Running Iperf on WICED-SDK Console Application

Iperf is available in the WICED SDK console test application. Load the console application onto a WICED evaluation board, then connect with a terminal program (eg. PuTTY) with parameters 115200 8N1.

Once the console application boots, standard IPerf commands are available through the console. Before running IPerf, the application needs to join to a Wi-Fi Access Point (AP) as a Wi-Fi client (WICED STA) or start a Wi-Fi AP (WICED SoftAP). Some useful examples follow.

Joining a Wi-Fi AP as WICED STA

The STA needs to join a Wi-Fi AP and either assign an IP address statically or use DHCP. The following is an example of joining an open Wi-Fi AP and assigning a fixed IP address:

```
> join your_ssid open 0102 192.168.1.108 255.255.255.0 192.168.1.1
Joining : your_ssid
Successfully joined : your_ssid
Network ready IP: 192.168.1.108
Network mask: 255.255.255.0
Gateway IP: 192.168.1.1
> ping 192.168.1.1
Pinging: 192.168.1.1
Ping Reply 9ms
```

Starting WICED SoftAP

To start a SoftAP:

```
start_ap <ssid> <open|wpa2|wpa2_aes> <key> <channel> <wps>
```

Notes and known issues:

- Starting a Soft AP in open mode (no encryption) requires a dummy argument for <key>
- SoftAP may be stopped by using the stop_ap command.

Examples:

Start an AP using WPA2-AES:

```
> start_ap YOUR_AP_SSID wpa2_aes 12345678 6
```

Start an AP using WPA2/WPA mixed mode:

```
> start_ap YOUR_AP_SSID wpa2 12345678 6
```

Start an AP using open mode (no encryption):

```
> start_ap YOUR_AP_SSID open DUMMY_KEY
```

Standard Iperf Commands

To run Iperf in a thread so that other Iperf instances can also be run, see the thread_spawn commands further below. The number of Iperf threads that can run depends on the available memory on the WICED module. The BCM943362WCD4 module can generally support two Iperf threads.

Start a TCP server:

```
> iperf -s
```

Start a UDP server:

```
> iperf -s -u
```

Start a UDP Multicast server bound to 224.1.1.1:

```
> iperf -s -u -B 224.1.1.1
```

Start a TCP client:

```
> iperf -c 192.168.1.136
```

Start a UDP client and send at voice priority (using the -S option), for 90 seconds at 10 Mbps, followed by the same command at video, best effort and background levels of priority:

```
> iperf -c 192.168.1.136 -u -S 7 -t 90 -b 10M
------------------------------------------------------------
Client connecting to 192.168.1.136, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[ ID] Interval       Transfer     Bandwidth
[  0]  0.0-90.0 sec  95.3 MBytes  8.88 Mbits/sec
[  0] Sent 67995 datagrams
[  0] Server Report:
[  0]  0.0-90.0 sec  95.2 MBytes  8.87 Mbits/sec   1.383 ms  106/67996 (0.16%)

> iperf -c 192.168.1.136 -u -S 5 -t 90 -b 10M
------------------------------------------------------------
Client connecting to 192.168.1.136, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[ ID] Interval       Transfer     Bandwidth
[  0]  0.0-90.0 sec  93.7 MBytes  8.73 Mbits/sec
[  0] Sent 66817 datagrams
[  0] Server Report:
[  0]  0.0-90.0 sec  93.7 MBytes  8.73 Mbits/sec   1.210 ms    9/66818 (0.013%)

> iperf -c 192.168.1.136 -u -S 0 -t 90 -b 10M
------------------------------------------------------------
Client connecting to 192.168.1.136, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[  0]  0.0-90.0 sec  90.6 MBytes  8.44 Mbits/sec
[  0] Sent 64607 datagrams
[  0] Server Report:
[  0]  0.0-90.0 sec  90.6 MBytes  8.44 Mbits/sec   1.449 ms    2/64608 (0.0031%)

> iperf -c 192.168.1.136 -u -S 1 -t 90 -b 10M
------------------------------------------------------------
Client connecting to 192.168.1.136, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[  0]  0.0-90.0 sec  42.1 MBytes  3.92 Mbits/sec
[  0] Sent 30037 datagrams
[  0] Server Report:
[  0]  0.0-90.0 sec  42.1 MBytes  3.92 Mbits/sec   3.077 ms   30/30038 (0.1%)
```

Start a UDP Multicast client:

```
> iperf -c 224.1.1.1 -u
------------------------------------------------------------
Client connecting to 224.1.1.1, UDP port 5001
Sending 1470 byte datagrams
Setting multicast TTL to 1
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[ ID] Interval       Transfer     Bandwidth
[  0]  0.0-10.1 sec  1002 KBytes   816 Kbits/sec
[  0] Sent 698 datagrams
```

Starting Iperf UDP server and client in separate threads (ThreadX NetX example):

```
> thread_spawn 7 iperf -s -u -i 10 -p 6000
Started thread 0x200130a8 ("iperf")
Spawning a listener.
------------------------------------------------------------
Server listening on UDP port 6000
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
> thread_spawn 7 iperf -c 192.168.1.139 -u -i 10 -t 90
Started thread 0x200149c8 ("iperf")
------------------------------------------------------------
Client connecting to 192.168.1.139, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
------------------------------------------------------------
[  1] local 0.0.0.0 port 5001 connected with 192.168.1.139 port 5001
> Spawning a server.
[ ID] Interval       Transfer     Bandwidth
[  1]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  0]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   3.894 ms    0/  891 (0%)
[  1] 10.0-20.0 sec  1.25 MBytes  1.05 Mbits/sec
[  0] 10.0-20.0 sec  1.25 MBytes  1.05 Mbits/sec   3.709 ms    0/  892 (0%)
[  1] 20.0-30.0 sec  1.25 MBytes  1.05 Mbits/sec
[  0] 20.0-30.0 sec  1.25 MBytes  1.05 Mbits/sec   4.050 ms    0/  891 (0%)
[  1] 30.0-40.0 sec  1.25 MBytes  1.05 Mbits/sec
[  0] 30.0-40.0 sec  1.25 MBytes  1.05 Mbits/sec   6.100 ms    0/  891 (0%)
[  1] 40.0-50.0 sec  1.25 MBytes  1.05 Mbits/sec
[  0] 40.0-50.0 sec  1.25 MBytes  1.05 Mbits/sec   6.219 ms    0/  892 (0%)
[  1] 50.0-60.0 sec  1.25 MBytes  1.05 Mbits/sec
[  0] 50.0-60.0 sec  1.25 MBytes  1.05 Mbits/sec   9.140 ms    0/  892 (0%)
[  0]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec   8.462 ms    0/ 5351 (0%)
[  1] 60.0-70.0 sec  1.25 MBytes  1.05 Mbits/sec
[  1] 70.0-80.0 sec  1.25 MBytes  1.05 Mbits/sec
[  1] 80.0-90.0 sec  1.25 MBytes  1.05 Mbits/sec
[  1]  0.0-90.0 sec  11.3 MBytes  1.05 Mbits/sec
[  1] Sent 8026 datagrams
[  1] Server Report:
[  1]  0.0-90.0 sec  11.2 MBytes  1.05 Mbits/sec   5.338 ms    3/ 8027 (0.037%)
Thread 0x200149c8 (iperf) exited with return value 0

>
```

Restrictions & Known Issues

- If Iperf is run without first associating and getting an IP address WICED will reset.
- If Iperf is started in client mode and cannot connect with the server then WICED will lock up.
- Joining an open WLAN and using a fixed IP address requires inputting a dummy key because the join command is designed to also cater for security.
- Many Iperf parameters cannot be changed, for example TCP window size and max segment size.
- Unless using the thread_spawn command, after running an Iperf server the WICED module must be reset, i.e. there is no break key to restore the console prompt.
- Iperf was designed to run as a process rather than in a thread, and expects the operating system to clean up after Iperf exits. This cleanup does not happen with an RTOS, even after using the thread_kill command. Therefore after running an Iperf server in a thread, if a new test requires a new setting for an Iperf server then

the WICED module should be rebooted and reconfigured.

- It is not advisable to run an Iperf client and Iperf server in separate threads concurrently. Iperf is designed to run as a separate process and it does not appear to work properly when a client and server are run in separate threads on an RTOS.
- Using the Iperf -i command to print statistics at small time intervals, e.g. 1 second intervals, will degrade throughput performance.