APSD Test Method

Greg Chesson

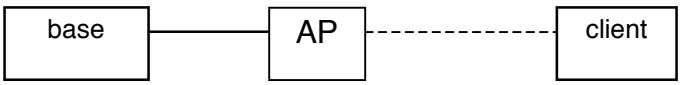
Atheros Communications

1. Introduction

The current lack of APSD-ready clients poses a problem for testing an APSD AP. This note describes a strategy that we have used to develop and test Unscheduled APSD, *aka* UAPSD or UPSD, in a development environment. The concepts may prove usable in a multi-vendor testing environment. This version of the document is not complete in any sense. It is meant as a starting point for discussion and elaboration.

2. Topology

The AP under test has one or more wireless clients (stations) plus at least one computer system (called the "base" station) connected to its Ethernet port. The test procedure is built around message exchanges between client station(s) and the base station.



In this scenario the client will generate uplink wireless traffic that forwards to the base station which then echoes the traffic back to the AP where it stays until released for downlink transmission by an uplink trigger frame from the client. Thus the timing of uplink and downlink transmission is controlled at the client – which is exactly what APSD is about.

More complex scenarios are possible. For example there could be an Ethernet on the client in addition to the wireless link. The Ethernet could be used as a control and reporting channel by a another computer, e.g. a Chariot console or other meta entity. However, the simpler approach involving a self-contained traffic generator at the client was chosen for simplicity and cost and the ability to quickly create dedicated and special-purpose tests.

3. Startup

Stations initially associate and authenticate with the AP. This is expected behavior and requires no new testing techniques except for the following controls needed at the client:

1. Select APSD association or normal association
2. Select which ACs are to be trigger/delivery enabled
3. Select entering power-save mode after association, or normal mode

We added controls to the wireless client driver for each of the selectable options. Note that option 3 is used to signal the AP that the device is in power-save mode even though it remains fully powered during the test.

This small set of controls is sufficient to place the station/AP combination into a testable mode using the association exchange as the APSD enabling mechanism. If TSPECs are to be tested, then the client station also needs a facility for generating the TSPEC exchange. Since there is no standard practice or API in current operating systems that would generate a TSPEC, we have used a dedicated application to bypass the wireless device driver at the client station and directly generate the management exchange.

The most simple and direct test profile is to select APSD association with AC_VO designated as the trigger and all ACs delivery-enabled. This configuration is expected to be common in APSD deployments, so is a useful starting place for testing.

The minimum functionality needed at the AP for testing is similar to that needed for WME whereby an operator can interrogate the AP to determine if the AP has recognized an APSD negotiation (or a TSPEC negotiation).

4. APSD Method

The concept is that we run a frame-generating program (*upsd*) on a client station that sends frames to a program (*ap*) running on the "base station" which is on the Ethernet-side of the AP. The *ap* program on the base station expects UDP frames on a certain port (port 12345) and simply echos any received packets back to their source. An important property of *ap* is that it preserves the dscp value of any received frames. Thus uplink frames containing the dscp value for voice (0xb8) will be recognized as downlink voice frames when they're echoed from the base station and transmitted over the Ethernet to the AP.

The client station in cooperation with the *ap* application simulates both the uplink and downlink components of a voice call. The traffic of multiple streams can be simulated by adjusting the packet generation rate at the *upsd* application or by running multiple instances of the client station and the *upsd* application.

The frame generation rate of *upsd* is controlled by specifying the interpacket interval. This can range from a few microseconds to many seconds and is useful both for generating a load for performance testing as well as generating very slow frame rates for debugging.

The *upsd* program generates frames that contain timestamps and sequence numbers. These are used to measure the turnaround time at the AP. The sequence of events begins with frame #0 uplink from the client station to the AP which then forwards frame #0 to the *ap* application. This frame is returned to the AP where it will be placed into an APSD power-save queue for the client station until an APSD trigger from that station is received. The trigger is supplied by the next uplink frame, #1, from the station. Thus frame N triggers frame N-1. The program can build a histogram of response times and

print the result after every 1000 frame exchanges. It also resynchronizes itself if a packet is lost and counts the losses.

Single-frame packet exchanges as outlined above are useful for checking APSD basic operation – one packet uplink and one packet downlink. Additional tests are needed to validate other APSD modes of operation. These may take the following forms:

1. Sending a mix of trigger and non-trigger uplink frames. This can be done by running both *upsd* and another application, e.g. *ftp*, on a client station.
2. Sending a mix of downlink traffic types in addition to AC_VO. This can be done by running any WME test suite between the base station and client station including Chariot as well as other test programs from which *upsd* evolved.
3. Selecting one or more ACs for PS-Poll-style legacy delivery while also running an APSD test.

The *upsd* and *ap* applications are Linux programs that depend only on standard socket APIs and the *gettimeofday()* call. It is a single thread of execution that does not depend on pthreads or multiple processes. It has been verified that most of the socket calls used by these programs work properly on MicroSoft operating systems that can run the Cygwyn programming environment. Some porting work would be required to make the programs useful under windows. It has been suggested that these programs could be loaded onto a Knoppix-like live CD so that they could run on an hw supported by a live distribution.

5. Measurement and Validation

The method described in the previous section is roughly equivalent to having a modified version of iperf or netperf running on the client and base stations. Traffic generation and measurement is the responsibility of the client. That places a validation burden on the correctness of the test program as ported to the client – not something we worry about when creating engineering testbench programs for ourselves, but definitely a consideration in choosing a methodology for multi-vendor interop tests.

There appears to be less risk – for development, lifetime support, and validation – to have the simplest possible client. That would be a client that only executes APSD negotiation handshakes and generates very basic traffic and triggering signals. Monitoring, measurement and compliance-checking would be done with a wireless packet sniffer.

There are several packet sniffers, particularly *tcpdump* and *ethereal*, that are in active use in the WiFi test environment. These sniffers usually are used by test operators who must personally inspect formatted trace files or formatted screen images. That is laborious work and error-prone as well. The data must be checked and often repeated before the results are considered valid. It would be far more efficient and reliable to operate these sniffers in a mode where they copy on-air traces to a capture file. Test operators would then apply analysis scripts that scan the trace files according to the test plan. We propose that the creation of a test plan would necessarily incorporate test scripts for each of the test items that would be accomplished in this manner.