AVB/AVBTP layering, management objects and data transfer processing study
Draft 0.01

Alan K. Bartky, Bartky Networks

alan@bartky.net

www.bartky.net

Notice of copyright release

- **Notice:**
  - This document has been prepared to assist the work of the IEEE 1722 and IEEE 802 Working Groups. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.
- **Copyright Release to IEEE:**
  - The contributor grants a free, irrevocable license to the IEEE to incorporate material contained in this contribution, and any modifications thereof, in the creation of an IEEE Standards publication; to copyright in the IEEE's name any IEEE Standards publication even though it may include portions of this contribution; and at the IEEE's sole discretion to permit others to reproduce in whole or in part the resulting IEEE Standards publication. The contributor also acknowledges and accepts that this contribution may be made public by the IEEE 1722 Working Group or the IEEE 802 Working Group.

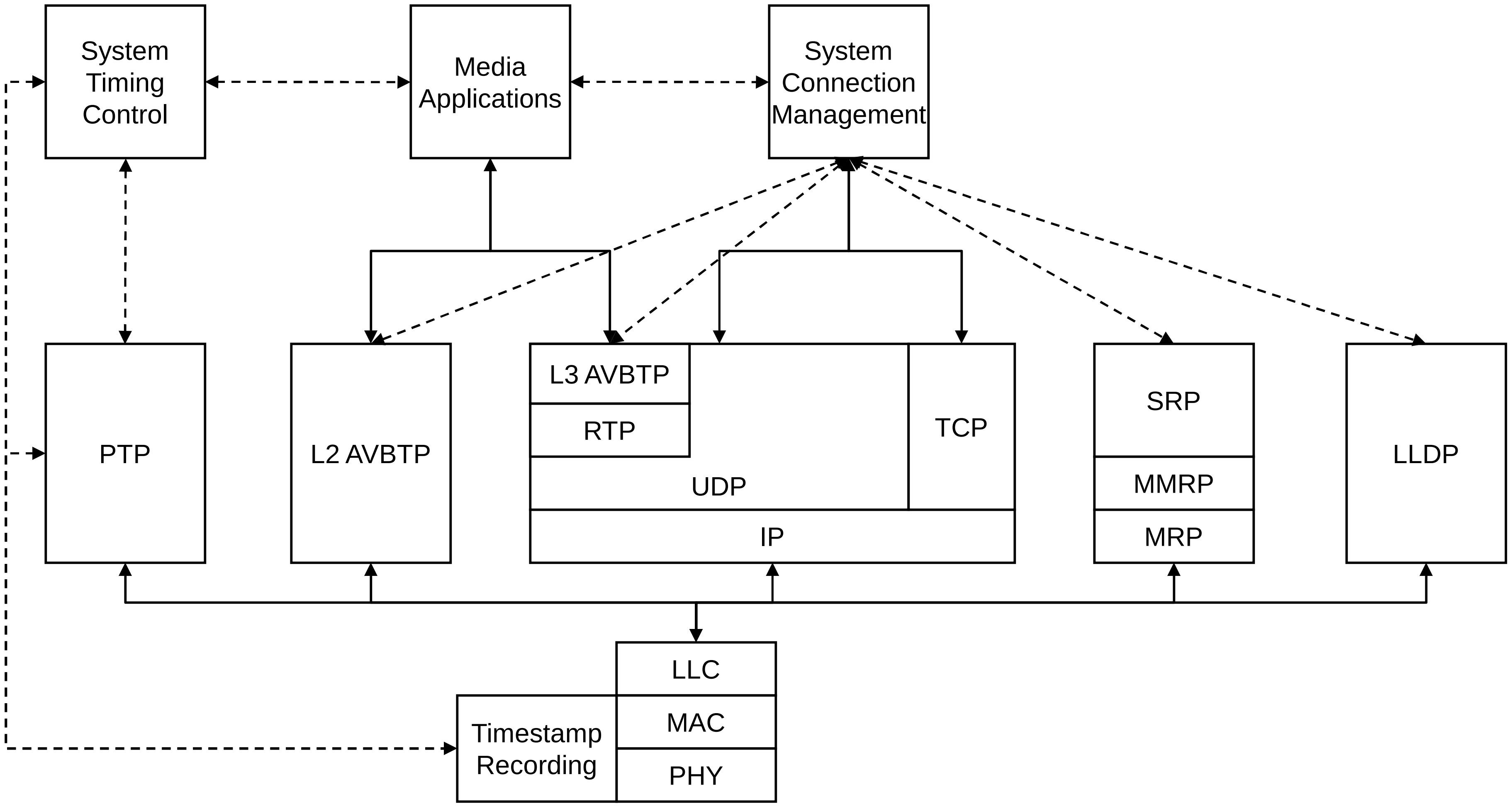
Revision History

Rev	Date	Comments
0.00	2007-04-27	First version for comments
0.01	2007-05-11	Added comments from AVBTP call and also new thoughts on service interfaces and functions.

Overview

- Disclaimer: “brainstorming” mode.
- Goal for this presentation is to model at a high level layering and stream data transfer processing between AVB/AVBTP layers and to work out and possibly help verify layer responsibilities and interaction between layers.
  - Focus on end station, but hopefully verify assumptions of operations of the bridge.
  - Hopefully help verify operation of service interfaces, state machines and layer responsibilities
  - Look into possible management objects both those needed by applications and others for system operation or debugging.
  - Presentation intentionally has lots of options, need to start removing options, but goal is to discuss them and get consensus on removing them.

AV end station layering



Timer Control module

- For discussion purposes, this assumes a single timer module available for all parts of the system for both hardware and software timing purposes. Discussion also assumes that most modules interested in time/timing/etc. are not interested in talking directly to PTP. Some high level hardware and software functions for the system timer function are:
  - Hardware:
    - Ability to tune frequency when running as slave or drive it as master.
    - Ability to provide output reference clock(s) to external HW devices (e.g. CODECs) based on 802.1AS clock.
    - Ability to trigger time stamp measurement by HW for later use to read by HW or SW.
  - Software
    - Ability to provide get and set of absolute time and provide relative time measurement and/or scheduling.
      - Should be able to supply POSIX functionality for time of day and timer services based on timespec or timespec like (seconds, nanoseconds) functions and calls.
      - Ability to read and correlate triggered events by hardware for software processing functions.

POSIX system timer examples

- The <*time.h*> header shall declare the structure **timespec**, which has at least the following members:

```
– time_t   tv_sec    // Seconds.
  long     tv_nsec   // Nanoseconds.
```

- Main functions to get and set timespec based time

```
– int clock_getres( clockid_t, struct timespec *);       // Get resolution of a system clock
  int clock_gettime(clockid_t, struct timespec *);       // Get time of a system clock
  int clock_settime(clockid_t, const struct timespec *); // Set time of a system clock
```

- ***SEE ALSO***
  - *<signal.h>*, *<sys/types.h>*, the System Interfaces volume of IEEE Std 1003.1-2001, *asctime()*, *clock()*, *clock_getcpuclockid()*, *clock_getres()*, *clock_nanosleep()*, *ctime()*, *difftime()*, *getdate()*, *gmtime()*, *localtime()*, *mktime()*, *nanosleep()*, *strftime()*, *strptime()*, *sysconf()*, *time()*, *timer_create()*, *timer_delete()*, *timer_getoverrun()*, *tzname*, *tzset()*, *utime()*

- Excerpts above pasted and edited from <http://www.opengroup.org/onlinepubs/009695399/basedefs/time.h.html>

- Note 1: As defined today, time_t is always 64 bits in 64 bit systems and usually 32 bits in 32 bit systems. There are various proposals on how to handle time in 32 bit systems to try and balance between the legacy codebases out there and the ability to handle dates greater than the rollover value of seconds since Jan 1 1970 00:00:00 epoch.
- Note 2: POSIX does allow multiple clocks in a system. This presentation will stick to one.

1588 objects of possible use

- >> Editor's note: I did a full review of the management objects as defined in the latest draft of IEEE 1588 v2. Here is what I believe may be of use to applications in an end station or bridge.
- General (bridges and end stations)
  - **clock_accuracy -- Integer**
    - **Editor's note>> Here is a case where current method used of reporting accuracy as ranges, does not meet current POSIX mechanisms. For service primitive for the system clock IMHO it should present in the same manner as POSIX time primitives clock accuracy where it is reported in nanoseconds.**
  - **time_traceable -- Boolean**
    - The value is TRUE if the timescale and the value of current_utc_offset are traceable to a primary standard; otherwise the value shall be FALSE.
  - **frequency_traceable -- Boolean**
    - The value is TRUE if the frequency determining the timescale is traceable to a primary standard; otherwise the value shall be FALSE.
- Transparent Clock (applications running in bridges)
  - **syntonized**
    - The value shall be TRUE if the clock is syntonized to a master clock of the primary syntonization domain, see 10.1, and FALSE otherwise. The initialization value shall be FALSE.

SNMP like sysClock objects

- >> Editor’s note: this is how I would define SNMP like objects for management of a system clock. Data below is broken up into what is useful for an application and others of use by system administration or debug.
  - Typical values for Admin (Administrative) Status (from ifAdminStatus, RFC 2863):
    - up(1), down(2), testing(3)
  - Typical values for Oper (Operational) Status (from ifOperStatus, RFC 2863):
    - up(1), down(2), testing(3), unknown(4), dormant(5), notPresent(6), lowerLayerDown(7)

- **Visible by applications:**
  - **sysClockValue --** TimeSpec64
    - POSIX formatted clock value in 64 bit seconds (0 through 0xFF-FF-FF-FF-FF-FF-FF-FF) and 32 bit nanoseconds (0-999,999,999)
  - **sysClockOperStatus --** Integer
    - Operational status of system Clock
  - **sysClockSource --** Integer
    - Internal, External (other than network), Network (IEEE 1588 or 802.1AS)
  - **sysClockAccuracy** -- Integer32
    - Accuracy of 802.1AS global clock in nanoseconds (useable by POSIX) clock accuracy function.
  - **sysClockLocalOutputClockSourceFrequency --** Integer64
    - Frequency of local clock driven by 802.1AS global clock available to applications and/or hardware . If multiple frequencies available, then max value shown.
  - **sysClockUTCOffset --** Integer
    - Seconds offset from UTC for calculating time from SystemClock
  - **sysClockUTCTraceable** -- Boolean
    - Indicates if traceable to a UTC clock whereby actual UTC time of day is traceable to a recognized UTC time provider.

- **Other management/debug objects (probably not needed by applications)**
  - **sysClockAdminStatus --** Integer
    - Administrative Status of System Clock

SNMP like PTP objects

- >> Editor's note: these are some additional objects not defined in 802.1AS or 1588 that I would recommend supporting. Note: assuming a system clock module providing services to applications, none of these objects would be needed IMHO by applications but would be useful for the system clock module and for PTP administration/debug. Also the objects below assume an SNMP table where it is one PTP instance per port. is how I would define SNMP like objects for management of a system clock. Data below is broken up into what is useful for an application and others of use by system administration or debug.

- **Other PTP management/debug objects (probably not needed by applications)**
  - **ptpProtocolAdminStatus** -- Integer
    - Administrative Status of the PTP protocol
  - **ptpProtocolOperStatus** -- Integer
    - Operational Status of the PTP protocol
  - **ptpPortOperStatus** -- Integer
    - Oper Status of the PTP port
  - **ptpPortAdminStatus** -- Integer
    - Admin Status of the PTP Port
  - **ptpLowerInterface** – ifIndex (note- possibly use ifStack table instead??)
    - Lower level interface associated with this PTP protocol entity (in SNMP, this would normally be the unique ifIndex of the Ethernet or Wireless port).

AVBTP stream objects

- >> Editor’s note: these are some early draft proposal for objects owned by and/or used by the AVBTP layer and/or the stream management layer. To have a starting point, this only focuses on layer 2 AVBTP and not layer 3 and also assumes a flow specification using parameters such as used by a Single Rate Tri Color Marker (srTCM, see http://www.ietf.org/rfc/rfc2697.txt) algorithm for policing which could be hopefully also be used to have a simply defined method for defining parameters for egress shaping as well.
- Stream Control Objects:

```
–   streamMacAddress      Physaddress,
–   streamRemoteAddress   Physaddress,
–   streamLowerInterface  ifIndex,   -- (use ifStack table instead?)
–   streamAdminStatus     INTEGER,   -- (enabled, disabled)
–   streamOperStatus      INTEGER,   -- (joining, active, leaving, inactive)
–   streamRate            INTEGER,   -- (octets per second)
–   streamCommittedBurst  INTEGER,   -- (Bc, octets)
–   streamExcessBurst     INTEGER,   -- (Be, octets)
–   streamMode            INTEGER,   -- (61883/1394, proprietary, other??)
–   streamOUI             OCTET STRING -- (3 byte field indicating 0080C2 for IEEE,
                                        -- or other value for proprietary streams).
–   streamMaxLatency      INTEGER,   -- Stream max latency in microseconds
–   streamMinLatency      INTEGER,   -- Stream min latency in microseconds
```

- Per Stream data statistics
  - In/Out Octets and Packets for Unicast, Multicast, Stream Data, Stream control (like ifTable counters, but on a per stream basis)
  - Packets discarded due to, protocol error, sequence error (perhaps just reduce to something like ifTable ifInErrors and ifOutErrors??)

SNMP “Interface” tables

- >> Editor’s note: AVBTP or other layers could present themselves as logical interfaces and represent themselves using ifTable, ifXTable, ifStackTable and/or RcvAddressTable (from http://www.ietf.org/rfc/rfc2863.txt?number=2863). Data below for reference:
-
```
IfStackEntry ::=
 SEQUENCE {
     ifStackHigherLayer  InterfaceIndexOrZero,
     ifStackLowerLayer   InterfaceIndexOrZero,
     ifStackStatus       RowStatus
  }
```
-
```
IfRcvAddressEntry ::=
 SEQUENCE {
     ifRcvAddressAddress   PhysAddress,
     ifRcvAddressStatus    RowStatus,
     ifRcvAddressType      INTEGER
 }
```

SNMP “Interface” tables

```
IfEntry ::=
SEQUENCE {
    ifIndex                 InterfaceIndex,
    ifDescr                 DisplayString,
    ifType                  IANAifType,
    ifMtu                   Integer32,
    ifSpeed                 Gauge32,
    ifPhysAddress           PhysAddress,
    ifAdminStatus           INTEGER,
    ifOperStatus            INTEGER,
    ifLastChange            TimeTicks,
    ifInOctets              Counter32,
    ifInUcastPkts           Counter32,
    ifInNUcastPkts          Counter32,  -- deprecated (MC & BC)
    ifInDiscards            Counter32,
    ifInErrors              Counter32,
    ifInUnknownProtos       Counter32,
    ifOutOctets             Counter32,
    ifOutUcastPkts          Counter32,
    ifOutNUcastPkts         Counter32,  -- deprecated (MC & BC)
    ifOutDiscards           Counter32,
    ifOutErrors             Counter32,
    ifOutQLen               Gauge32,    -- deprecated (gone)
    ifSpecific              OBJECT IDENTIFIER -- deprecated (gone)
}
```

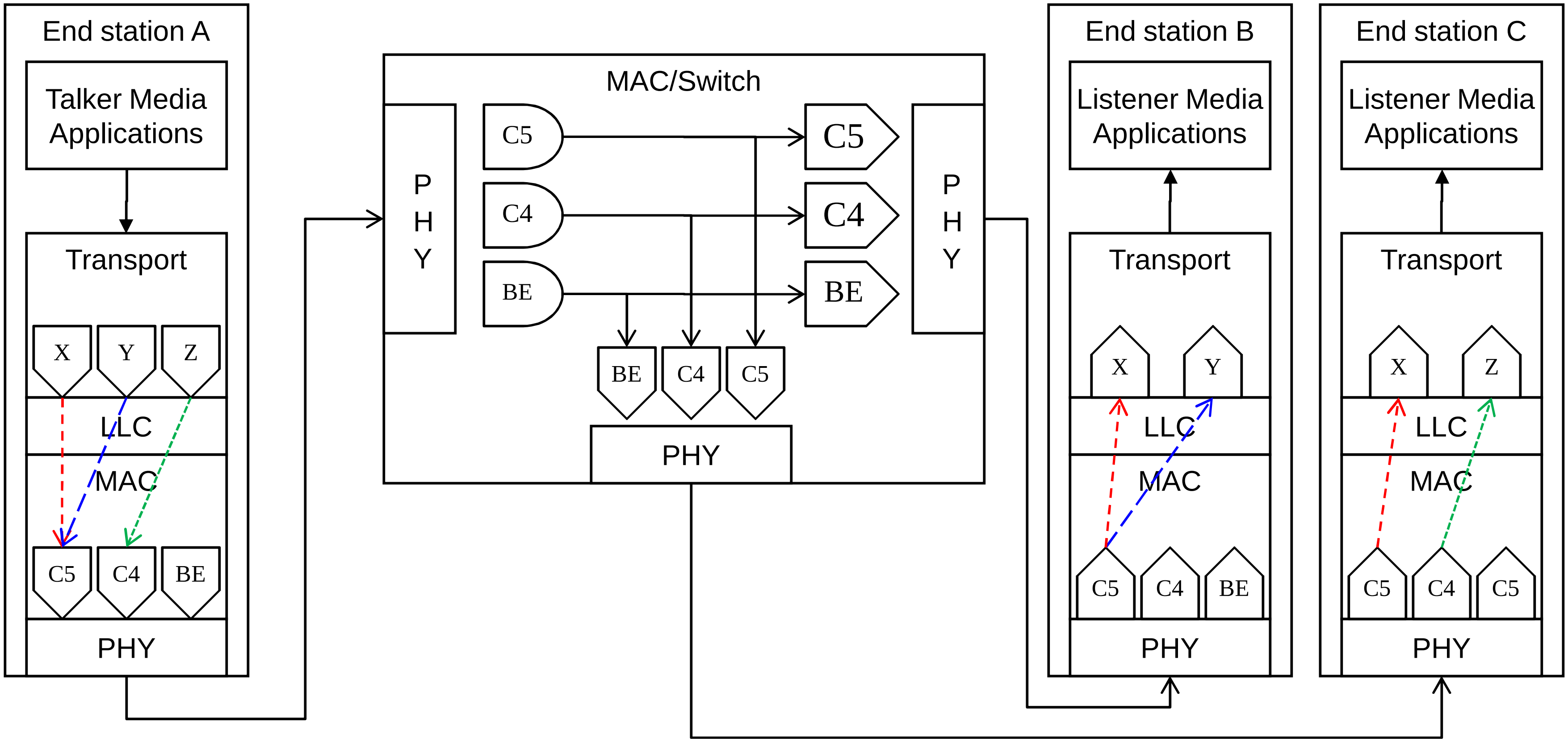
SNMP “Interface” tables

```
IfXEntry ::=
SEQUENCE {
    ifName                  DisplayString,
    ifInMulticastPkts       Counter32,
    ifInBroadcastPkts       Counter32,
    ifOutMulticastPkts      Counter32,
    ifOutBroadcastPkts      Counter32,
    ifHCInOctets            Counter64,
    ifHCInUcastPkts         Counter64,
    ifHCInMulticastPkts     Counter64,
    ifHCInBroadcastPkts     Counter64,
    ifHCOutOctets           Counter64,
    ifHCOutUcastPkts        Counter64,
    ifHCOutMulticastPkts    Counter64,
    ifHCOutBroadcastPkts    Counter64,
    ifLinkUpDownTrapEnable  INTEGER,
    ifHighSpeed             Gauge32,
    ifPromiscuousMode       TruthValue,
    ifConnectorPresent      TruthValue,
    ifAlias                 DisplayString,
    ifCounterDiscontinuityTime TimeStamp
}
```

Day in the life of an AVB stream, misc layer responsibilities, etc.

>> Editor's note: this has not been edited since version 0.00, waiting for some more consensus to emerge on where to queue and how to queue before coming back and taking a second look at this.

AV stream queuing/policing

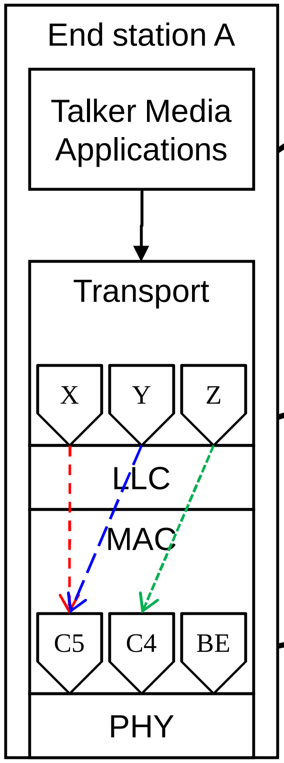


Key:
BE: Best Effort; C4: Class 4; C5 Class 5
X: Stream X; Y: Stream Y; Z: Stream Z

Police: ◗

Queue, Shape and/or Schedule: ▷

Talker Details



1) Interface between talker applications sends stream packets at a rate controlled by application, transport , some local clock frequency or the frequency from global clock.
2) Packets optionally associated with global time by application or transport, either:
    a) Application reference time
    b) Requested transport presentation time
    c) None

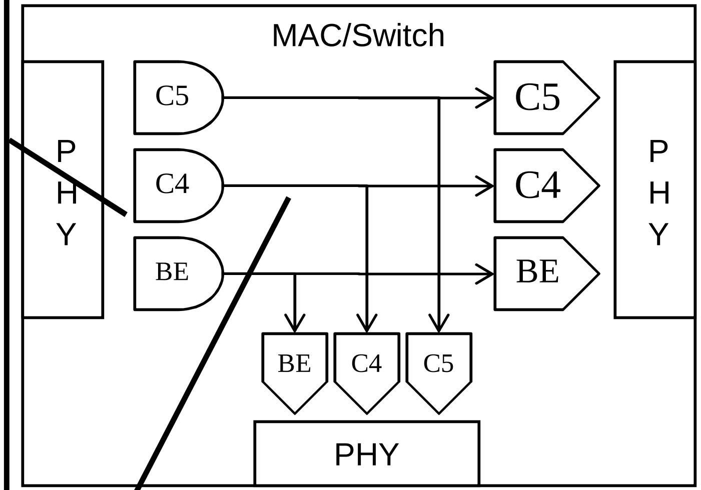
3) Transport shapes each stream on a per stream basis to requirements of 802.1Qav (if not already shaped from application that can meet Qav requirements).

4) MAC schedules class 5, class 4 and other non-AVB traffic (Best effort) for transmission to the PHY. Shaping is not required here as streams are already pre-shaped, so simple priority should suffice.

5) Stream frames exit talker end station meeting 802.1Qav requirements on a per stream and per class basis.

Switch Details

1) Frames enter switch from end station shaped on a per stream and per class basis, or from another bridge on a per class basis.
2) Frames are policed on a per class basis based on the aggregate values of all streams for a given class, for this example:
   - C5 = X + Y
   - C4 = Z



3) Data from streams not discarded due to policing are multicast to one of more ports of the switch to their respective class queues based on their destination MAC multicast group address.

4) Data is shaped on egress on a per class basis based on the on all streams for a given class, for this example
   1) Port to End station B:
      1) C5 = X+Y
      2) C4 = 0
   2) Port to End station B:
      1) C5 = X
      2) C4 = Z

5) Stream data exits switch shaped on a per class basis.

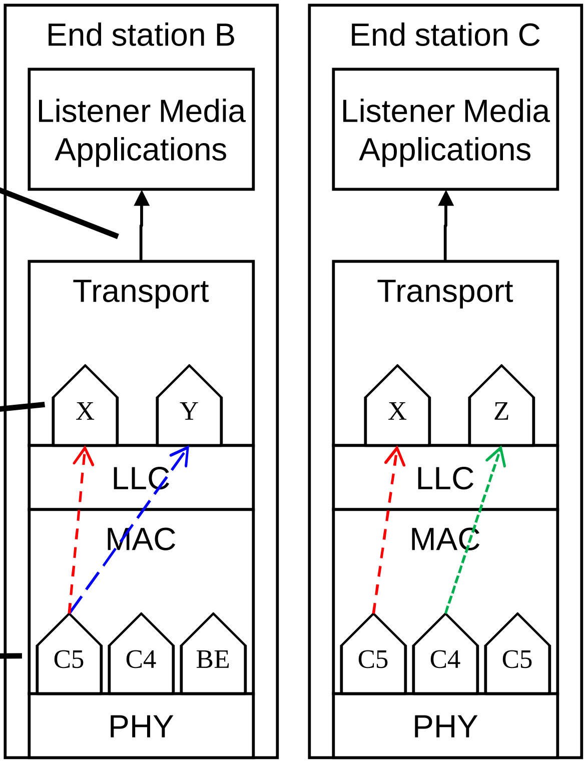
Talker Details

5) Transport sends stream packets at a rate controlled by application, transport, some local clock frequency or the frequency from global clock.
6) For each packet, any transport level associated time information is passed to the application for its information and possible use.

3) Transport optionally re-shapes each stream on a per stream basis to the characteristics of the original stream or to the requirements (timing/frequency) of the receiving device.
4) Transport optionally delays data in the queue for a given stream until ready to present to the application based on presentation time (i.e. option where transport is taking care of these delays rather than the application.

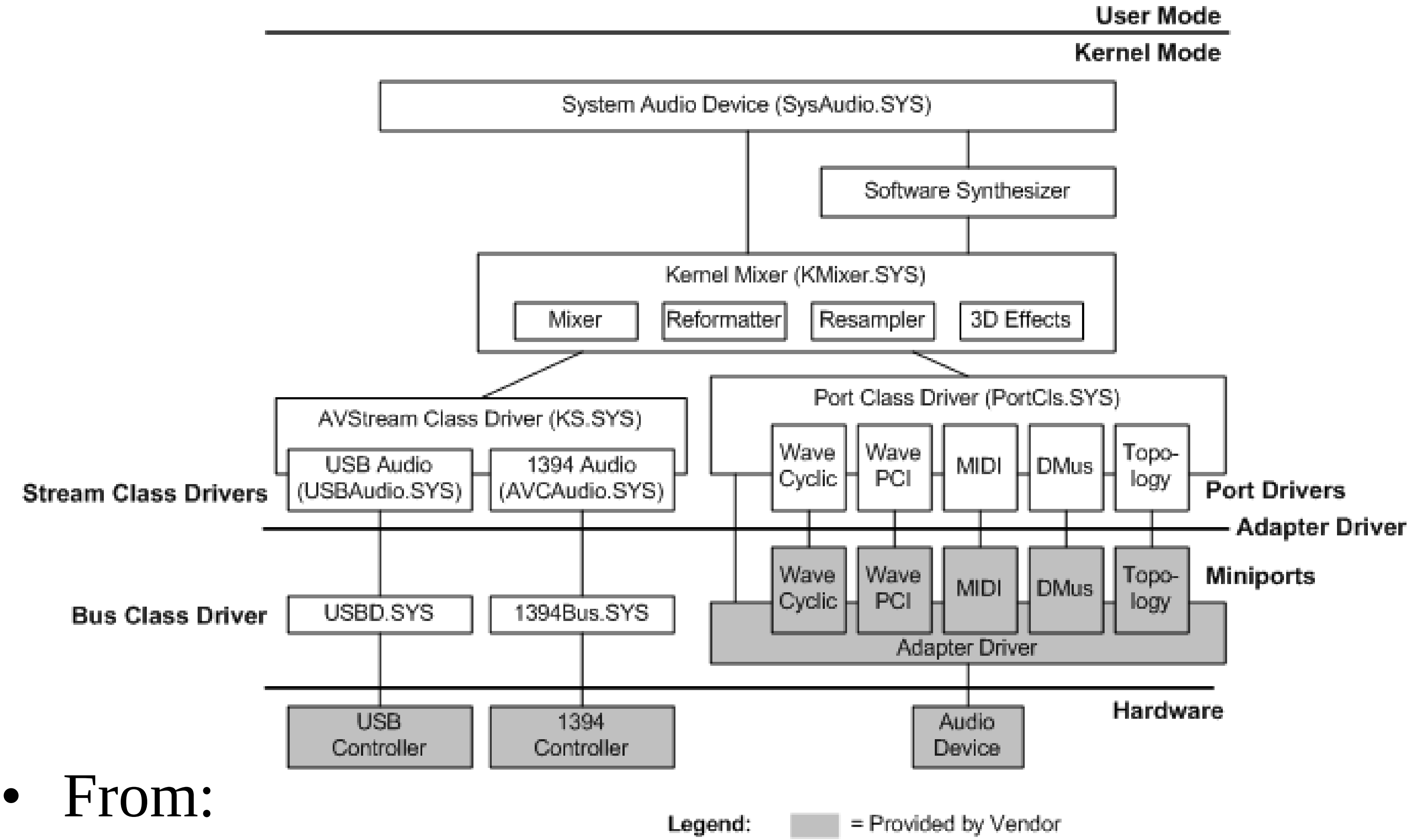
2) MAC Layer optionally prioritizes internal servicing of frames by upper layers by scheduling processing of streams with higher priority than best effort traffic.
>> Editor’s note: This is not part of the MAC reference model to my knowledge, but is supported by some MAC level devices.

1) Data enters end station shaped on a per class basis if attached to a switch, or per stream and class basis if attached to an end station
>> Editor’s Note: In theory, it has been discussed that as long as all end stations shape their streams properly and switches shape their classes, then the streams exiting the switch should be as well shaped as when they cam from the end station (can this be proved, and also can we trust all end stations to properly shape??).



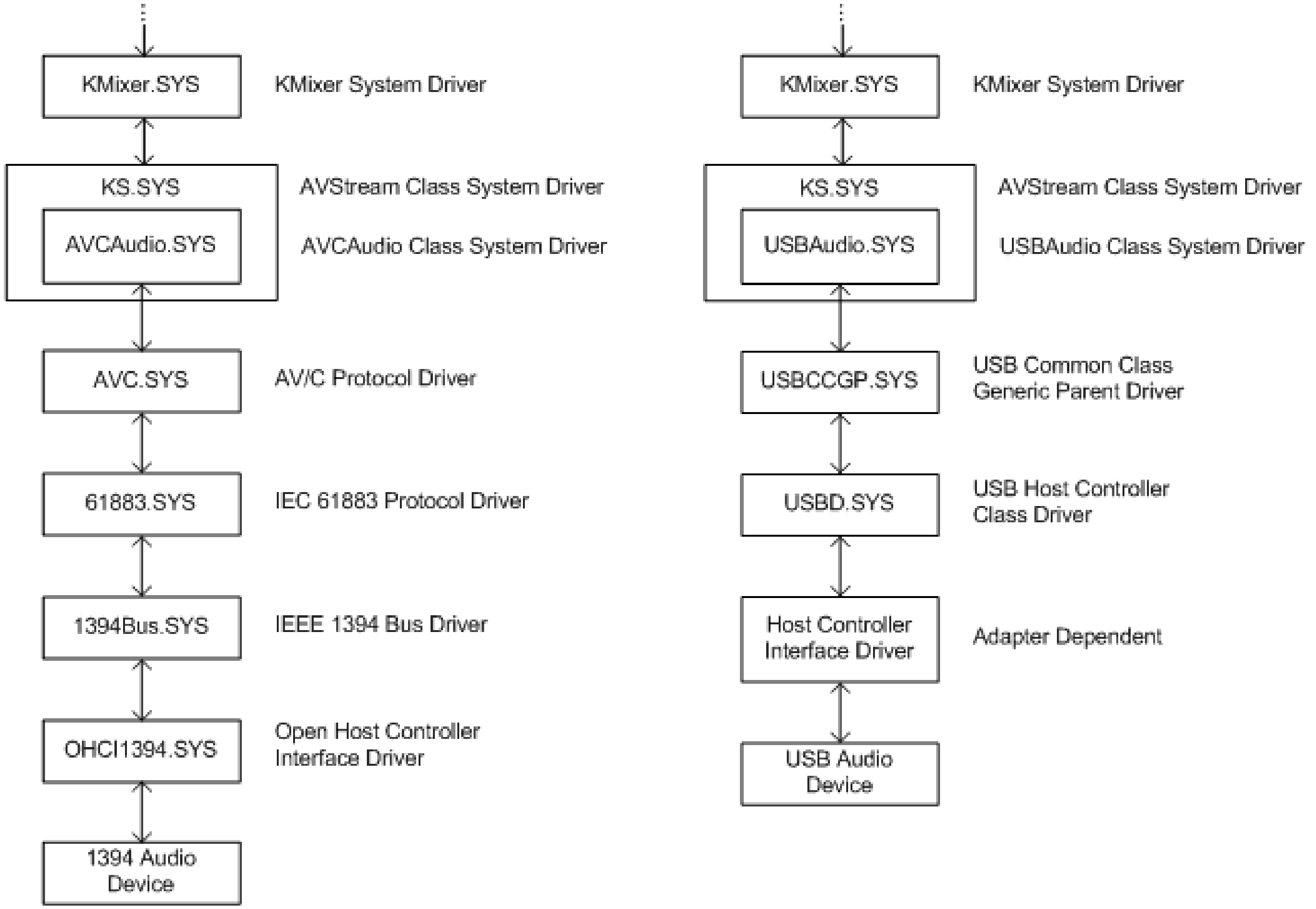
Backup

Microsoft Audio Layering



- From:
  - http://msdn2.microsoft.com/en-us/library/ms790577.aspx

Microsoft detailed AV/C & USB audio layering


KMixer.SYS
KMixer System Driver
KS.SYS
AVStream Class System Driver
AVCAudio.SYS
AVCAudio Class System Driver
AVC.SYS
AV/C Protocol Driver
61883.SYS
IEC 61883 Protocol Driver
1394Bus.SYS
IEEE 1394 Bus Driver
OHCI1394.SYS
Open Host Controller Interface Driver
1394 Audio Device
KMixer.SYS
KMixer System Driver
KS.SYS
AVStream Class System Driver
USBAudio.SYS
USBAudio Class System Driver
USBCCGP.SYS
USB Common Class Generic Parent Driver
USBD.SYS
USB Host Controller Class Driver
Host Controller Interface Driver
Adapter Dependent
USB Audio Device

- From: http://msdn2.microsoft.com/en-us/library/ms789375.aspx