursts (i.e. it essentially toggles between 0 and 180 degrees on the circle) whereas the read pointer is byte based. When the read pointer reaches the end of a packet it checks whether the write pointer is at the same point. If it is then this means that there is no new data in place for the PIC to read out to the Mitel/DVT(DA) and so the read pointer is reset to a diametrically opposite position, i.e. bounced back to the start of the packet buffer just read.

The action of resetting the pointer produces two effects. The first is the insertion of a packet worth of silence into the data stream, required when the ethernet traffic has a large jitter. The second effect is to maintain the cadence signal at the same level for two packet periods. This is used by the uCsimm along with packet sequence numbering in its decision to discard a packet received off the ethernet.

A.4. Examples

The effects of the algorithms implemented in the PIC chip and the uCsimm can best be seen by the following examples. The details of the control algorithms are discussed in the detailed design section.

A.4.1 Normal Operation

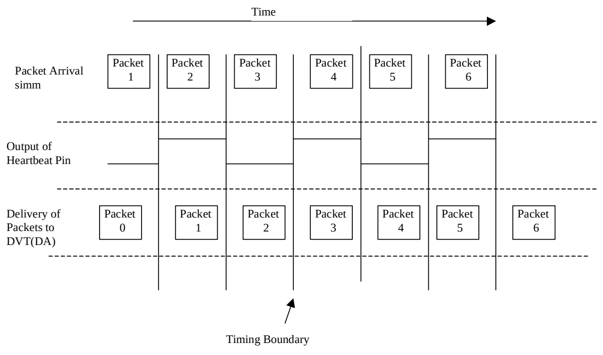

Figure 6 - Normal operation of the device

This example demonstrates the normal operation of the device. All packets arrive on time and are sent to the Mitel/DVT(DA) straight away. The cadence effect can be seen. The uCsimm sees each packet arriving in turn and the PIC cadence signal alternates its value when the packets arrive at the uCsimm indicating that they can be written into the PIC. Note that only the digital stream is passed to the PIC, no packet numbers are available to it.

A.4.2 Packets Dropped by the Network

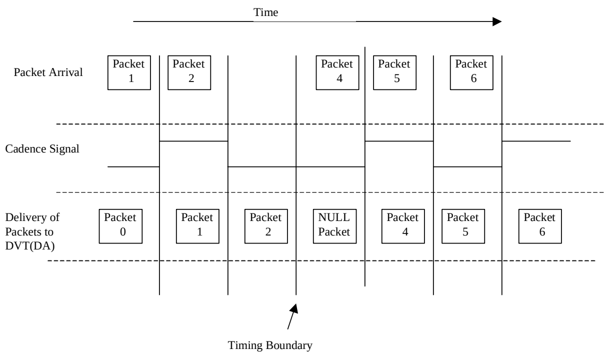

Figure 7 - The effect of packets dropped by the network

If a packet is dropped by the ethernet then the PIC will eventually underflow the buffer and the read pointer will bounce back. Within the time resolution of the uCsimm the cadence is still operating normally, i.e. at each packet arrival the cadence is inverting. However, the uCsimm can see the packet sequence broken but it knows it can immediately send the current packet surmising that the PIC will be sending a null packet to fill the sequence gap.

A.4.3 Delayed Packets

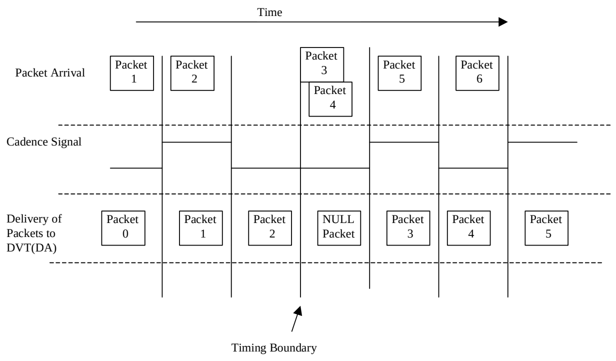

Figure 8 - The effect of late packets

This diagram shows the effect of a late packet. In the example packet 3 is delayed by a whole time increment (one packet worth of audio data, which is 35 msec). The uCsimm does not have the time resolution to see that the packet is late, but the cadence signal indicates it can be written down to the PIC. When packet 4 arrives, the cadence signal has not changed so it is buffered in the uCsimm, effectively adding an additional 35 msec of dejitter buffer. In due course the cadence resumes its rhythm and the process continues with an additional dejitter delay from the packet buffered in the uCsimm.

A.4.4 Extremely Late Packets

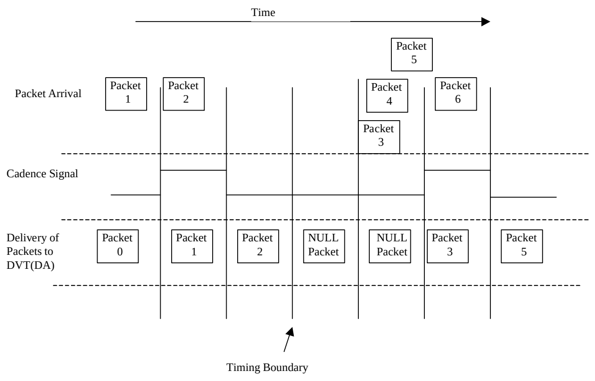

Figure 9 - The effect of extremely late packets on the device

This diagram shows the effect of multiple packets arriving extremely late. Ethernet is temporally constant so packets will arrive in the order that they are sent (in all but the most contrived situations). The order of the packets will always be increasing in sequence number (though packet loss may cause uneven increments). In the above situation, packets 3 and 4 have been delayed by the ethernet until the fifth packet's "timeslot" and then they all burst in. As soon as packet 3 arrives it is written down the line and then packet 4 is buffered awaiting the cadence signal change in level. To ensure that the end-to-end delay does not become excessive, the uCsimm buffer is limited to one packet. When packet 5 arrives it replaces packet 4 in the buffer. This means that the packet 4 data has been dropped from the system and replaced by a null packet sent earlier by the PIC.

This effect may occur when the ethernet network is under a very high utilisation (i.e. when a file transfer occurs).

Appendix B: uCSimm Pseudo Code

```
while (forever) {
   wait for ethernet port or serial port to have data;
   if( ethernet port ready) {
       read packet from port;
       if ( value from PIC heartbeat different from last time) {
           if ( no currently buffered packet) {
               write packet out of serial port;
           } else {
               write buffered packet out of serial port;
               copy current packet to buffer;
           }
       } else {
           copy packet to buffer;
       }
   } // end of if(ethernet port ready)

   if ( buffered packet AND value from PIC heartbeat different
from last time) {
       write out buffered packet to serial port;
   }

   if ( serial port ready) {
       read bytes available from serial port to incoming buffer;
   }

   if ( packet worth of bytes in serial buffer) {
       if ( buffer too full) {
           drop packet worth of bytes;
       }
       create ethernet packet with buffer;
       send packet;
   }
} // end of while(forever)
```

Pseudo Code 2 - The main loop for the uCSimm logic