COMS E6901 Projects in Computer Science

DYSWIS-Wifi Final Report

Ying-Chi Meng (ym2394)

Xiangying Qian (xq2120)

I. Introduction

In this project, we take two approaches in analyzing the wireless network status: Capturing frames in Linux and Querying Statistics in Windows. In both approaches we can acquire many useful parameters, such as transmitted frame counts, retry counts, ACK failure counts. We would like to observe some patterns that can help us distinguish between different sources of suffering of the network and finally provide reliable suggestion for user to recover.

As we progressed in the experiment steps, we find that it would be too imprecise if we want to classify too many different kinds of interference with the scenarios and devices we currently using. Therefore, we decided to focus on the reasons of suffering that have more obvious features for detection and easier to be set up: microwave oven interference, baby monitor interference, channel contention.

II. Design concept

It is nontrivial to diagnose the root cause of the poor performance of a wireless network. Because the real wireless environment is complex with a large number of potential interferences and collisions, which themselves could interleave with each other and make the situations more complicated.

To make the diagnosis simple and practical, single root cause and four types of sources of packet loss are assumed.

- Interference from non 802.11 sources
- Collisions for the same AP
- Collisions for other APs working on the same channel
- Interference due to a neighboring channel

We further simplify the situations by deliberately eliminating the collisions for other APs working on the same channel and interference due to a neighboring channel. The experiments are done in an environment without neighboring channel influence.

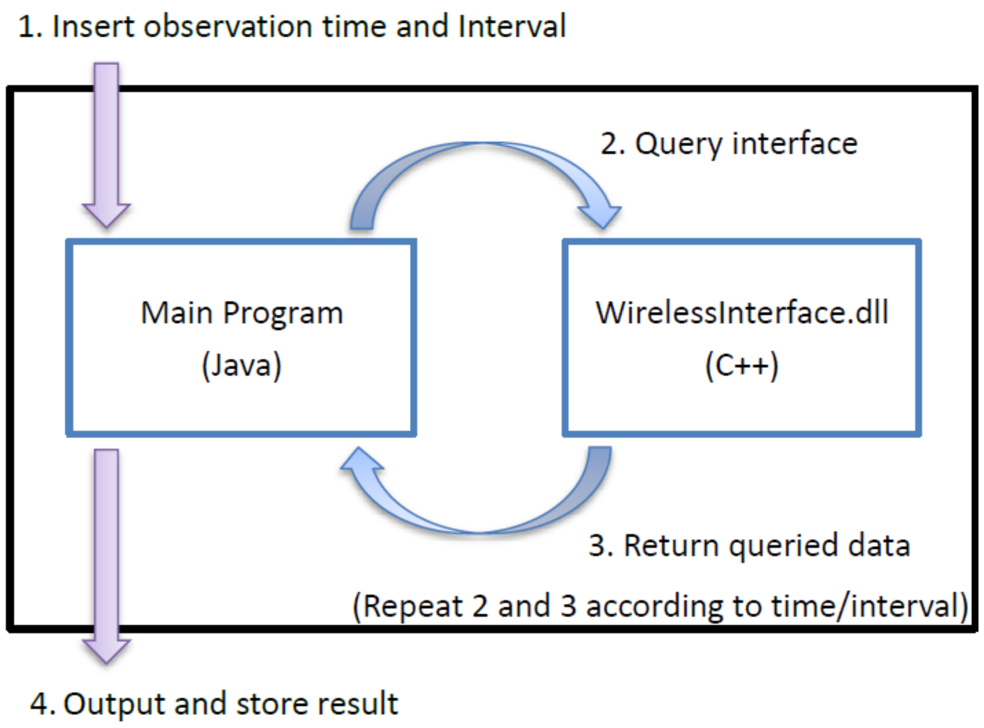
III. Implementation

A. Applications

(A) Windows – Native Wi-fi API

In Windows part, we originally expected to capture every frame before we can do any analysis. Unfortunately, the operating system doesn't support retrieving these detail information with jpcap library. Therefore, we took a statistics approach by analyzing the return values in Microsoft Native Wifi API.

In this application, we use JNI to pass values back and forth with the Native library, which was implemented in C++ (the structure shown as Pic. 3-1).


1. Insert observation time and Interval
2. Query interface
Main Program
(Java)
WirelessInterface.dll
(C++)
3. Return queried data
(Repeat 2 and 3 according to time/interval)
4. Output and store result

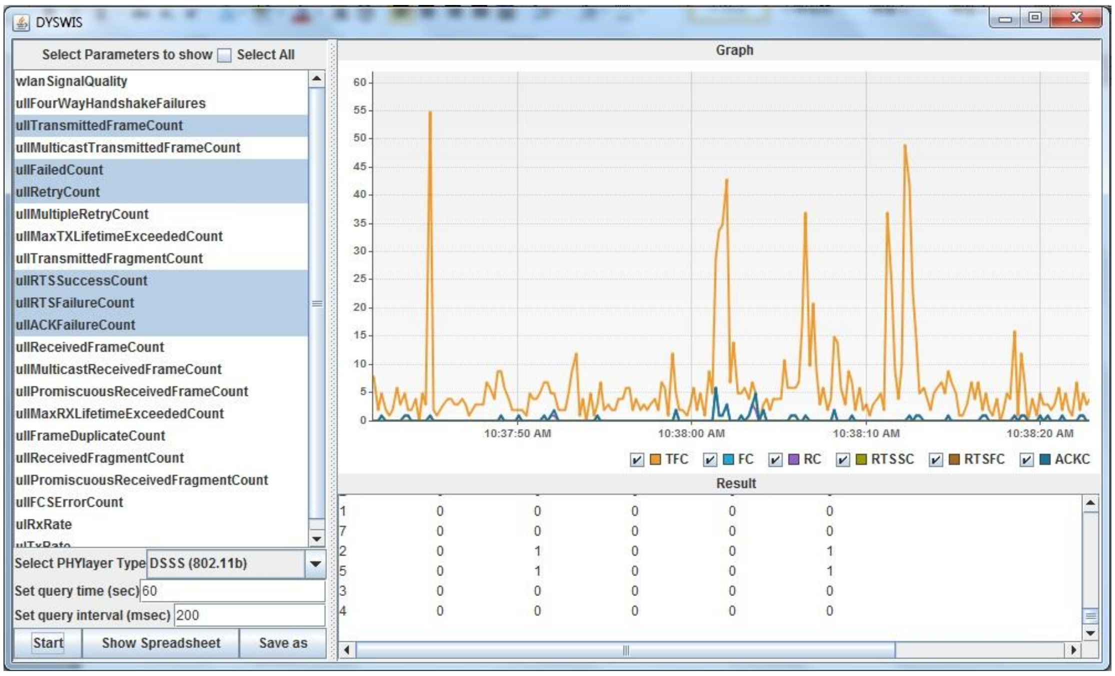
▲Pic. 3-1: The structure of the application of Microsoft native wifi API

About the data we retrieve, we implemented the data shown in physical layer. There are 3 sets of PHY layer structures (shown as Pic. 3-2) working in different modes (DSSS, OFDM, DSSS+OFDM) we can observe. To make it into a more robust tool, we integrate the function of custom configuration of parameters, saving, and showing results dynamically; all of them shown in a compact Java swing GUI (shown as Pic. 3-3).

```
struct _WLAN_PHY_FRAME_STATISTICS
        ULONGLONG ullTransmittedFrameCount
        ULONGLONG ullMulticastTransmittedFrameCount
        ULONGLONG ullFailedCount
        ULONGLONG ullRetryCount
        ULONGLONG ullMultipleRetryCount
        ULONGLONG ullMaxTXLifetimeExceededCount
        ULONGLONG ullTransmittedFragmentCount
        ULONGLONG ullRTSSuccessCount
        ULONGLONG ullRTSFailureCount
        ULONGLONG ullACKFailureCount
        ULONGLONG ullReceivedFrameCount
        ULONGLONG ullMulticastReceivedFrameCount
        ULONGLONG ullPromiscuousReceivedFrameCount
        ULONGLONG ullMaxRXLifetimeExceededCount
        ULONGLONG ullFrameDuplicateCount
        ULONGLONG ullReceivedFragmentCount
        ULONGLONG ullPromiscuousReceivedFragmentCount
        ULONGLONG ullFCSErrorCount

struct _WLAN_ASSOCIATION_ATTRIBUTES
        WLAN_SIGNAL_QUALITY wlanSignalQuality
        ULONG               ulRxRate
        ULONG               ulTxRate
```

▲Pic. 3-2: The structure of parameters we used in Microsoft native wifi API (shaded more important)



Select PHYlayer Type DSSS (802.11b)
Set query time (sec) 60
Set query interval (msec) 200

```
1      0      0      0      0      0
7      0      0      0      0      0
2      0      1      0      0      1
5      0      1      0      0      1
3      0      0      0      0      0
4      0      0      0      0      0
```

▲Pic. 3-3: The GUI version of the application

(b) Linux, Macbook – Wireshark + Alpacka

We construct the application basically based on Wireshark and Alpacka, which is a library for parsing the 802.11 frames. The structure will be more complex for the jpcap library doesn't support capturing 802.11 frames on the fly; therefore, we have to use other application that support libpcap (in our case, Wireshark), and store the result for further analysis with the *loadoffline* method.

For each network card, it supports signal-level header in different structure, and so we have write a specific parser to match its content. Integrate information from Radiotap (signal layer, shown as Pic. 3-4) and 802.11 (data link layer, shown as Pic. 3-5) header, we can retrieve some useful data for frame analysis.

Radiotap Header – Atheros (26 Bytes)

Header revision	Header pad	Header length	Present flags	MAC timestamp	Flags 2nd bit: bad FCS
1 Byte	1 Byte	2 Bytes	4 Bytes	8 Bytes	1 Byte

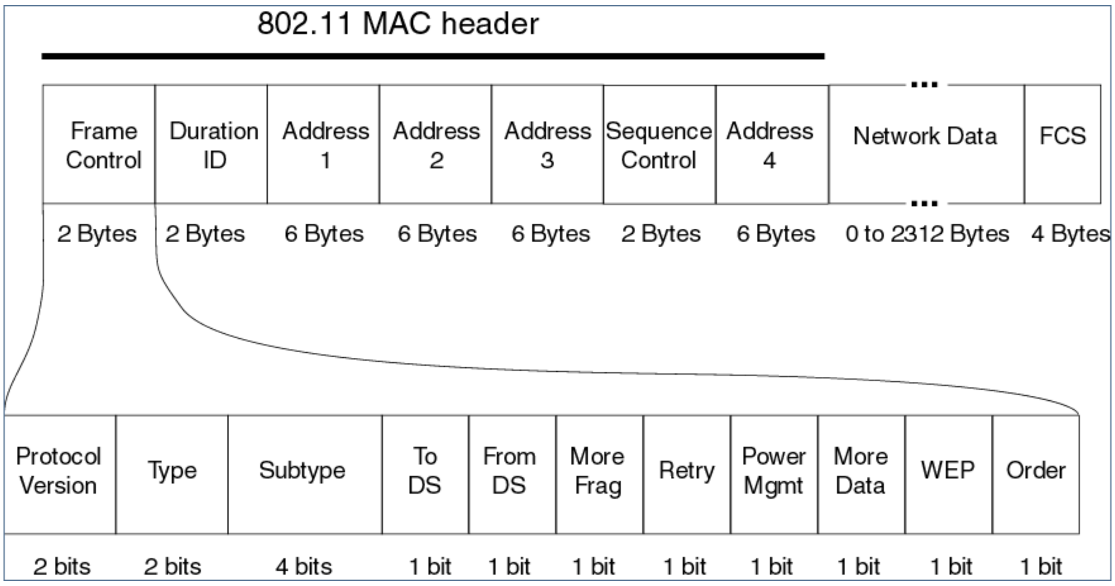
Data Rate	Channel frequency	Channel type	SSI signal	Antenna	RX flags
1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	2 Byte

Radiotap Header – Macbook (25 Bytes)

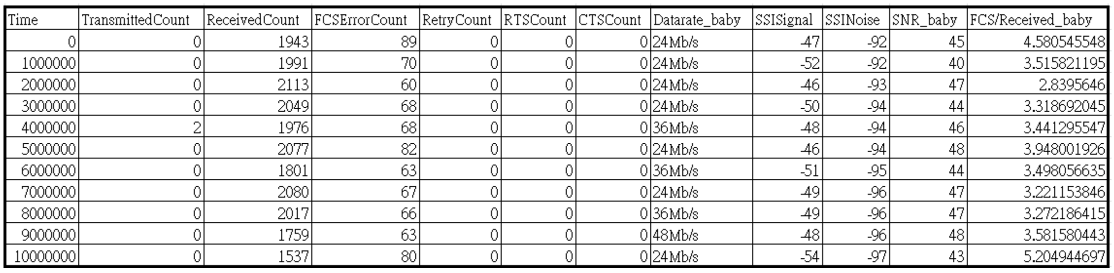
Header revision	Header pad	Header length	Present flags	MAC timestamp	Flags 2nd bit: bad FCS
1 Byte	1 Byte	2 Bytes	4 Bytes	8 Bytes	1 Byte

Data Rate	Channel frequency	Channel type	SSI signal	SSI Noise	Antenna
1 Byte	2 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte

▲Pic. 3-4: Different version of radiotap header we're using in the experiments (shaded more important)



▲Pic. 3-5: 802.11 header structure

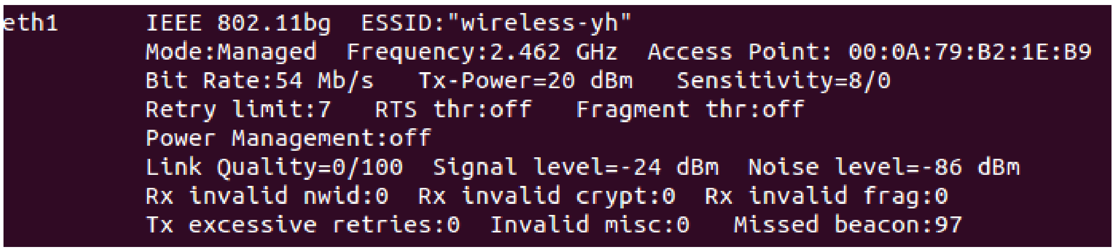


Time	TransmittedCount	ReceivedCount	FCSErrorCount	RetryCount	RTSCount	CTSCount	Datarate_baby	SSISignal	SSINoise	SNR_baby	FCS/Received_baby
0	0	1943	89	0	0	0	24Mb/s	-47	-92	45	4.580545548
1000000	0	1991	70	0	0	0	24Mb/s	-52	-92	40	3.515821195
2000000	0	2113	60	0	0	0	24Mb/s	-46	-93	47	2.8395646
3000000	0	2049	68	0	0	0	24Mb/s	-50	-94	44	3.318692045
4000000	2	1976	68	0	0	0	36Mb/s	-48	-94	46	3.441295547
5000000	0	2077	82	0	0	0	24Mb/s	-46	-94	48	3.948001926
6000000	0	1801	63	0	0	0	36Mb/s	-51	-95	44	3.498056635
7000000	0	2080	67	0	0	0	24Mb/s	-49	-96	47	3.221153846
8000000	0	2017	66	0	0	0	36Mb/s	-49	-96	47	3.272186415
9000000	0	1759	63	0	0	0	48Mb/s	-48	-96	48	3.581580443
10000000	0	1537	80	0	0	0	24Mb/s	-54	-97	43	5.204944697

▲Pic. 3-6: Sample output of Alpacka application

(c) Linux – iwconfig

We wrote a JAVA program to run iwconfig – the built-in function in Linux system for observing network status, and parse its result. We can get Signal Level and Noise Level from this function, and store them for analysis (shown as Pic. 3-7).



```
eth1      IEEE 802.11bg  ESSID:"wireless-yh"
          Mode:Managed  Frequency:2.462 GHz  Access Point: 00:0A:79:B2:1E:B9
          Bit Rate:54 Mb/s   Tx-Power=20 dBm   Sensitivity=8/0
          Retry limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=0/100  Signal level=-24 dBm  Noise level=-86 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:97
```

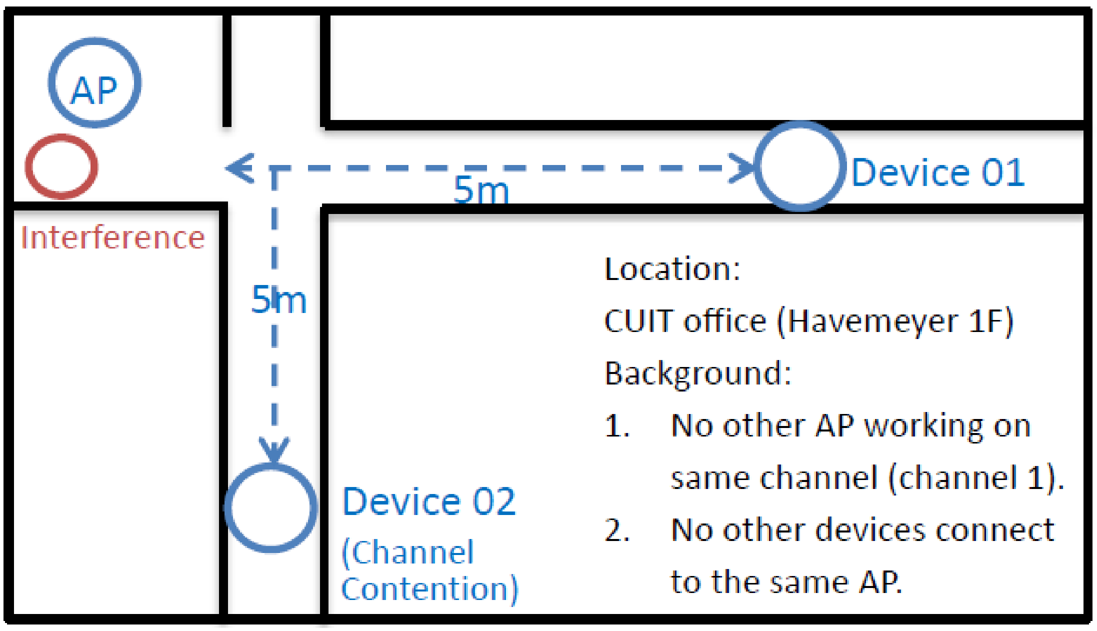
▲Pic. 3-7: Sample output of iwconfig

(d) Iperf

We use iperf to create stable TCP throughput for our experiment, and thus make variance of transmitted frame count between experimental sets smaller.

B. Experiment

We find an office placed in CUIT that have a wi-fi environment only between channel 6-11, therefore the environment make the optimized place for network analysis experiment in Columbia University we found. And thus, we conduct most important experiments right here. (Environment shown as Pic. 3-6)


AP
Interference
5m
5m
Device 01
Device 02
(Channel Contention)
Location:
CUIT office (Havemeyer 1F)
Background:
1. No other AP working on same channel (channel 1).
2. No other devices connect to the same AP.

▲Pic. 3-8: The experimental environment setup

For experiment, we use shell script (in Windows, batch file) to start iperf and our application at the same time. In our recent experiment, we follow the rules:

- *CONTROL VARIABLES:*

1. Distance between AP and device (signal strength at the same time, theoretically)
2. TCP throughput (by slightly moving the source of interference)

- *INDEPENDENT VARIABLES:*

Type of Interference/channel contention

- *DEPENDENT VARIABLES:*

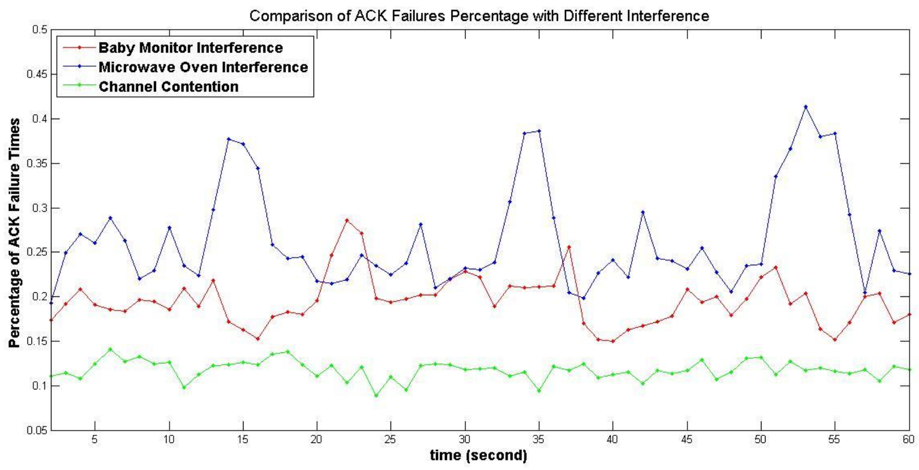
1. Successfully transmitted frame, retry frame, ACK failure, FCS error count
2. Data rate
3. Noise Level / SINR

IV. *Result and Analysis*

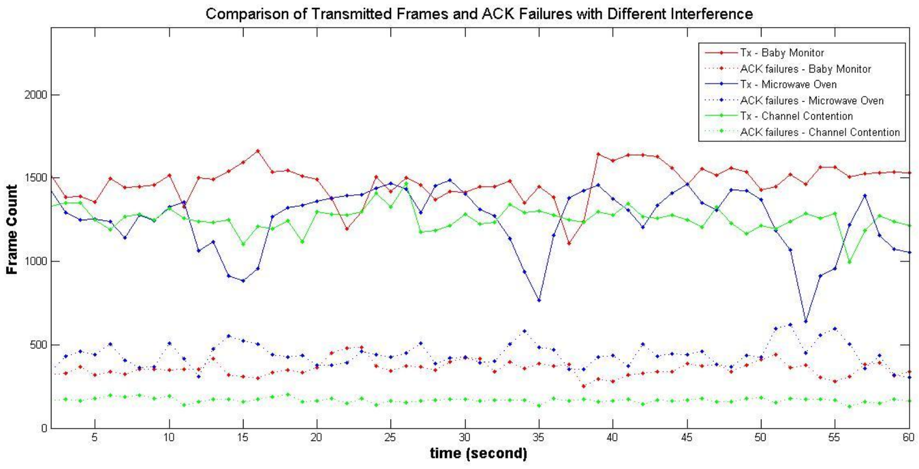
There are two control variables on our experiments. By controlling the distance of the AP and the device, we can obtain corresponding results on the number of successfully transmitted frames (Tx), the number of retransmitted frames, the number of ACK failures, the signal strength (and the TCP throughput). By controlling the TCP throughput, we can again get these results differently. We have done extensive experiments on various distances before. For now the latter case is of our interest because we hope to find some patterns on these indicators so as to identify the most likely culprit for the poor wireless network performance (which could appear as low throughput in TCP or upper layers).

A. Windows

By querying native Wifi API in windows, we are able to obtain the number of successfully transmitted frames (Tx), the number of retransmitted frames, the number of ACK failures and the signal strength.



▲Fig. 4-1: Comparison of the percentage of ACK failures under three different conditions



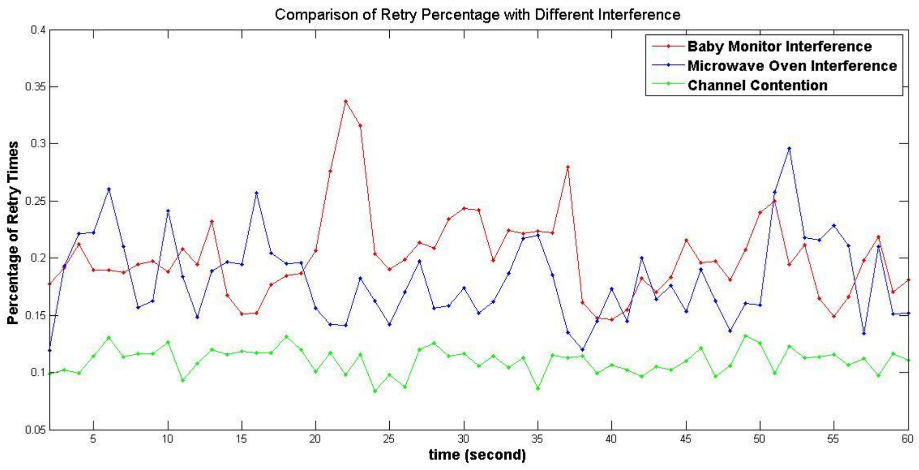
▲Fig. 4-2: Comparison of the successfully transmitted frames and ACK failures under three different conditions

Native Wifi API defines the number of ACK failures as the number of times an expected ACK has not been received (in response to an attempted transmission). On the experiment we make the TCP throughput around 7Mb/s by adjusting the distance between the AP and the injected interference source or the number competing terminals for the same AP.

As shown in Fig. 4-1 and Fig. 4-2, the number of ACK failures induced by either microwave oven or baby monitor changes with time much more sharply than the one resulted from channel contention. In addition, the percentage of ACK failures, which is given by

$$\frac{number\ of\ ACK\ failures\ within\ a\ query\ interval}{(number\ of\ successfully\ transmitted\ frames + number\ of\ ACK\ failures\ )within\ a\ query\ interval}$$

is always lowest on the condition of channel contention. The result is reasonable in the sense that interference is more likely to cause the loss of ACKs. Channel contention, on the contrary, uses CSMA/CD technique to reduce the chance of simultaneous frame transmission and frame collision and therefore lowers the number of ACK failures. Moreover, based on Fig. 4-1 and Fig. 4-2 we cannot obtain further safe conclusion on distinguishing baby monitor interference from microwave oven interference. (It seems that the variance of Tx and ACK failures on the condition of baby monitor is a little bit smaller than that of microwave oven.)



▲Fig. 4-3: Comparison of the percentage of retransmitted frames under three different conditions

Native Wifi API defines retry count as the number of MSDU/MMPDUs successfully transmitted after one or more retransmissions. On the experiment, the percentage of retransmitted frames is given by,

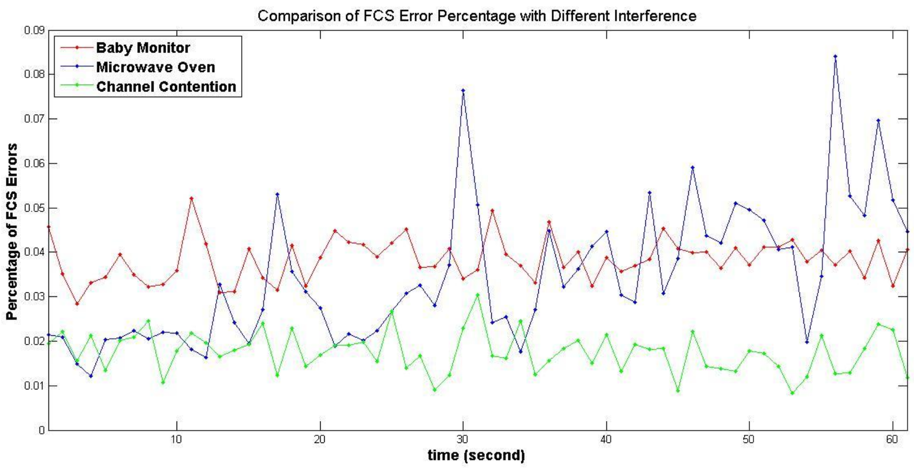
$$\frac{retry\ count\ within\ a\ query\ interval}{number\ of\ successfully\ transmitted\ frames\ within\ a\ query\ interval}$$

We make the interval 200ms, which is assumed to be large enough to do the normalization. Fig. 4-3 again shows that channel contention leads to most stable and lowest level (10%) of retransmissions. The result is as expected because the retry count should generally reflect the same property as ACK failures. In fact, one or more ACK failure corresponds to a retransmitted frame.

It should be noted that we also collect signal quality on windows experiments. However, we are not able to obtain the noise level directly. So it makes no sense to merely measure the signal strength as it does not change obviously with injected interference. We will show the SINR measurement results in Linux section.

B. Macbook (Wireshark)

We do some data rate, SINR and FCS error analysis in the aid of popular wireless network tool Wireshark. We also use the Iperf tool to control the experiment TCP throughput at 10Mb/s to 12Mb/s.

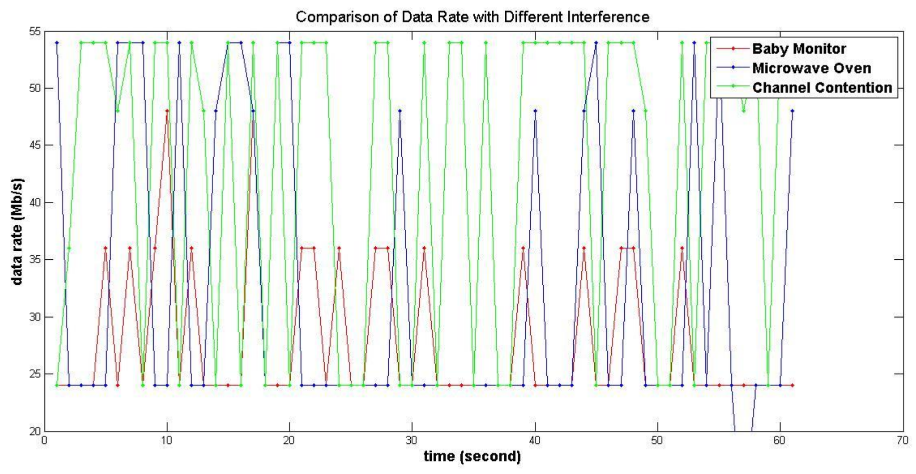


▲Fig. 4-4: Comparison of the percentage of FCS errors under three different conditions

Using Wireshark on Macbook we are easy to obtain current maximum data rate, signal strength, noise level, FCS error count and the number of successfully received frames. The FCS error percentage is computed as

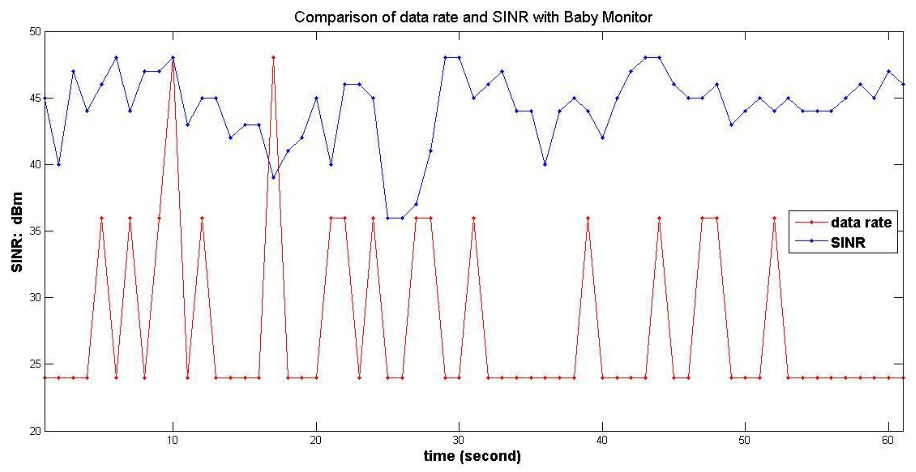
$$\frac{number\ of\ FCS\ erros\ within\ a\ query\ interval}{number\ of\ successfully\ received\ frames\ within\ a\ query\ interval}$$

Since the FCS errors can partially account for the ACK failures, the same observation can be found in Fig. 4-4. Typically, the variance of microwave oven induced FCS errors is much larger than the rest.

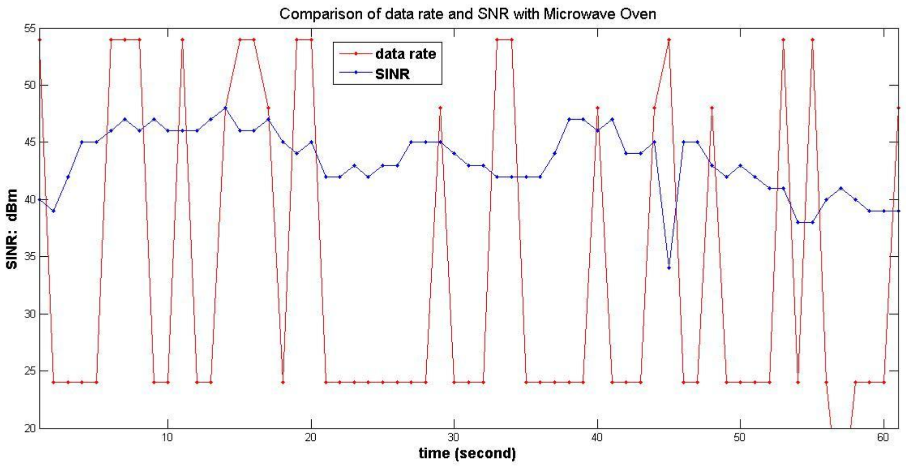


▲Fig. 4-5: Comparison of data rate under three different conditions

It is clearly shown in Fig. 4-5 that on the condition of channel contention, data rate is highest on average while fluctuates most rapidly. In general it changes from 54Mb/s to 24Mb/s and vice versa every 1 to 2 seconds. For the case of baby monitor induced interference, the data rate ranges from 24Mb/s to 38Mb/s with rare exceptions. For the case of microwave oven induced interference, the average data rate is almost the same as that of baby monitor, while the fluctuation is larger. We try to explain the reasons as follows. Channel contention does not lead to as much noise as interference, so the data rate, which is partially determined by the SINR, is less affected. But the competition among all stations connecting to the same AP may account for the instability. Interferences, on the contrary, directly lower SINR level and therefore lower the data rate. On the extreme cases, the data rate could become 0 due to extremely weak SINR. We are not very sure about the difference between microwave oven and baby monitor. It could be the inaccuracy of our experiments.

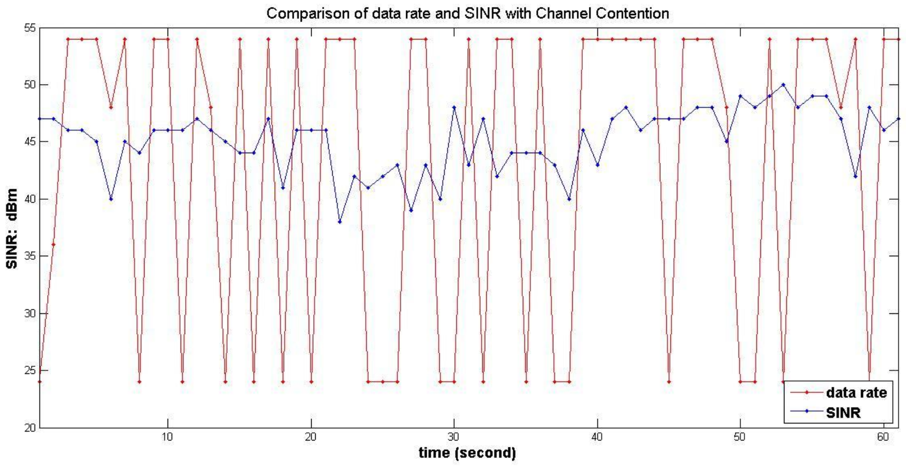


▲Fig. 4-6: Comparison between data rate and SINR with baby monitor interference



▲Fig. 4-7: Comparison between data rate and SINR with microwave oven interference

Fig. 4-6 and Fig. 4-7 show the relationship of data rate and SINR under two kinds of interference. We find that the variation trend between data rate and SINR for each case is similar to some extent, while the data rate changes less responsively. That is to say, the instantaneous data rate can well accommodate to a range of SINRs. We also observe that, at some time, the data rate changes a little bit later than the SINR. So the data rate is partially related to SINR on these cases.

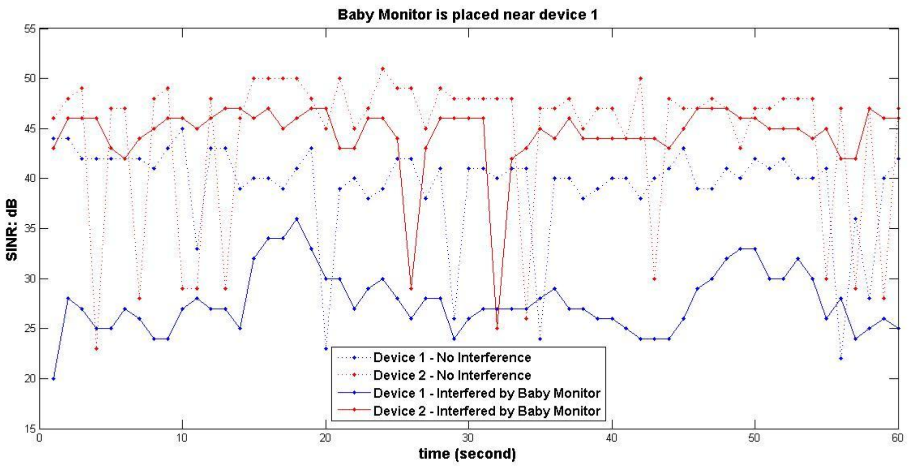


▲Fig. 4-8: Comparison between data rate and SINR with channel contention

As shown in Fig. 4-8, there is no apparent relationship between the data rate and SINR on the condition of channel contention. This is because other factors such as the number of competing terminals for the same AP might influence the data rate as well.

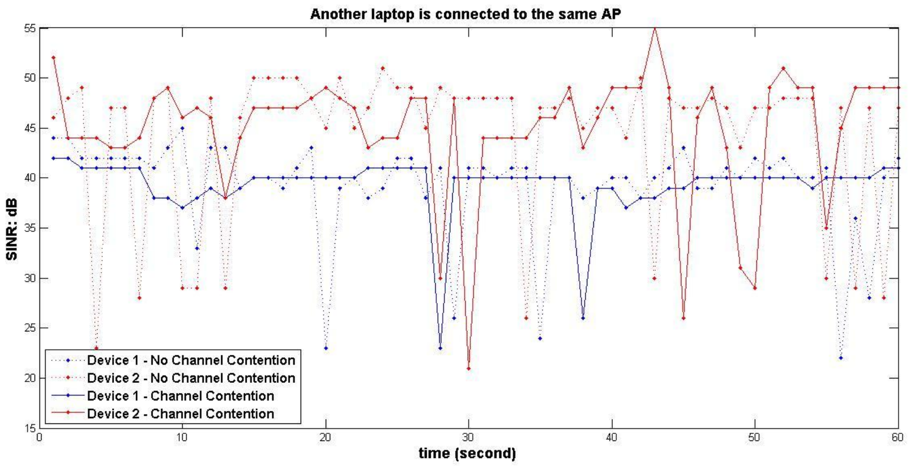
C. Linux

In Linux environment we collect instantaneous signal strength and noise strength at each sampled time and record the SINR as Signal Strength (in dB) – Noise Strength (in dB). Given the information sharing architecture, if the interference source is close to the AP, it would interfere with all devices; if it's close to only one device (device 1 in the experiment), it should only interfere with one, while the others should see less interference. So the purpose of our experiment is to compare the SINR characteristics on the two situations with two interference sources as well as channel contention.



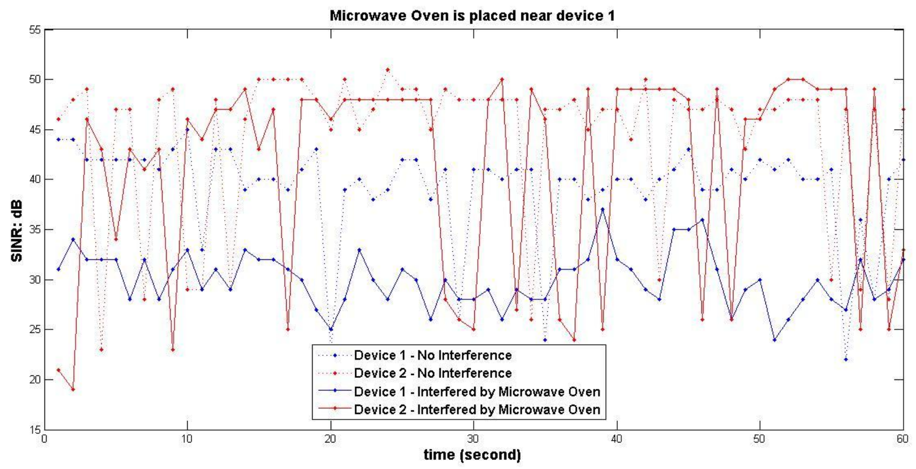
▲Fig. 4-9: Comparison of SINR between Device1 and Device 2 with baby monitor interference

Fig. 4-9 compares the SINR degradation between device 1 and device 2 when placing a baby monitor adjacent to device 1. We find an average of 11.5dB degradation for the SINR of device 1, compared to almost no degradation (less than 0.01dB) for that of device 2. The result is as expected because the severe interference induced by baby monitor is only exposed to device 1. Therefore, one can detect the exact situation by sharing information with peers based upon DYSWIS architecture.



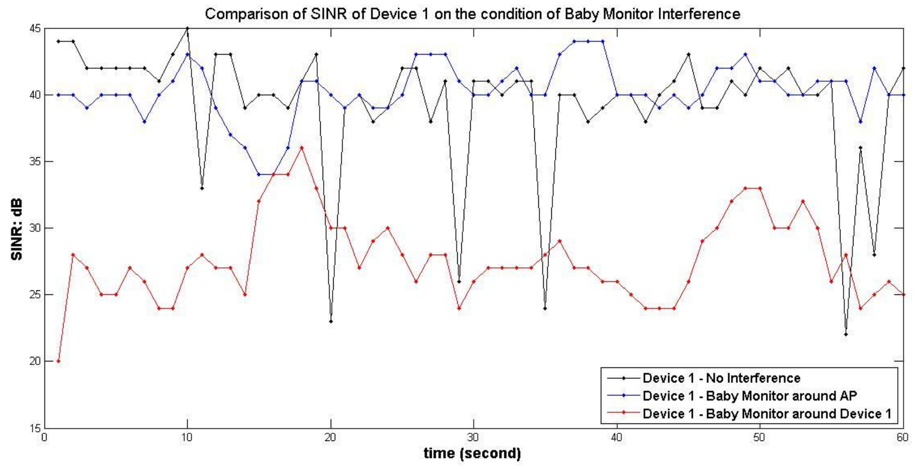
▲Fig. 4-10: Comparison of SINR between Device1 and Device 2 with channel contention

Fig. 4-10 compares the SINR on normal case to the SINR with severe channel contention. In fact there is always channel contention between device 1 and device 2 that both connect to the same AP. We worsen the situation by explicitly connecting an additional laptop to the same AP. Then we compare the two scenarios and evaluate the influence of the third laptop exerted on device 1 and device 2. As shown in Fig. 4-10, severe channel contention result in an average lower SINR for both device 1 and device 2. But such influence is not distinguishable between the two devices, which is still reasonable.

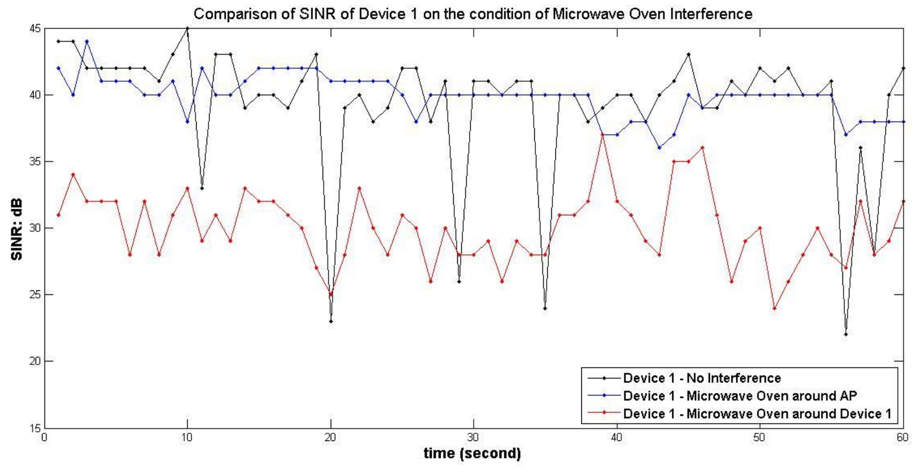


▲Fig. 4-11: Comparison of SINR between Device1 and Device 2 with microwave oven interference

Fig. 4-11 compares the SINR degradation between device 1 and device 2 when placing a microwave oven close to device 1. We find an average of 9.2dB degradation for the SINR of device 1, compared to 3.1dB degradation for the SINR of device 2. The gap between 9.2dB and 3.1dB is around 6dB, which is not as large as that of 11dB on the case of baby monitor. But it is still applicable to distinguish which device is suffering from nearby interference. The reason device 1 has fewer SINR degradation on the condition of microwave oven might be the microwave oven exhibits an ON/OFF pattern on occupying the spectrum. People have done some experiments on the influence of the two interference sources on TCP throughputs and found that, at the extreme worst cases where baby monitor thoroughly interrupt the communications, microwave oven can still maintain a 30% of TCP throughput as no interference. Our result is reasonable in the sense that SINR is proportionally related to TCP throughput. The variations in the average SINR degradation could be helpful to identify baby monitor from microwave oven interference.



▲Fig. 4-12: Comparison of SINR of Device1 with different location of baby monitor



▲Fig. 4-13: Comparison of SINR of Device1 with different location of microwave oven

Fig. 4-12 and Fig. 4-13 show the SINR of device 1 with different location of interference sources. It is clearly shown that device 1 suffers less degradation when placing the interference sources around the AP. The increasing distance between the interference source and the device could partially account for the result. Compare SINR characteristic with the percentage of ACK failures and retransmissions we find that SINR is insensitive to the injected interference around AP, while others are hugely affected if the interference source is around AP. Assuming the distance between the AP and the station is large, this is also helpful to identify the possible location of interference.

All Figures and results are based upon our latest experiments, though we have done many experiments until now.

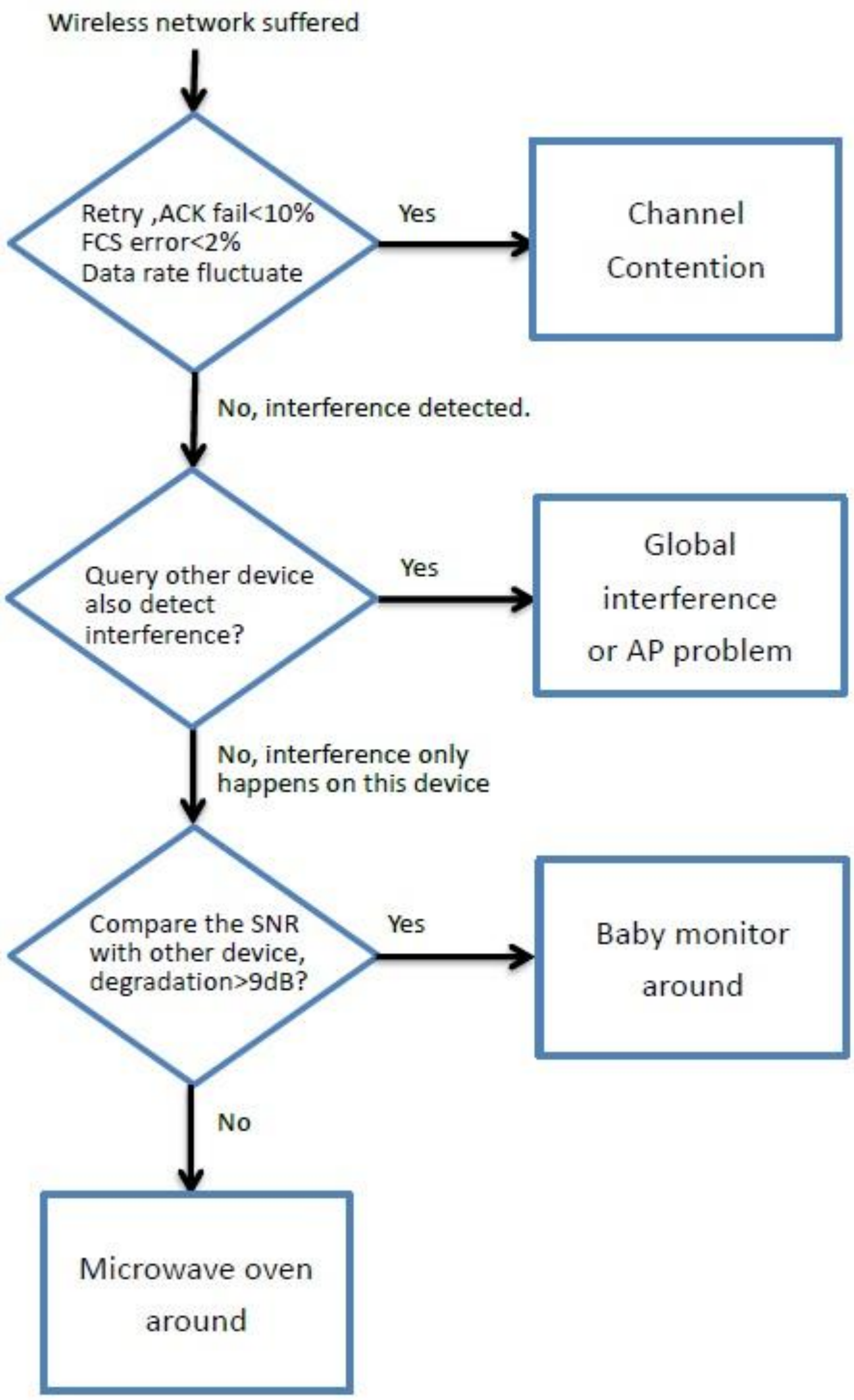
V. Conclusions

Diagnosing the root cause of the poor performance of a wireless network is nontrivial because the real wireless environment is complicated, where there are a large number of potential interferences and collisions, which themselves could have interleaved with each other and make the situations more complicated. To make the diagnosis simple and practical, we limit to a single root cause and four kinds of sources of packet loss, which are interference from non 802.11 sources, collisions for the same AP, collisions for other APs working on the same channel and interference due to a neighboring channel. We further simplify the situations by deliberately eliminating the collisions for other APs working on the same channel and interference due to a neighboring channel. So our experiments are done in an environment without neighboring channel influence. Our extensive experiment results are helpful to diagnose the root single cause of the low Wifi throughput, which is equivalent to distinguish non 802.11 sources from channel contention, indicate the possible locations of non 802.11 sources by sharing information among peers inside a WLAN network, and partly identify interference sources between baby monitor and microwave oven.

Our conclusions are shown as follows.

1. If the percentage of ACK failures and the percentage of retransmitted frames during a period of time is around 10% in average; the average percentage of FCS errors is no more than 2%; in addition, the data rate varies/fluctuates quickly and largely (the particular number is dependent on the total number of competing stations) every 1 to 2 seconds, we are safe to diagnose the root cause as a channel contention. Otherwise, we assume it as a non 802.11 sources (either baby monitor or microwave oven).
2. The approach to tell the locations of non 802.11 sources (either close to AP or close to one station) is based upon sharing information among peers using DYSWIS architecture. By exchanging SINR information among peers we are able to find out whether or not all of the stations experience the same situation. If one of them has notable SINR degradation than the rest, the interference sources are surely placed closed to the very station. Otherwise, the interference sources are likely to around the AP. Under this circumstance, all the stations are supposed to have SINR as no 802.11 interference.
3. If the interference source happens to be near one station, we can further analyze the SINR degradation of the particular station and compare it to the others. If the gap/difference of degradation is less than 9dB, possibly it is a microwave oven. Otherwise it is more likely a baby monitor.

As for the above conclusions, we have to say even though we find some patterns to do the root cause diagnosis (shown as Fig. 5-1), the exact statistic numbers used for the clarification or identification may not accurate because of our limited experiment trials. To make the conclusions more accurate and convincing we put forward some available future work in the following.


Wireless network suffered
Retry ,ACK fail<10%
FCS error<2%
Data rate fluctuate
Yes
Channel Contention
No, interference detected.
Query other device also detect interference?
Yes
Global interference or AP problem
No, interference only happens on this device
Compare the SNR with other device, degradation>9dB?
Yes
Baby monitor around
No
Microwave oven around

▲Fig. 5-1: The flow chart of our current diagnosis process

VI. *Related Work*

Authors in [7] provided a comprehensive view of the building of the peer to peer communication networks. They acquired parameters like Received Signal Strength, Beacon Loss Rate; introduced some potential accidents that made a node disconnected and provided an algorithm describing steps to analyze those parameters and categorized the outcomes.

Authors in [8] proposed a system that employed online trace-driven simulation as an analytical tool to diagnose faults and developed a complete diagnosis algorithm to find out all possible root causes. They characterized the network with four components: traffic load, routing, wireless signal and faults (random packet dropping, external noise sources and MAC misbehavior). Some small-scale experiments are done in a testbed with different scenarios.

Authors in [9] studied packet loss characteristics induced by wireless channel and MAC protocol; analyzed the loss variability across time (short burst, long burst, and residual); and explored some potential loss remedies (mainly MAC layer adaption algorithms). Wireless channel losses include external WiFi interference, non-WiFi interference and multipath interference. MAC protocol losses lie in protocol timeouts and the breakdown of CSMA over long distances and propagation delays. The paper did a series of experiments on two different real-world WiLD deployments, rural and urban and obtained two main results.

1) External WiFi interference leads to most significant amount of packet losses in WiLD links. (Compared to urban mesh networks, multipath interference is the most significant source of loss.)
2) Urban links suffer from a higher degree of residual loss.

Authors in [10] provide us a systematic architecture overview for detecting and diagnosing faults in 802.11 wireless networks. The content is very comprehensive without much detailed implementation information and diagnosis algorithm. But it mainly focuses on companywide commercial network, so that network security and overhead cost are two important factors.
The paper and related patent

Authors in [11] proposes an approach for detecting the presence of pulsed interference affecting 802.11 links, and for estimating temporal statistics (complete probability distribution) of this interference. They distinguish packet losses due to collisions (a feature of normal operation in 802.11 WLANs) from packet losses due to

interferences. It is important to distinguish between the two cases because the packet loss rate induced by collisions can be significant even in quite small WLANs. Detailed packet pair bursting approach can be found in the paper.
At the same time, the author set up an experiment based upon the packet pair bursting method to simulate Microwave Oven Interference and validate the experiment results with a spectral analyzer. (The results indicated that MWO interference is estimated to be approximately periodic with period of 11ms.)

IEEE 802.11 employs DCF as primary mechanism to access the medium. Its performance is very sensitive to the number of stations competing on the same channel. Authors in [12] proposed a modified Kalman Filter estimation of n, which is unbiased and of high accuracy, as function of the conditional collision probability (p) encountered on the channel.

As the ISM band becomes increasingly crowded with diverse technologies, many 802.11 access points may not find an interference-free channel. Authors in [2] presented a TIMO, a MIMO design that enables 802.11n to communicate in the presence of high-power cross-technology interference. To compare and evaluate the performance improvements using the proposed technique, the authors firstly did some experiments focusing on digital cordless phone, baby monitor, microwave oven and frequency hopping bluetooth and concluded their characteristics respectively.

Authors in [13] presented detailed and comprehensive experimental results (using the ORBIT radio grid testbed) to quantify the effects of inter- and intra-radio interference in representative SOHO scenarios. In particular, different topologies, traffic loads and number of interfering devices were emulated to show the impact of multi-radio interference and to characterize each kind of interference. Further, a cross-layer, multi-radio interference diagnosis framework (called “spectrum MRI”) was described with the aim of isolating and classifying multi-radio interference problems using heuristic and model-based methods.

VII. Future Work

- ***Find the best compatible network card for linux (or use macbook instead)***

In our attempt to experiment linux system using Wireshark, we find that both the internal and external network cards are both too old to run with some newer drivers that are useful in detecting noise level. And also we observed that the data captured by old cards are quite unstable when surrounding signals are too complex, i.e. many frames will be hided and cannot be detected. We tried to buy one new adapter (ConnectGear WU260N), but unfortunately it still didn't meet our need. And thus, we can only implement the set of experiment using Wireshark on Macbook, and then analyze the data using linux for now.

Another suggestion is that we should use Macbook for experiment instead because we already find that Macbook has a full-supported network card, which is able to run in monitor mode without any external supports. In addition, Macbook has a more stable and identical system, which would make experiments like channel contention more convincing if all devices are using the same interface.

- ***Apply machine learning method to find some admissible heuristic for distinguishing different types of interference***

For now, we can only find kinds of tendencies by analyzing the result. The hardest part of this project is that we are lack of ways to demonstrate if the result we get in trustable. For example, the Signal Level and Noise Level may be different between the result using iwconfig, wireshark, and Native wi-fi API at the same time, but we cannot make sure which one is true.

A reason of the uncertainty can be the experimental errors, and we are possible to solve it by collecting large amount of data. However, for observing those data only in time domain may be too difficult for people, and we think the best way doing this is to apply machine learning to classify the conditions. Traversing each independent variable and record the result for training may make the algorithm more reliable.

For now, to avoid too much amount of uncertainty, we are only able to focus on a specific condition and make conclusions by observation of graphs.

VIII. Reference

[1] DYSWIS, Collaborative network fault diagnosis, Kyung-Hwa Kim, Vishal Singh, Henning Shulzrinne

[2] Gollakota S, Adib F, Katabi D, Seshan S, Clearing the RF Smog: Making 802.11 Robust to Cross-Technology Interference, SIGCOMM 2011 http://www.cs.washington.edu/homes/gshyam/Papers/TIMO.pdf

[3] **http://jpcap.sourceforge.net/** Jpcap

[4] **http://www.wireshark.org/** Wireshark

[5] http://code.google.com/p/alpacka/ Alpacka

[6] http://msdn.microsoft.com/en-us/library/windows/desktop/ Microsoft Native Wifi API

[7] Ranveer Chandra, Venkat Padmanabhan, and Ming Zhang, WifiProfiler: Cooperative Diagnosis in Wireless LANs, Microsoft Research June 2006 http://research.microsoft.com/en-us/um/people/ranveer/docs/WiFiProfiler-MobiSys.pdf

[8] Lili Qiu, Paramvir Bahl, Ananth Rao, and Lidong Zhou, Fault Detection, Isolation and Diagnosis in Multihop Wireless Networks. MSR-TR-2004-11, December 2003. http://pages.cs.wisc.edu/~suman/courses/740/papers/fault-isolation.pdf

[9] Nedevschi, S. Patra, R. Surana, S. Brewer, E. Subramanian, L. Packet Loss Characterization in WiFi-based Long Distance Networks. INFOCOM 2007. 26th IEEE International Conference on Computer Communications.

[10] Atul Adyta, Paramvir Bahl, Ranveer Chandra, Lili Qiu, Architecture and techniques for diagnosing faults in IEEE 802.11 infrastructure networks. Appears in the Proceedings of the Tenth Annual International Conference on Mobile Computing and Networking (MobiCom 2004) http://research.microsoft.com/pubs/73484/diagnostics.pdf

[11] Brad W. Zarikoff, Douglas J. Leith, Measuring Pulsed Interference in 802.11 Links, IEEE/ACM Transactions on Networking, Issue: 99.

[12] Giuseppe Bianchi, Ilenia Tinnirello, Kalman filter estimation of the number of competing terminals in an IEEE 802.11 network. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. INFOCOM 2003. Pages 844 - 852 vol.2. http://www.ieee-infocom.org/2003/papers/21_02.pdf

[13] Akash Baid, Suhas Mathur, Ivan Seskar, Sanjoy Paul, Amitabha Das, Dipankar Raychaudhuri, Spectrum MRI: Towards Diagnosis of Multi-Radio Interference in the Unlicensed Band. 2011 IEEE Wireless Communications and Networking Conference (WCNC), Pages 534 - 539.