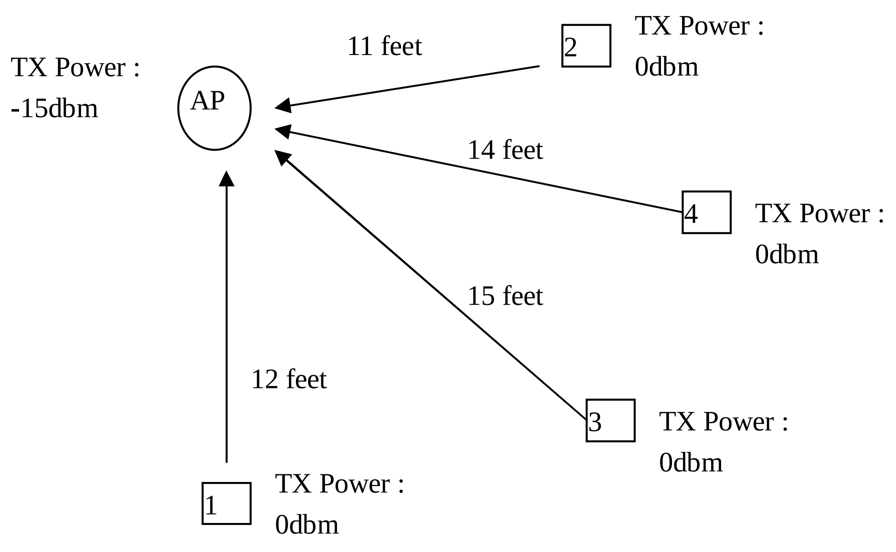
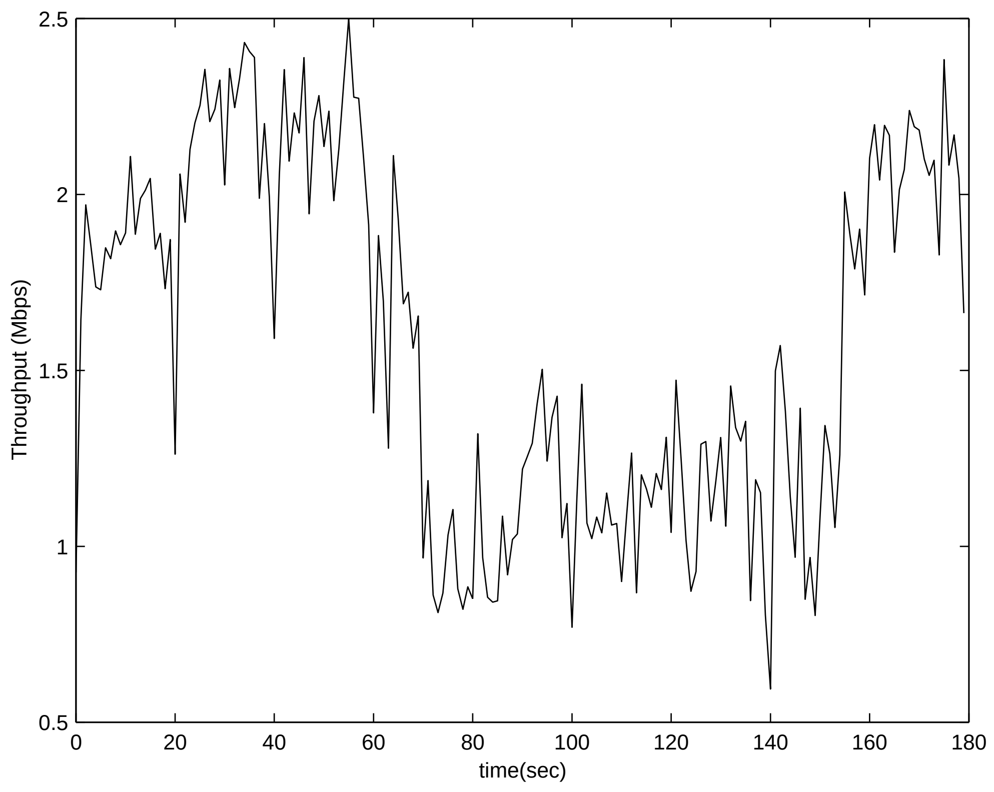
**Report of Experiment.    (Zhibin Wu, Feb 15)**

## Experiment Environment

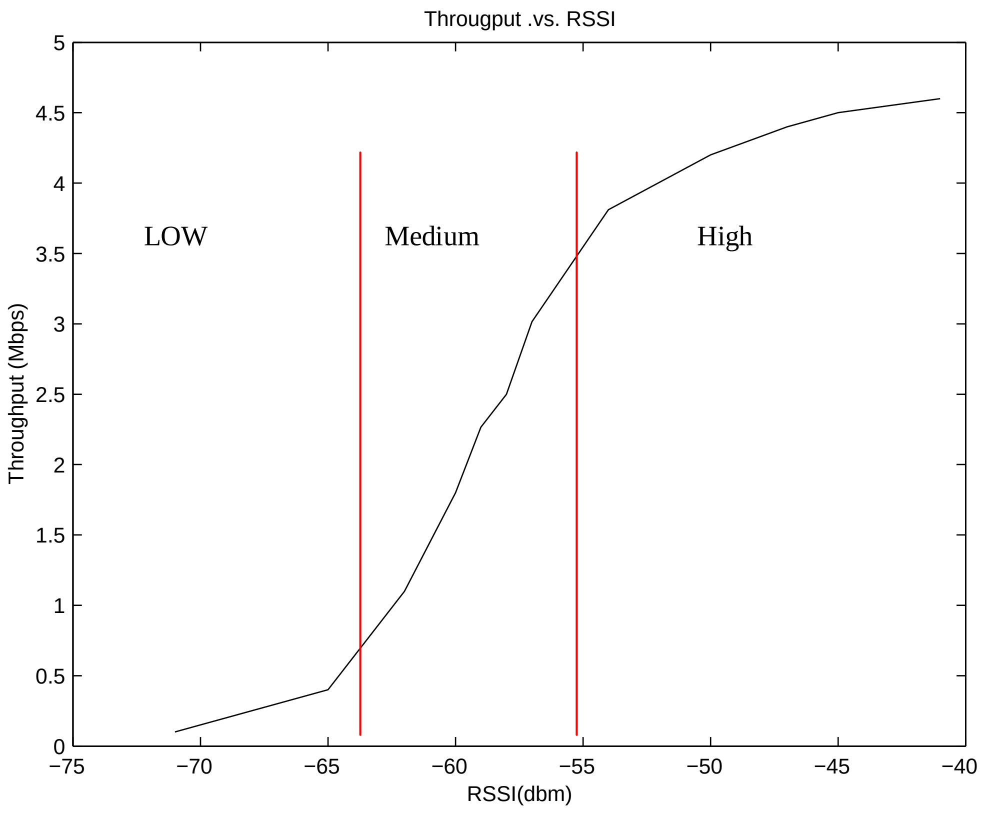


## 1. Correlation Throughput with RSSI



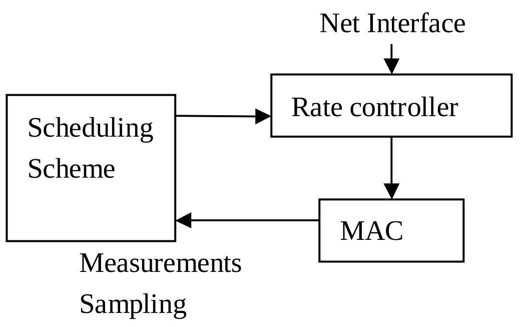
At the same time, I monitor the RSSI value, and found a RSSI value reduction corresponding to the throughput reduction.

2. Measure the Throughput-RSSI curve when only one link is available in WLAN. The curve is calibrated when packet size is 1024Bytes. Throughput are measured in application layer. (UDP payloads)



**Distributed MAC Scheduling Algorithm:**

**Idea**: With distributed scheduling algorithm, each node schedules its own transmission based on local measurements like RSSI without the instruction from AP. So when there are multiple links present in the WLAN, chances are that performance could be improved if some poor link is muted while good link is more utilized



**Scheme A**:

Each node obtains the RSSI value for every 10 ms, and after 100ms, the RSSI value is averaged and a PER (Packet Error Rate) of last 100ms is reported with average RSSI to the scheduler. Then the Rate of this node is set as

$$R = R_{\max}(1 - P_e)^n$$

where, $R_{\max}$ is the maximum throughput of current RSSI, and $P_e$ is the PER measured., and n could be 1 or 2.

**Scheme B:**

As the $R_{\max}$ is the throughput this node can achieved when there is no contention at all. So, let us assume the appropriate rate control for this node is:

$$R = \frac{R_{\max}}{C}$$

where $C$ is the contention index, the more contention, the less throughput the node should expect.

To calculate C, we first calibrate a PER table which gives an expected $P_0$ for a given RSSI value when there is no contention. Then,

$$C = 1 + (P_e - P_0)\frac{20}{S - 50}$$

where S is the measured RSSI value in percent, from (0-100).

This is from the idea that additional PER reduction is resulted from the time spent on contention.

**Results and Analysis**:

Generally, the experiment does not express the advantage of Distributed MAC scheduling, for both schemes. The respective throughput is tabled. Each node is feed with a 1200Kbps (K=1024) UDP traffic, and the reception are measured in AP as

| Without scheduling | Mbps | | | | | |
|---|---|---|---|---|---|---|
| Node 1 | 0.0456 | 0.2048 | 0.0030 | 0.0979 | 0.0864 | 0.0715 |
| Node 2 | 1.1784 | 1.1827 | 1.2174 | 1.2168 | 1.2172 | 1.2377 |
| Node 3 | 0.1622 | 0.7682 | 0.9858 | 1.0055 | 0.9836 | 1.1100 |
| Node 4 | 0.7150 | 0.7187 | 0.1035 | 0.3487 | 0.4479 | 0.8826 |
| with scheduling | | | | | | |
| Node 1 | 0.0083 | 0.1098 | 0.1979 | 0.0802 | 0.1002 | 0.1837 |
| Node 2 | 1.2343 | 1.1980 | 1.2338 | 1.2357 | 1.2353 | 1.2343 |
| Node 3 | 0.5974 | 1.1792 | 0.7699 | 0.8066 | 1.0003 | 0.9143 |
| Node 4 | 0.9899 | 1.0478 | 0.3568 | 0.8503 | 0.3229 | 0.2783 |

The last row of above table shows some performance improvement, but that's partly because I only run a MAC scheduling algorithm on node 1.
During a 4-day experiment, I only see a few cases that throughput improved slightly around 10%. And I think the main reason is:
If we think the channel as a two state "Good" and "Bad", the "Exponential Backoff" of 802.11 will punish the bad link automatically, although it does not punish as heavy as in 802.3. In this case, as the bad node has already been muted by an extended contention window coupled with constant channel occupation by traffic from the "good" link. There is little chance to see a significant throughput hike.
So, I tries to put more nodes in the "Gray Area" (between Good and Bad) to see if distributed scheduling could help some of them. By introduce MAC scheduling; I hope some nodes will not waste much time in sending a packet in a high PER channel. However, as the MAC scheduling is going to punish all nodes basically (because some delay are introduced before transmission), It' very difficult to tell who will benefit from this scheme. In many case, multiple nodes are suffered from the MAC scheduling algorithm while the saved free time is not efficiently used by other nodes.
Also, as can be seen from the RSSI-throughput curve, the nodes in the gray area are very sensitive to RSSI change, both throughput and PER have dramatic variation in this region. Thus, when I stage an environment to test the throughput, the randomness in this environment often leads to unpredictable results, even I don't move any of them. So, it is difficult to keep environment fixed as expected, and then tune the algorithm with reliable test results.
Additionally, another minor problem is that the PCMCIA wireless cards I used is found to have erratic behavior after long-time heavy traffic test, such as reporting very good SNR while actually enduring ~90% packet loss. So, each test has to be limited to several minutes.