AirO

The Air Observer from Aruba Networks

“If your app needs a guide, the User Interface must be pathetic” – anonymous app guru

AirO is intended for technical and not-too-technical owners of Wi-Fi capable Android devices. It displays the health of the Wi-Fi (“Local Area”) connection, and measures the characteristics of a “Wide Area” connection to a server deeper in the network. It can be used to answer questions like:

* What’s wrong with my Wi-Fi today?
* How strong is my Wi-Fi signal?
* Is there evidence of wireless interference?
* Is the problem in the Wi-Fi connection, or out on the Internet (or corporate network)?
* Is the overall connection to the data center good enough to run my corporate apps?

# “Wi-Fi and Local Area Network”

The top “Wi-Fi and Local Area Network” section of the screen displays three measurements that show the health of the Wi-Fi connection:

* Signal Strength. In Wi-Fi terms, this is received signal strength, or RSSI. It is measured in dBm and is always negative. A larger (negative) number indicates a weaker signal.
  + When right next to an access point, the figure might be -35 dBm
  + When standing underneath a ceiling-mounted access point, perhaps -40dBm
  + A ‘good’ signal is better than -68dBm
  + The signal will be problematic below -75dBm
  + A signal of -80 dBm is generally unreliable to unworkable

We measure signal strength first because if it’s poor, there is no chance of getting a good connection. The remedy, in simple terms, is to get closer to the access point.

* Link Speed. This is technically the modulation rate of the Wi-Fi link in Mbps.
  + A new 802.11ac device (Samsung Galaxy S4) might show 200+ Mbps maximum
  + An 802.11n device might reach 65 Mbps or 130 Mbps with 40 MHz channels
  + Under normal conditions a link faster than 20 Mbps is very good
  + Once a link drops below 5 Mbps it is likely to be unreliable

The usual cause of low link speeds is poor signal strength. But sometimes, even when signal strength is good, interference on the air from Wi-Fi and non-Wi-Fi sources reduces link speeds.

* Ping. This is the familiar ICMP echo test to the network’s default gateway, normally a router close behind the access point. It indicates the round-trip delay between sending a packet to this gateway and receiving a response.
  + Ping results vary quite widely
  + With good conditions, a stable delay of less than 20 msec is expected
  + If ping times are consistently over 100 msec, application performance will suffer
  + A ping time above 500 msec indicates something’s wrong

A low link speed will often cause long ping times. If link speeds are good but pings slow, it may be a long way to the default gateway over a narrow broadband connection.

On the right of the ‘Wi-Fi’ section is a small icon showing “Strongest AP”. In the Wi-Fi protocol, devices are responsible for choosing the access point they connect to, and sometimes when their owner moves, they remain ‘sticking’ to an old, and now distant access point. The left-hand bar of the icon shows the signal strength of the connected (“this”) access point. The right-hand bar shows the signal strength of the strongest signal the device (with the same SSID) showing on the device’s Wi-Fi scan. Under each is the signal strength value, in the same units as for the ‘signal strength’ reading. Because the scan is a different measurement, some variation is expected, but if ‘this AP’ is more than about 10dB weaker than ‘best AP’ it’s time to move closer to ‘this AP’ or shake the tree by putting the device into airplane mode and out again - one way to induce the device to move to the stronger access point. (Note: Some newer Android devices are very reluctant to scan, if both readings are at -99 but there’s a good connection, it is because the Wi-Fi interface is not returning scan information to the AirO app. Putting the device into/out of airplane mode, or toggling Wi-Fi off/on, should fix this for a while.)

The header line of “Wi-Fi and Local Area Network” displays the device’s IP address, the SSID of the current connection and the radio channel (channels < 15 are in the 2.4GHz band, > 30 are in the 5GHz band).

At the bottom of the ‘Wi-Fi and Local Area Network’ section is a horizontal bar graph. This displays an overall estimate of the health of the Wi-Fi network, from a proprietary formula by the CTO group. We are aware that this is a subjective figure, and the coloring may not always be consistent with the icons beside individual figures (e.g. the bar is yellow when there are at least two red icons). And because the figures bounce around, the bar is unsteady. There’s more to do here.

# “Wide Area Network”

The lower section of the screen displays results from tests between the device and a server computer, usually in the corporate data center or on the Internet. The address of this server is chosen from a number configured in ‘settings” – but once chosen, only one server address is used for these tests.

* Ping. There is a ping measurement to this server. It is the same ICMP ping test as above, but because this one goes farther it will normally (but by no means always) take longer. Again, 20msec would be fast and 500 msec would be slow.

Some networks may block ICMP (ping) traffic. In this case, the Wide Area Network ping test will always fail, but normal (e.g. Web) traffic may pass.

* Speedtest. The next tests are ‘speedtests’. For this, we use the iPerf function (iPerf v2). In a corporate context, this should be an iPerf server instance set up somewhere in the core of the network, probably a data center. Because it is a throughput test, the figures here will never be more than about 50% of the ‘link speed’ figure for the Wi-Fi connection. The iPerf client in the app is configured to run in bidirectional mode, first an upstream test then downstream.
  + In a corporate network, uplink and downlink speedtest should give a figure higher than 10M bps.
  + In a home network, the speed will usually be limited by the broadband connection rate. A figure of 1 to 10 Mbps can be expected. In home networks, or behind firewalls, the downstream test may fail to pass the firewall, but this does not indicate normal (e.g. Web) traffic will fail.
  + The icons to the left of the screen spin after an iPerf request has been sent, and before a response is received. This gives the user an idea of what’s going on because iPerf doesn’t give us continuous measurements.

Most apps should perform adequately even with network rates below 1 Mbps, but at some point below 0.3 Mbps, application response times will slow.

An appendix to this document contains more details on the iPerf tests and setting up an iPerf server.

The horizontal bar graph shows the overall health of the Wide Area connection, a combination of Ping and iPerf measurements.

# Blue buttons

The left-hand ‘email report’ button does just that – it generates a snapshot of Wi-Fi and phone conditions and emails it to <whoever you want>, usually a helpdesk address. The default address is in ‘settings’ but can be over-ridden when sending the email. The data is contained in a CSV formatted file that can be opened by Excel or another spreadsheet program, headers are included so it should be self-explanatory.

One virtue of the email report is that the snapshot can be taken when the device has no connectivity, saved as a draft and sent later, when conditions are better. We believe users are happier when they can do something to report trouble (we’ll see whether helpdesks agree).

The right-hand ‘diagnostic ideas’ button brings up a relatively simple set of conclusions drawn from the data displayed on the screen. Each diagnostic idea explains in simple terms what’s going wrong and how things might be improved. It will be obvious to the user that, particularly in bad conditions, readings jump around a lot, and it is difficult to get a consistent estimate of what’s wrong… we’ll be working to improve this function in the future.

# Settings

Settings are accessed in the usual way (varies with device and Android version but usually the “ : - symbol with three dots”).

* There are five available address fields for iPerf server addresses. IPv4 or FQDN (the app isn’t tested with IPv6) points to the host that is used for WAN Ping and Speedtest tests. When first launched, and every minute following, a test utility checks reachability to port 5001 (the iPerf default receive port) of each address in turn. The first one that passes the test is used, so the list is an ordered list… It is pre-populated with two values at the bottom, one for a public iPerf instance (54.201.159.31) and the last for a private address in the Aruba Sunnyvale network (10.6.100.216). When used in a corporate context, the addresses of one or more private iPerf servers should be entered in one of the upper address fields.
* iPerf test frequency is actually the repeat interval. The default is 20 sec. It is not reasonable to reduce this below about 10 sec, and it can be increased. If there is no response to iPerf requests, the app will timeout and re-start after 2x this interval. Results are staled out after 3x this interval.
* iPerf duration is the –t parameter in the iPerf command line. The default is 4 sec. As noted below, the default behavior on this app is

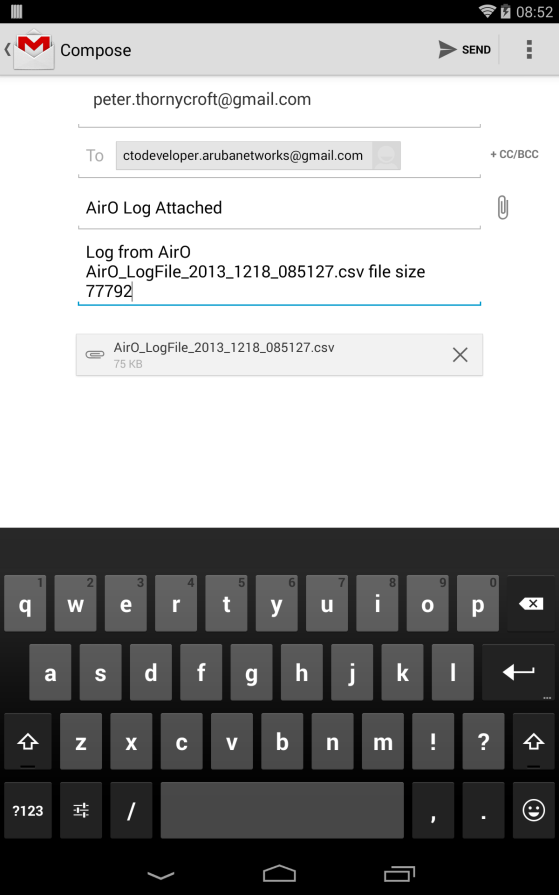
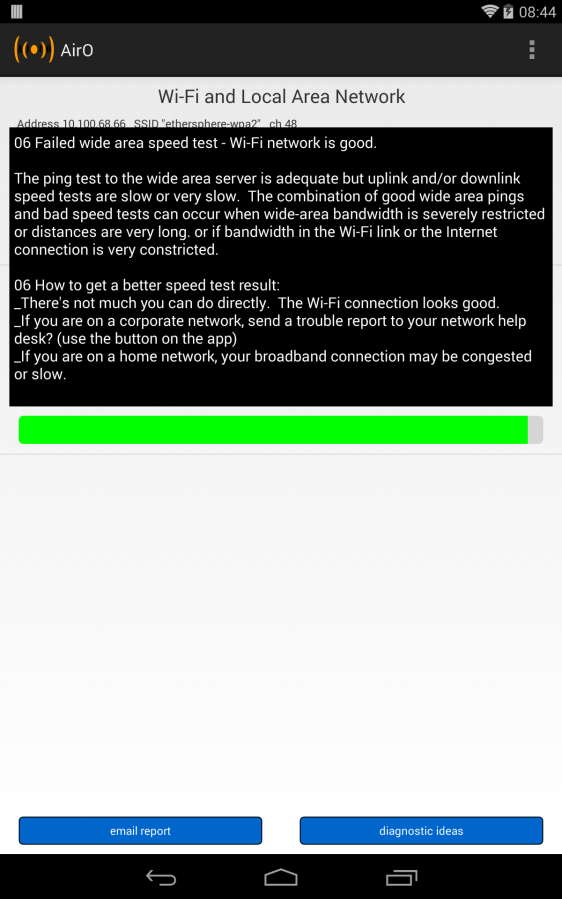
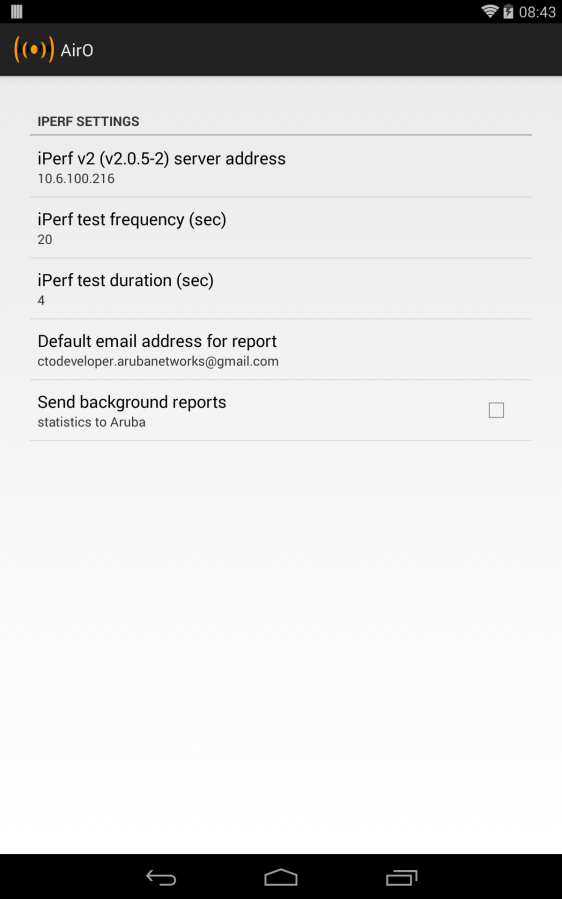
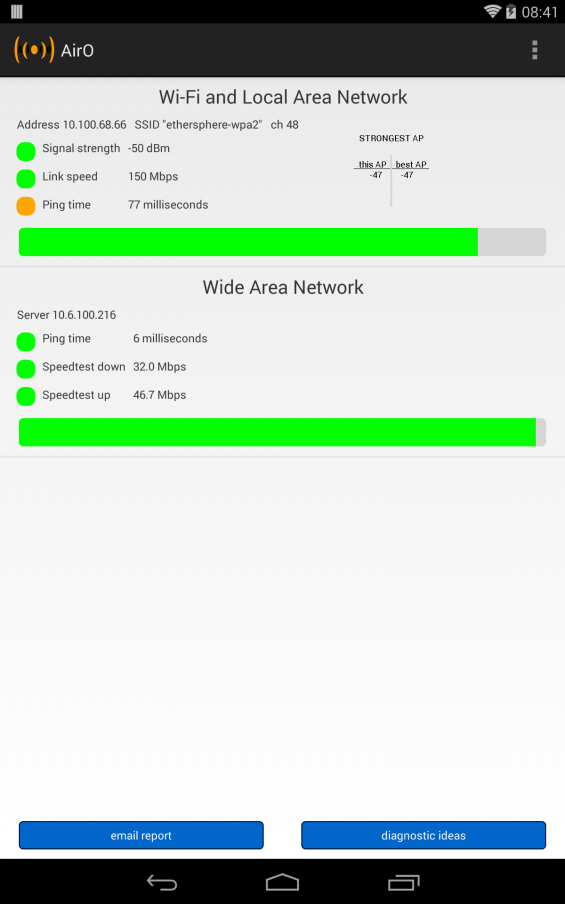
*iperf –c <server address> -r –i 2 –t <duration>*

With the default of 4 sec, this takes 4 sec for the upstream test then another 4 sec for downstream. Given some latency, the overall test will take 10-16 sec, which is why the default frequency is set to 20 sec. The idea of bursty, not continuous testing is to avoid congesting the network with test traffic.

This default command line can be over-ridden by the knowledgeable user (see appendix).

* Default email address for report. The ‘email report’ button triggers the device’s email functions to come to the foreground, pre-populated with the report attached, a title and this address as the addressee. This is just a time-saver, it can be over-ridden.
* ‘Send background reports’ is an opt-in option (default is off) to send periodic reports about wireless conditions to a server maintained by Aruba. When used by a significant percentage of a population over time, this can provide statistical insights into Wi-Fi patterns across a campus.

# Screenshots (AirO v4)



# Appendix: iPerf configuration

iPerf was chosen as the bandwidth test tool for this app because it is widely used and understood, and server code is available for all common platforms. But iPerf is not quite ideal for our purposes. This appendix covers some issues and workarounds.

The client-side implementation is iPerf 2.0.5-2 (see below for the -2 patch). The command syntax depends on settings for <server address>, the server address (either IP address or FQDN), and <duration>:

*iperf –c <server address> -r –i 2–t <duration>*

A special option allows the user to change iPerf options. If the ‘duration’ setting starts with a “-“, it will be interpreted as the command option string and added to the iperf exec verbatim. Thus, if ‘duration’ is “-p 5010 –i 5 –t 55” the command executed will run without the –r option as:

*iperf –c <server address> -p 5010 –i 5 –t 55*

In ‘normal’ –r mode, the app will run iPerf for <duration> seconds in upstream, then the server changes to client mode and runs again <duration> seconds in downstream mode. When it does this, it opens a new socket which can be blocked by client-side firewalls (the default receiving port is 5001). The downstream test will not complete under these circumstances, but a figure for upstream speedtest will be displayed.

Another issue is a bug in the most common version of iPerf, v2.0.5. The symptom is that when the server switches to client mode for the second half of a bidirectional test, it will exit at the end of the test. There is a patch for this bug. Compiled versions of 2.0.5-2 are available at [www.iperf.fr](http://www.iperf.fr), or this simple patch can be applied to the source:

In file Client.cpp

*void Client::Run( void ) {*

*char\* readAt = mBuf;*

*#if HAVE\_THREAD*

*+ /\**

*if ( !isUDP( mSettings ) ) {*

*RunTCP();*

*return;*

*}*

*+ \*/*

*#endif*

*// Indicates if the stream is readable*

In other words, comment out the code from “#if” to “#endif”.

Finally, iPerf servers have been known to crash from time to time. There is an available daemon mode which may be restarted automatically, but we have not tested it. It is probably prudent to write a script to spot when iPerf is not running, and restart it. Here is a simple Powershell example for Windows:

*# Starts num\_copies of iperf in server mode in this case, and*

*# periodically checks how many copies are still alive every num\_seconds, and*

*# restarts as needed to keep num\_copies active iperfs.*

*# This is necessary because if -r or -d are invoked from the client,*

*# when the server goes into client mode it can stop at the end of the test.*

*# Also, when multiple clients access the same iperf server simultaneously with the -r or -d options,*

*# the iperf instance can crash. On the server, this will popup the Windows Error Report window*

*# which will sit waiting for keyboard input and hang the crashed program. So disable WER by going*

*# to local group policy editor (run > edit group policy > administrative templates > windows components*

*# > windows error reporting... there are several options, we had success with 'display error notification*

*# disabled', 'disable WER enabled', 'prevent display... enabled'.*

*# You can monitor with task manager > processes while one or two clients are running in -r or -d mode*

*# to see iperfs closing and being restarted. There should always be num\_copies active.*

*$iperfargs = "-s"*

*$num\_copies = 1*

*$num\_seconds = 10*

*$b = 0*

*Start-Process -windowstyle Normal -WorkingDirectory "C:\Program Files\iperf-2.0.5-2\" .\iperf.exe $iperfargs*

*while ($b -le 2) {*

*if ( ( Get-Process iperf | Measure-Object ).Count -le ($num\_copies - 1) ) {*

*Start-Process -windowstyle Normal -WorkingDirectory "C:\Program Files\iperf-2.0.5-2\" .\iperf.exe $iperfargs*

*Write-Host -NoNewline (Get-Date)*

*Write-Host -NoNewline " starting a new iPerf, new total "*

*Write-Host ((Get-Process iperf | Measure-Object ).Count)*

*}*

*Start-Sleep -s $num\_seconds*

*}*