Device Discovery and Configuration

Ashley Butterworth
Apple Inc.

Device Discovery

- multicast DNS based (Bonjour)
- publish _1394ta-fcp._udp service with TXT record
  - UDP port used for transmitting/receiving FCPDUs
  - TXT Record: txt version, device EUI64, fcp version, unit specifier id and unit version, vendor ID, vendor name, model ID and model name

TXT Record Format

- **General Format Rules for DNS TXT Records**
  - A DNS TXT record can be up to 65535 (0xFFFF) bytes long. The total length is indicated by the length given in the resource record header in the DNS message. There is no way to tell directly from the data alone how long it is (e.g. there is no length count at the start, or terminating NULL byte at the end).
  - The format of the data within a DNS TXT record is one or more strings, packed together in memory without any intervening gaps or padding bytes for word alignment. The format of each constituent string within the DNS TXT record is a single length byte, followed by 0-255 bytes of text data. These format rules are defined in Section 3.3.14 of RFC 1035, and are not specific to DNS-SD. DNS-SD simply specifies a usage convention for what data should be stored in those constituent strings.
- **DNS TXT Record Format Rules for use in DNS-SD**
  - DNS-SD uses DNS TXT records to store arbitrary key/value pairs conveying additional information about the named service. Each key/value pair is encoded as its own constituent string within the DNS TXT record, in the form "key=value". Everything up to the first '=' character is the key. Everything after the first '=' character to the end of the string (including subsequent '=' characters, if any) is the value.

TXT Record Fields for FCP-UDP

- TXT version
  - key “txtvers”
  - value “1”
- FCP version
  - key “fcp_version”
  - value “1”
- EUI64 generated from MAC address
  - key “eui64”
  - Value follows IEEE MAC to EUI64 expansion rule
    - First 3 bytes of MAC, 0xff, 0xfe, last 3 bytes of MAC

TXT Record

- unit_specifier_ID (hex string with no 0x)
  - key “unit_specifier_ID”
  - value “00a02d”
- unit_version (hex string with no 0x)
  - key “unit_version”
  - value “010001” for AVC
  - value “000102” for IIDC

TXT Record

- Vendor ID
  - key “vendor_id”
  - value your 24-bit OUI as hex string (without 0x)
- Vendor Name
  - key “vendor_descr”
  - value your company name

TXT Record

- Model ID
  - key “model_id”
  - value 24-bit integer uniquely identifying your product
- Model Nam
  - key “model_descr”
  - value the name of your device

“txt” Source code

```
#define kFCPIPTextRecordKey_EUI64 "eui64"
#define kFCPIPTextRecordKey_FCPVersion "fcp_version"
#define kFCPIPTextRecordKey_UnitSpecID "unit_specifier_ID"
#define kFCPIPTextRecordKey_UnitVersion "unit_version"
#define kFCPIPTextRecordKey_TextVersion "txtvers"
#define kFCPIPTextRecordKey_VendorID "vendor_id"
#define kFCPIPTextRecordKey_ModelID "model_id"

char myTextRecordBuf[1024];
char intString[8];

TXTRecordCreate ( &mLocalTextRecordRef, 1024, myTextRecordBuf);

dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_EUI64, 16, (void*)eui64 );

if ( dnsReturnValue == kDNSServiceErr_NoError )
{
     dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_FCPVersion, 1, "1" );
     if ( dnsReturnValue == kDNSServiceErr_NoError )
     {
            dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_UnitSpecID, 6, "00a02d");
            if ( dnsReturnValue == kDNSServiceErr_NoError )
            {
                  snprintf(intString, 8, "%06x", unitVersion);
                  dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_UnitVersion, 6, intString);
                  if ( dnsReturnValue == kDNSServiceErr_NoError )
                  {
                        dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_TextVersion, 1, "1" );
                        if ( dnsReturnValue == kDNSServiceErr_NoError )
                        {
                              snprintf(intString, 8, "%06x", vendorID);
                              dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_VendorID, 6, intString);
                              if ( dnsReturnValue == kDNSServiceErr_NoError )
                              {
                                    dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_VendorDescription,
                                                                         strlen(pVendorNameString)+1, pVendorNameString );
                                    if ( dnsReturnValue == kDNSServiceErr_NoError )
                                    {
                                          snprintf(intString, 8, "%06x", modelID);
                                          dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_ModelID, 6, intString);
                                          if ( dnsReturnValue == kDNSServiceErr_NoError )
                                          {
                                                dnsReturnValue = TXTRecordSetValue ( &mLocalTextRecordRef, kFCPIPTextRecordKey_ModelDescription,
                                                                                     strlen(pModelNameString)+1, pModelNameString );
                                          }
                                    }
                              }
                        }
                  }
            }
     }
}
```

FCPDU

- UDP/IP based transport
- transaction based
- 4 message types we are concerned with
  - 0: FCP Command frame (AVC)
  - 1: FCP Response frame (AVC)
  - 2: Write to Address
  - 3: Read from Address

FCPDU frame

- UDP Packet Payload

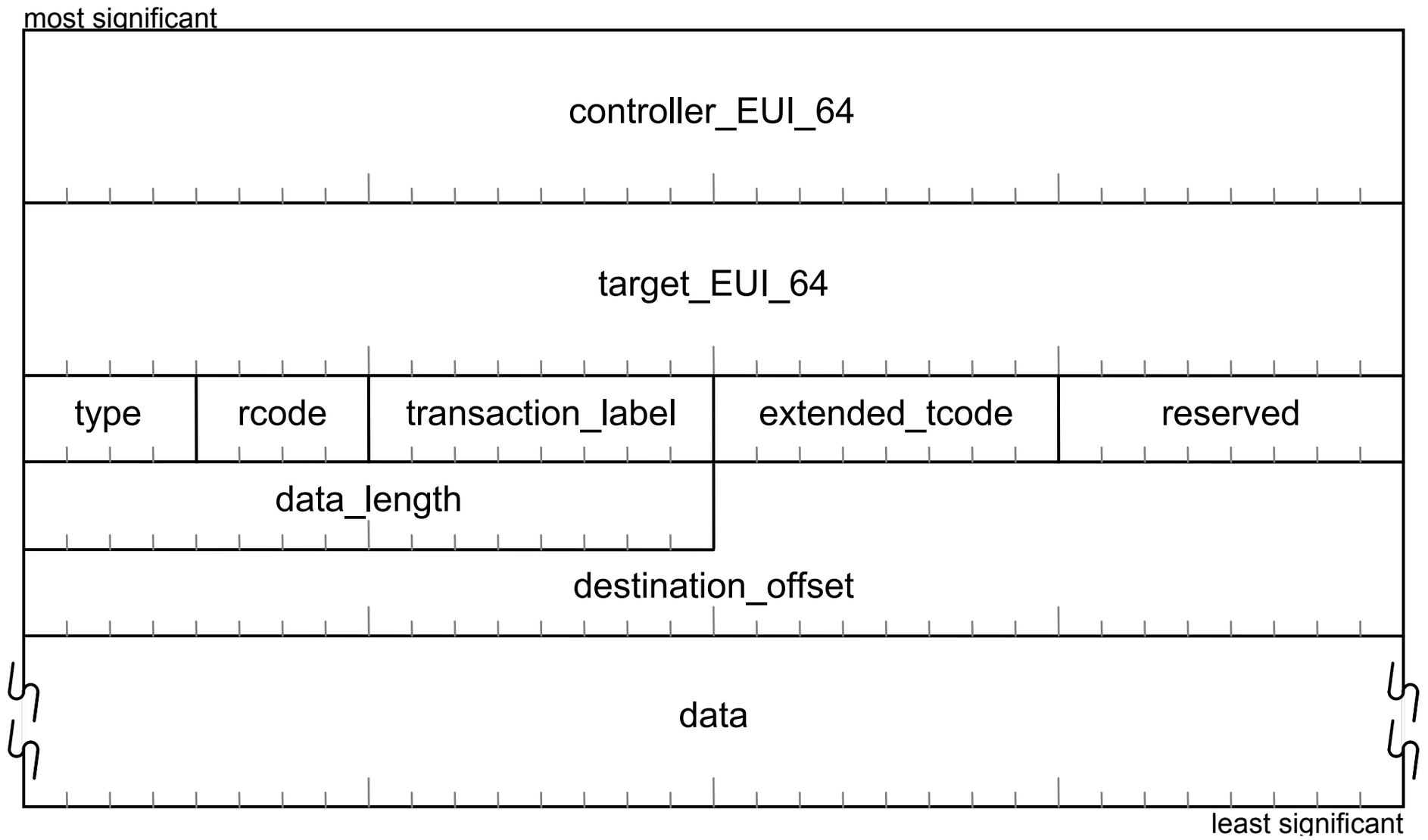


Figure 8 – FCP/IP protocol data unit format

p18 IEEE1394TA TS2006021 Draft 1.0

FCPDU - Messages

- Type 0
  - AVC Command
  - destination_offset is ignored
  - data is AVC command - max 512 bytes
- Type 1
  - AVC Response
  - data is AVC response - usually same size as command, max 512 bytes

FCPDU - Messages

- Type 2
  - Memory read
  - destination_offset is address to read
  - device reads memory into data
- Type 3
  - Memory write
  - destination_offset is address to write
  - device writes contents of data to memory

AVC

- Established standard for 1394
  - Define device capabilities and controls
  - Device is a unit with one or more functional subunits
  - Standard subunits for many device features
    - Audio/Music
    - Cameras
    - DVRs
    - and so on

AVC Device Info

- Unit has identifier descriptor
  - device name
  - info blocks for plugs to the outside world
    - “isoch” plugs for transport stream
    - external plugs for analog/digital interfaces
    - info block provides name, direction, type and ID

AVC Subunits

- Can have 7 instances of each subunit type per device
- Can have multiple subunit types per device
- Each subunit type has own specification expanding on general AVC specification

AVC Audio Device

- Base on Music subunit v2.0
- Music subunit encapsulates codec converting transport stream to/from audio samples
- Music subunit v2.0 provides function blocks and controls for manipulating audio
  - Volume, muxing, DSP, etc.
- Legacy 1394 devices use Audio and Music subunits

AVC Connection Management

- 1394 uses Plug Control Register
- Apple currently uses vendor unique AVC commands to make and break connections
  - two make commands (input and output)
  - one break command (uses reference returned by make command)

AVC Commands and Responses

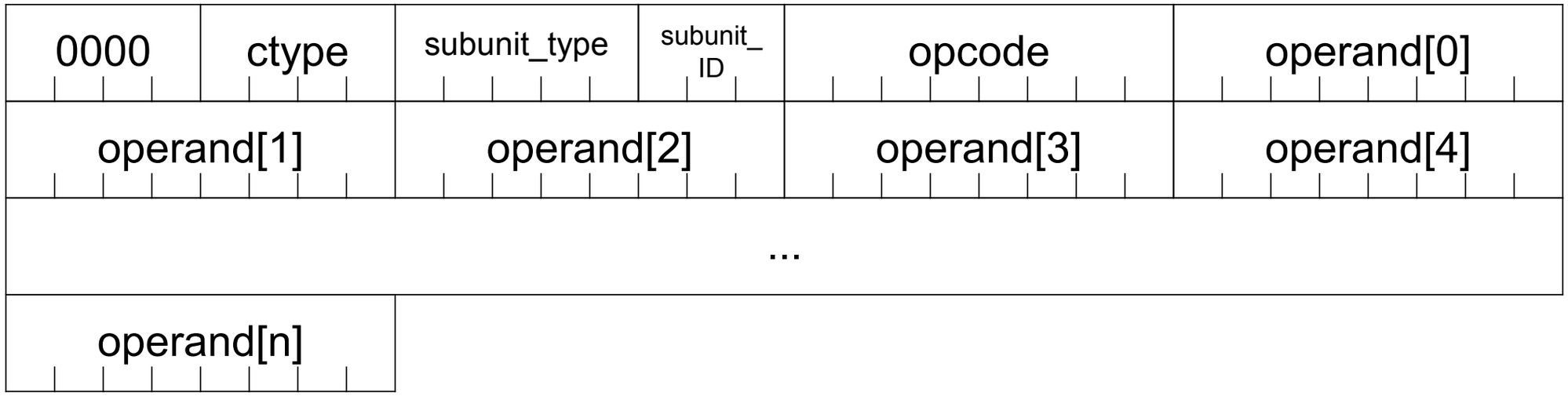
- AVC Command/Response is carried in data of FCPDU
  - Maximum length of 512 bytes
  - Commands and Responses usually the same size
    - Command is padded out with 0xff to be the size of response
- Addressed directly to subunit or the unit

AVC Commands

- Command is one of
  - Control
  - Status
  - Specific Inquiry
  - Notify
  - General Inquiry

AVC Command Format

Transmitted First



Transmitted Last

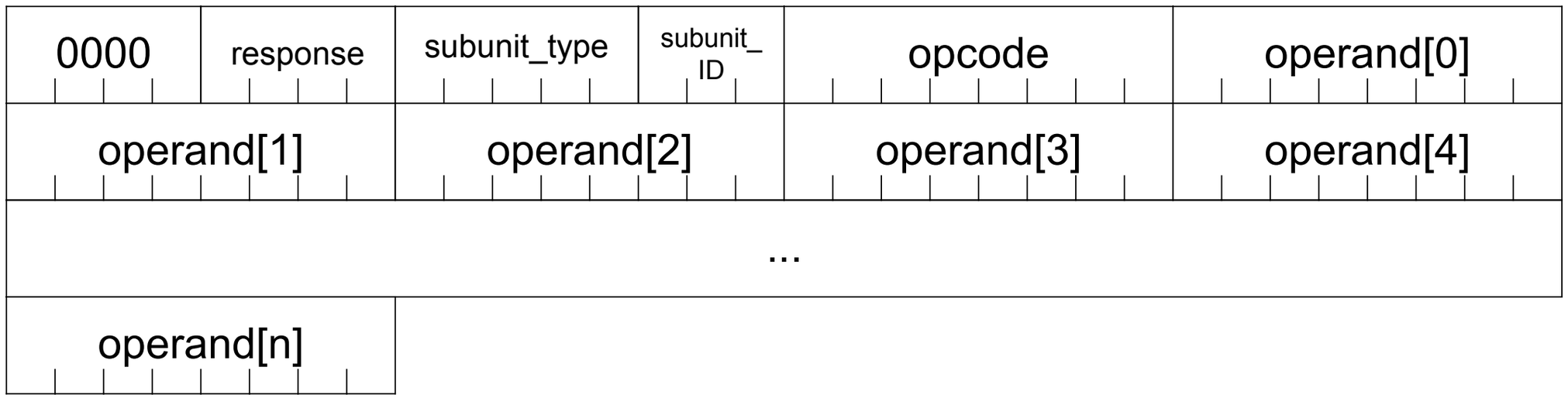
p12 IEEE1394TA 2007013 4.3 Revision 0.2

AVC Responses

- Response is one of
  - Not Implemented
  - Accepted
  - Rejected
  - In Transition
  - Implemented/Stable
  - Changed
  - Interim

AVC Response Format

Transmitted First



Transmitted Last

p12 IEEE1394TA 2007013 4.3 Revision 0.2

IIDC

- IIDC in 1394 is register based
  - read and write 32 bit registers
- Reading and writing registers can be accomplished with FCPDU message types 2 and 3
  - Message type 2 with 4 byte data for register write
  - Message type 3 with 4 byte data for register read

IIDC Connection Management

- IIDC streaming on 1394 is controlled by 4 bit value in register
- Apple currently implements by adding 2 registers to status and control registers and using MSRP

Offset	Upper 16 Bits	Lower 16 bits
0x638	Stream Index	Destination MAC bytes 0 & 1
0x63c	Destination MAC bytes 2 & 3	Destination MAC bytes 4 & 5