IEEE 802.11 Packet Analysis System

by Joshua Jenkins

Technical University of Crete

Introduction

The goal of our system is to perform parsing and statistical evaluation of Wi-Fi frames by capturing detailed metrics related to Wi-Fi throughput between two devices, or an overall perceived Wi-Fi channel density. The throughput analysis system provides a reliable theoretical estimate of throughput without occupying the channel. The 802.11 density system generates scores that reflect the overall perceived network density based on the captured trace file. This is not intended as a direct metric for performance, but rather as an indicator of how "active" the 802.11 environment is around you. It reflects this activity by combining factors like nearby access points, busyness, and traffic into one general score.

The system uses an Electron and React-based frontend with a Python backend. These two components communicate with each other via subprocess messaging. The backend is responsible for parsing and analyzing trace files and returns structured data objects for the frontend to display. When analyzing a trace file, the system provides two distinct modes: network density analysis or throughput analysis between two sources. Each mode produces a different type of report tailored to its purpose with the former focused on overall network activity and congestion, and the latter on link-specific performance between a sender and receiver. Both support overlaying trends in key metrics, to reveal correlations and relationship.

In evaluating our throughput metric, we compared its estimates against the ground truth provided by an online speed test. For tests conducted on the 2.4 GHz band, the metric consistently overestimated actual throughput, with an average overestimation of 62.1% across all scenarios. In contrast, the 5 GHz band showed significantly better alignment: the metric underestimated throughput by an average of 9.7% in two tests and overestimated it by 26.8% in two others. We conclude that the overall discrepancies are largely due to underestimating the impact of retransmissions and failing to account for channel contention from other devices.

Because there's no direct way to measure 802.11 "density," we validated our density score by checking its correlation with packet retry rate—a known symptom of contention **[1]**. We collected one-minute traces in the busiest 2.4 GHz and 5 GHz bands at a residential street, an internet café, and Chania's Old Venetian Port at five timepoints throughout the day. The density score achieved a moderate correlation ($r = 0.61$) in 2.4 GHz and a strong correlation ($r = 0.75$) in 5 GHz.

Design

In both density and throughput analysis, the system follows a structured pipeline consisting of the uploading, parsing, analysis, and visualization. Each stage in the pipeline includes robust error handling and guarding to ensure the reliability of the system. Before going in depth on how the uploading process works, it's important to understand that our Electron application follows a three-layer architecture: the presentation, orchestration, and data [1]. The presentation layer is our user interface component, which displays the data visualizations and handles user interactions. The orchestration layer handles the communication and coordination between our presentation and data layers. In our application, the orchestration layer handles tasks such as spinning up our backend Python process, managing the flow of requests and responses between layers, and catching any errors returned from the data layer [2] [3]. Our orchestration layer creates our Python process and establishes communication pipes on application startup [3]. Finally, the data layer operates outside of Electron's own processes and is where our Python backend resides.

The orchestration and data layers interact via inter-process communication, where the Electron main process launches our Python backend as a child process and exchanges requests and responses over its standard I/O streams [2] [3]. This approach relies on a direct parent–child relationship, meaning the child's lifecycle is managed entirely by the parent process rather than two separate processes communicating [4]. After spawning, the parent process gets three built-in communication streams with the child: stdin (writable input stream), stdout (readable output stream), and stderr (readable error stream). In our case, the Electron main process sends serialized JSON data to the Python backend via the child's stdin and listens for the processed output on the child's stdout. To determine when a complete message has been received, we use newline characters as delimiters, to parse each message cleanly. Some trade-offs of this design include having a single-entry point for communication, which can limit the ability to handle concurrent or parallel requests, error-prone message handling and buffering, lack of structure, and limited bi-directionality. However, because of its ease of setup, and our minimal request/response needs, this method triumphs over alternatives like TCP sockets or HTTP APIs for our case.

The upload stage serves as the entry point to the system. Users can provide a packet capture (PCAP) file obtained from an 802.11 sniffer using tools like MacOS's built in wireless diagnostics or Wireshark. The file should contain the frames of the specific channel and bandwidth that the user wants analyzed. Because our data layer exposes a single-entry point, the communication protocol must distinguish between throughput or density analysis. To handle this, all payloads sent over stdin include a "process" attribute that specifies the requested service, allowing the backend to route the request accordingly. The upload payloads are intentionally lightweight. For throughput analysis, the payload includes the file path, the access point address, and the host address, allowing the backend to filter data frames and isolate downlink activity between the specified AP and client. In contrast, the density analysis payload only includes the file path, as it does not require targeted filtering.

The parsing stage is where the PCAP file is opened, and all necessary packets are read into memory. This is done using Pyshark, which exposes an interface for TShark in Python [5]. This library doesn't do any packet parsing itself but just uses Wireshark's command-line utility XML exporting ability for parsing in Python. Our parsing system includes robust error handling to account for missing or malformed attributes. If a percentage of packets are missing critical fields, the system will fail processing to avoid generating unreliable analysis results. While Pyshark offers a very straightforward API, it has noticeable performance issues with larger captures. This is primarily due to its reliance on spawning a TShark subprocess and parsing packets sequentially. Consequently, our system suffers with processing larger PCAP files due to this high latency parsing stage. Although addressing this is outside the scope of our current work, future improvements could involve switching to a pure python library such as Scapy or interacting directly with the TShark utility without additional Python overhead.

The analysis stage consumes the structured output from the parser and applies metric-specific logic to compute insights. This responsibilities of this stage include handling metric-specific filtering and aggregation, applying our throughput and density algorithms, and transforming results into an easily readable format for the presentation layer. Each analysis module operates independently and computes a diverse set of supporting statistics that contextualize and enhance the interpretation of the primary metric, such as signal quality trends, retry behavior, most active access points, etc. The resulting structured data is then passed to the presentation layer, where it is stored and used to power the visualizations shown to the user. The specific implementation details and design justifications for each algorithm are presented in later sections.

The visualization stage is built using React and Recharts and Tailwind CSS, providing a modern, interactive and easy-to-navigate interface that displays the analyzed data in a clear and informative report format. One of the key strengths of this visualization format is the ability to overlay multiple metric graphs, allowing users to explore correlations and relationships between different aspects of the network behavior. Figures 1 and 2 illustrate this capability, showing how metrics such as throughput and retry rate can be visualized together for deeper insight.

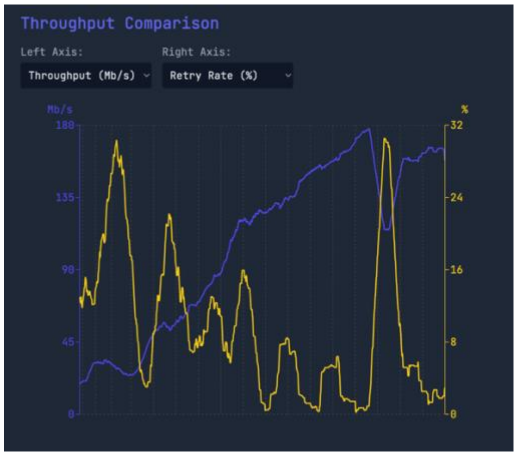

Figure 1. Overlay of throughput and retry rate

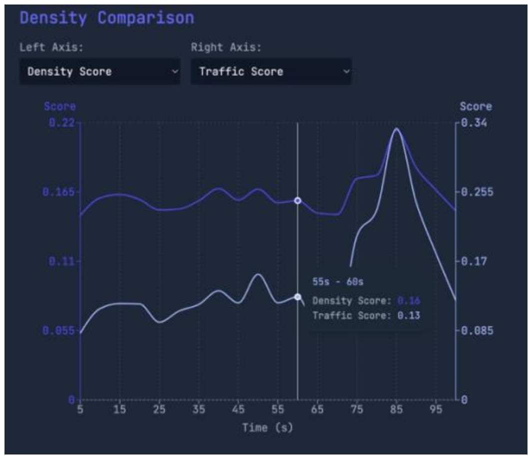

Figure 2. Overlay of Density Score and Traffic Score

Throughput Algorithm Design

The basic approach to find the theoretical downlink throughput between an access point and a client is estimated by scaling the frame data rate by the successful transmission rate.

$$T = \mathrm{R}_{\mathrm{frame}}\left(1 - \frac{\mathrm{N}_{\mathrm{retry}}}{\mathrm{N}_{\mathrm{data}}}\right)$$

This intuition is straight forward and reasonable first-order estimate of throughput. $\mathrm{R}_{\mathrm{frame}}$ represents the ideal data rate assuming all frames succeed on the first attempt. 802.11 retry frames are frames retransmitted whenever the sender does not receive an acknowledgment (ACK), indicated by the Retry bit in the frame control field [6]. Each retransmission, often triggered by collisions, interference, or poor SNR, consumes additional airtime, increases medium contention and latency. Consequently, retransmitted frames do not contribute new data and should be excluded from throughput calculations, which is indicated by $1 - \frac{\mathrm{N}_{\mathrm{retry}}}{\mathrm{N}_{\mathrm{data}}}$. However, real-world conditions are more complex and this simple estimation can misrepresent actual throughput due to unaccounted factors which influence performance.

One of these unaccounted factors is protocol overhead. In 802.11, a MAC service data unit (MSDU) is fragmented into a fragment burst, where once the transmitter wins the medium, it transmits Fragment 0 and then pauses only a Short Interframe Space (SIFS) to receive Acknowledgement (ACK) 0 before immediately sending Fragment 1 [7]. This pattern continues to repeat until the entire data unit is sent. Thus, with our initial approach, we are completely ignoring overhead with ACKs and SIFS. One paper showed that at a 54 Mb/s PHY rate, 802.11a delivers only 25 Mb/s of MAC-layer throughput, meaning 54% of the airtime is consumed by MAC headers, backoff, ACKs and other control overhead [8]. This illustrates how any realistic throughput model must account for these protocol-layer inefficiencies to avoid substantial overestimation. Moreover, every frame has a preamble, which consists of known symbols (Short and Long Training Fields) that enable the receiver to synchronize with the sender [9]. The preamble is necessary for packet decoding to compensate for multipath and frequency offset. The modern day OFDM PLCP preamble contains 12 symbols occupy 16 µs of airtime [10]. After the preamble, there is also a 4 µs SIGNAL field which carries the PLCP header needed to demodulate the payload.

Computing retry-rate naively as $\frac{N_{retry}}{N_{data}}$ breaks down on the client side because a passive listener often only ever sees the final successful transmission (with its retry bit set) and never the original failed attempts. Without being able to see the non-retried data frames, the retry rate will have a much greater impact on throughput than it expected. For example, the case where our client only sees retry frames, would result in a retry rate of 1 and a throughput of 0. But this clearly doesn't reflect reality as our client is still receiving valid data frames, just ones that happen to be retransmissions. The flaw in the original method is that it interprets retries as failed transmissions, when in fact they represent delayed, but still completed deliveries. Therefore, we need a more accurate model that reflects the cost of retries, not their presence alone. We redefine retry-rate as the average number of transmission attempts per successful reception. Whenever a received frame carries the retry flag, we assume it having required multiple tries, and then compute the retry-rate by averaging those inferred attempt counts over all successfully delivered frames. This approach more accurately captures retransmission overhead from the client's viewpoint, producing a more meaningful throughput estimate.

Accordingly, the downlink throughput can be expressed as:

$$T_{dl} = \frac{R}{A}$$

Here, R is the effective data rate including all protocol overhead such as MAC headers, control frames, and inter-frame spacing. A is the average number of transmissions per successful frame. We divide the effective data rate R by the average attempts A because each successfully delivered frame ties up the channel for A transmissions. By scaling R by $1/A$, we account for the extra airtime cost of retries and obtain the true average throughput.

$$A = \frac{N_{non\_retry} + k\,N_{retry}}{N_{data}}$$

In this expression, k denotes the mean number of transmission attempts required for a frame marked as a retry before it was successfully received. In our application, we arbitrarily chose $k = 2$ which assumes that retries succeeded on the second attempt for simplicity and to keep the scope manageable. In practice though, k should be derived empirically by analyzing the actual retry distribution and can be improved in future work.

To compute the effective data rate R, we pivoted from the nominal PHY rate labels and instead divided the total number of successfully received payload bits by the actual airtime consumed to transmit those bits. Aruba Networks validated reference design uses the same approach shows that, for a 512-byte payload, only 16 % of the Transmission Opportunity (TXOP) airtime carries user data while the remaining 84 % is consumed by MAC-level overhead and framing [11]. They then define the "effective" TXOP data rate as:

$$\mathrm{EDR} = \frac{\text{MPDU Payload Size (bits)}}{\text{TXOP Airtime }(\mu\text{s})}$$

We can narrow this equation down to the fragment level to find the effective data rate of each packet, finding the reserved airtime and payload size for each packet.

$$R = \frac{\text{Packet Payload Size (bits)}}{\mathrm{T}_{\mathrm{air}}}$$

In this study, we focus exclusively on the airtime consumed by fragmented MPDUs and omit the overhead associated with RTS/CTS exchanges and contention backoff preceding the first fragment. Incorporating RTS/CTS handshakes and backoff intervals would require modeling medium access contention across multiple users, which lies beyond the scope of this project. Moreover, we compute the ACK duration analytically rather than extracting it directly from the trace. Accurately identifying which ACK corresponds to which data frame would require tracking sequence numbers, timing alignment, and retry behavior across multiple packets, which introduces significant complexity. By assuming a fixed ACK duration, we can estimate ACK airtime without needing to reconstruct per-packet ACK associations. Nonetheless, for a more comprehensive throughput estimation this algorithm could be extended to include both additional MAC-layer overheads.

Combining all relevant components so far, the total airtime for a single data packet would be:

$$\mathrm{T}_{\mathrm{air}} = \mathrm{T}_{\mathrm{pre}} + \mathrm{T}_{\mathrm{sig}} + \frac{\mathrm{L}_{\mathrm{data}}}{\mathrm{r}_{\mathrm{data}}} + \mathrm{T}_{\mathrm{SIFS}} + \mathrm{T}_{\mathrm{pre}} + \mathrm{T}_{\mathrm{sig}} + \frac{\mathrm{L}_{\mathrm{ACK}}}{\mathrm{r}_{\mathrm{ack}}} + \mathrm{T}_{\mathrm{SIFS}}$$

We assume a standard 14-byte size for ACK frames, transmitted at a standard 6.5 Mbps rate [12]. We use the standard time for the SIFS of 16 µs [13]. We assume 16 µs for the preamble and 4 µs for the SIGNAL header, as we mentioned before. Wireshark computes the total duration of each data packet based on its PHY, length, and overheads excluding SIFS. This means the transmission time for the data portion is already abstracted and provided, allowing us to directly use $\mathrm{T}_{\mathrm{data}}$ field without manually calculating it [14]. Thus, our total transmission and overhead duration for one data frame comes out to the following:

$$\mathrm{T}_{\mathrm{air}} = \mathrm{T}_{\mathrm{data}} + 69\,\mu\mathrm{s}$$

After some initial testing, we found that our model was severely underestimating throughput in some cases due to not accounting for frame aggregation [15]. Frame aggregation combines multiple data frames into one transmission to reduce overhead and improve Wi-Fi throughput. As a result, our initial approach introduced artificial overhead, leading to a significant underestimation of actual throughput.

To account for frame aggregation, we extract each frame's aggregate ID (frames that belong to the same aggregation share the same ID) and duration. Instead of assigning overhead to each individual frame, we calculate overhead based on the frame's proportion of the total duration of its aggregate group. Thus, our calculated throughput for one sniffed frame is:

$$R = \frac{P_{\text{bits}}}{T_{\text{data}} + \delta \, \frac{T_{\text{data}}}{T_{\text{agg}}} O_{\text{agg}} + (1 - \delta) \, O_{\text{seq}}}$$

Where $\delta = 1$ if the frame is an aggregate; otherwise 0, P_{bits} is effective data bits, T_{data} is the duration given by Wireshark, T_{agg} is the frame's aggregate group duration, O_{agg} is the total overhead for an aggregate group, which we have set to 64 µs based on different trace analysis, and O_{seq} is original transmission overhead at 69 µs.

Combining this rate into a duration weighted aggregation of all frame throughputs, our total throughput for a group of frames becomes:

$$R_{\text{avg}} = \frac{\sum_i R_i \; T_{\text{air},i}}{\sum_i T_{\text{air},i}}, \quad \text{where } i \text{ indexes each parsed data frame}$$

Density Algorithm Design

Our density score integrates three metrics to characterize the local 802.11 environment captured by a trace file. It contains a proximity-weighted count of access points to capture device presence, airtime utilization to quantify channel occupancy, and sustained bitrate to reflect data throughput. Rather than attempting to model precise performance, this metric provides a high-level representation of the 802.11 landscape. Each of the three metrics is converted into a normalized score ranging from 0 to 1, then these normalized components are combined using a weighted sum, with weights selected to reflect their relative importance. We chose these weightings:

$$S_d = 0.5 \cdot N_{\text{eff}} \; + \; 0.35 \cdot U \; + \; 0.15 \cdot D$$

Where N_{eff} is the proximity-weighted access points score, U is the airtime utilization score, and D is the bit-rate score.

Because interference in an 802.11 network grows almost linearly with the number of active senders [16], we give N_{eff} the largest weight to reflect its outsized impact on channel contention. We utilize proximity weighting so only nearby transmitters drive the metric, while distant APs do not inflate the score, mapping out a "wireless grid" by assigning greater significance to nearer

APs. Second, channel busy time (CBT) correlates almost perfectly with full medium utilization ($r \approx 0.97$ in testbed and $r \approx 0.925$ in production) [17]. Thus, Airtime utilization is our second-highest weighted component, leveraging channel busy time to directly quantify how occupied the medium is. However, N_{eff} remains the dominant metric, as it reflects the number of potential interferers and overall network availability, both crucial factors in 802.11 density, which CBT is unable to show alone. Finally, D has the smallest weighting because the number of bits alone does not explain degradation in high density network environments [18]. Yet we include this metric because it reveals the proportion of busy airtime devoted to actual user data transfer, rather than control or management traffic.

Below we define each component of the density score, N_{eff}, U, D, with its corresponding normalization formula. Each equation maps the raw metric into the [0, 1] range based on representative data from Chania's busiest areas. Detailed descriptions of each component are provided in the following sections.

$$S = \sum_{\mathrm{i}=1}^{\mathrm{N_{AP}}} \sqrt{\max(0, R_{\mathrm{i}} - R_{\min})}\ (F_{\mathrm{i}} \geq F_{\text{cutoff}}), \quad N_{\text{eff}} = \min\left(1, \frac{S}{S_{\max}}\right)$$

Where $S_{\max}$ is the maximum proximity weighted AP score we sampled, F_{cutoff} is the minimum number of frames an AP must broadcast per minute to be included, $R_{\min}$ is our lower bound average advertising RSSI, and i runs over every access point from which at least one beacon frame was captured. We apply a square root to $R_{\mathrm{i}} - R_{\min}$ to compress large RSSI differences, to reflect diminishing returns as an AP moves closer. That way strong APs will still have a scaled contribution but will not dominate weaker APs.

$$\eta = \frac{\sum_{k=1}^{M} t_k}{T}, \quad U = \min\left(1, \frac{\eta}{\eta_{max}}\right)$$

Where η_{max} is the maximum airtime we sampled, T is the total time, and k runs over every frame in the capture.

$$\tau = \frac{\sum_{\mathrm{k}=1}^{\mathrm{M}} \mathrm{b_k}}{T}, \quad D = \min\left(1, \sqrt{\frac{\tau}{\tau_{\max}}}\right)$$

Where $\tau_{\max}$ is our maximum sampled throughput, T is total time, and k runs over every frame in the capture. We apply a concave transform to the normalized throughput to have small to medium differences more visible and preventing higher values from overwhelming the metric.

Because our density score is constrained to the [0, 1] range, we must establish realistic maximums for each component so that a "1" truly represents the most extreme conditions one can encounter. To do this, we sampled some of the busiest areas around Chania on a busy touristic Saturday to empirically define each metric's upper bound. At each location, we conducted 30-second sweeps across all available channels to identify the busiest frequency band. Once the peak 2.4 GHz and 5 GHz channels were determined, we performed continuous five-minute captures each to gather our representative data for normalization. We then divided each five-minute trace into five one-minute segments to derive representative bounds for each metric. The Figure 3 table summarizes our sampling choices.

LOCATION	**TIME**	**2.4 GHZ CHANNEL**	**5 GHZ CHANNEL**
OLD STORY CAFE	1 pm	1	40
VENETIAN HARBOR WATERFRONT	5 pm	11	36
INSPOT PROJECT EXTREME	8 pm	6	149
DENSE RESIDENTIAL	9 pm	6	36

Figure 3. Locations, times, and channels sampled.

To determine F_{cutoff}, we computed each AP's per-minute frame rate over the one-minute intervals during which it was present. As shown in Figures 4 and 5, the 2.4 GHz band exhibits a distinct spike in the lower tail only, whereas the 5 GHz band displays bimodal behavior with pronounced spikes in both its lower and upper tails. We then applied a square-root transform to each rate which softened the influence of high advertising APs, sorted the rates in ascending order and accumulated their sqrt-weights until they reached 20 % of the total weight. The rate at this weighted 20th percentile was chosen for F_{cutoff}. We chose a weighted percentile because our beacon-rate data is heavily left-skewed so an ordinary percentile would set the cutoff very low where many Anomalous broadcasters would be included. By weighting each AP's contribution, the cutoff aligns with actual airtime activity, giving a value that counts consistently advertising devices.

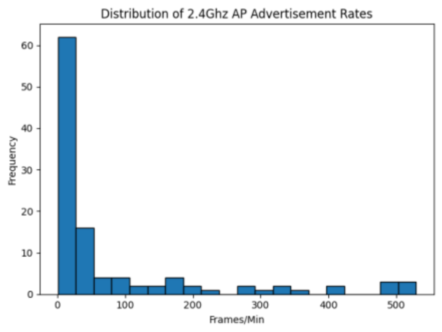

Figure 4.

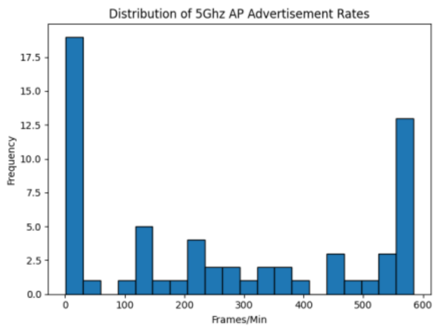

Figure 5.

Next, we identified $R_{\min}$ as the lowest meaningful RSSI, which establishes a baseline which AP signals below are treated as negligible. We set it to the 10th percentile of all AP's per-minute average RSSI across all advertising frames ordered ascending. Figures 6 and 7 illustrate the RSSI distributions for the two bands, with the 5 GHz measurements clustered toward lower signal levels and the 2.4 GHz measurements forming a tighter cluster around mid-range values. This reflects the inherently greater propagation range of 2.4 GHz signals.

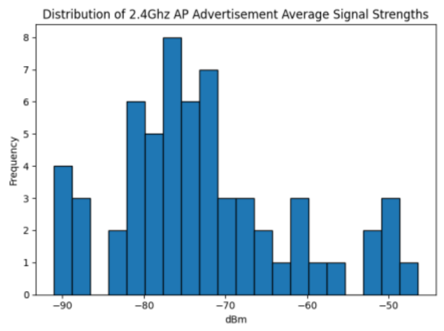

Figure 6.

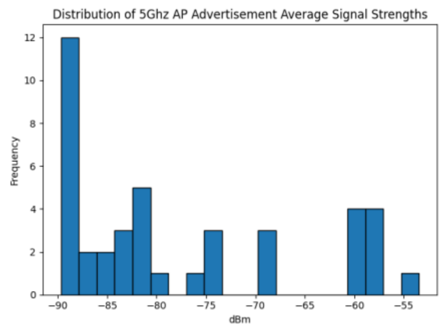

Figure 7.

Using our determined R_{min} and F_{cutoff}, we computed S for each one-minute bin in our data set and set S_{max} to the highest value observed.

We computed η_{max} and τ_{max} straightforwardly, by scanning each one-minute bin for the highest observed airtime utilization and throughput values. Figures 8-11 outline distributions for each metric.

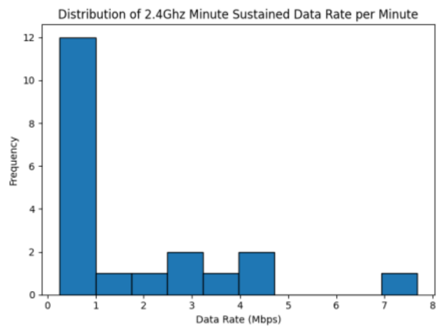

Figure 8.

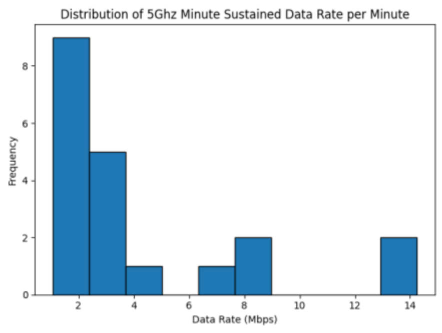

Figure 9.

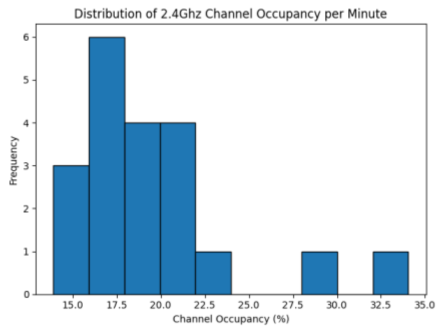

Figure 10.

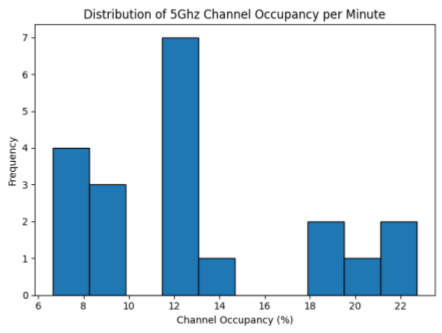

Figure 11.

Figure 12 shows provides a summary table of all our computed values.

BANDS	$R_{\min}$	F_{cutoff}	S_{max}	U_{max}	T_{max}
2.4 GHZ	-86.69	27.2	75.24	34.04	7.67
5 GHZ	-89.37	210	48.46	22.71	14.24

Figure 12. Normalization Constants and Thresholds for Density Score Components

Throughput Algorithm Evaluation

To evaluate our throughput estimation algorithm, we conducted varying scenarios using two devices. One device ran the Ookla speed test to generate maximum throughput over both 2.4Ghz and 5Ghz links, while a second device passively captured all packets during the test. We compared the real-time speed test results against our algorithm's calculated throughput to assess accuracy against a reliable benchmark. Figures 13 and 14 outline all scenarios and observations. The "Speed Test Observations" column has the observed fluctuations of throughput show by the speed test during the experiment. All units are in Mbps.

SCENARIO	OBSERVED	ESTIMATED	SPEED TEST OBSERVATIONS
HIGH RSSI	16.5	27.5	Consistent increase in throughput
LOW RSSI	10.1	14.9	Consistent increase in throughput
MOBILITY FAR	16.3	28.38	Gradual increase despite moving further away from access point
MOBILITY FAR TO CLOSE	14.54	23.27	Gradual increase in throughput as distance decreased

Figure 13. 2.4Ghz Throughput Evaluation

SCENARIO	OBSERVED	ESTIMATED	SPEED TEST OBSERVATIONS
HIGH RSSI	506	457.78	Consistent increase in throughput
LOW RSSI	38.98	52.57	Consistent increase in throughput
MOBILITY FAR	81.72	73.70	Gradual increase despite moving further away from access point
MOBILITY FAR TO CLOSE	74.73	88.64	Throughput capped at lower than expected despite being very close to access point

Figure 14. 5Ghz Throughput Evaluation

The 2.4 GHz results consistently overestimate actual throughput by an average of 62.1% across all scenarios. This bias likely arises because our metric doesn't account for the shared airtime from other devices using the channel. Our metric isolates the frames exchanged between two devices and ignores any simultaneous transmissions, so any additional channel traffic that would otherwise reduce real-world throughput is overlooked. Given that the 2.4Ghz band is generally much more crowded, this likely explains why we overestimate throughput in every scenario. One way to improve this issue would be to correlate transmission blocks with higher-layer requests. This would distinguish whether gaps between transmissions represent separate requests or interference. However, without access to this upper layer context, it would be impossible to know whether *N* bursts belong to one large transfer delayed by contention or if it was *N* distinct transfers, which risks misclassifying true throughput performance.

In contrast, the 5 GHz measurements align more closely with two tests showing a modest underestimation of −9.7% on average, with the other two exhibiting a 26.8% overestimation. The metric's slight underestimation likely stems from an over-accounting of protocol overhead. Overestimation only appeared in the highest-retry scenarios: 23.1% in "Mobility Far" and 20.2% in "Low RSSI". Our current metric treats each retry frame as exactly two back-to-back transmissions and ignores back-off intervals that follow a failed attempt. Consequently, we moderately underestimate airtime in single-retransmission scenarios and significantly underestimate in multi-retransmission scenarios. We could improve this estimate by incorporating inter-frame idle times between our two devices to get the true duration of a successful frame. However, we run into the same issue with our 2.4Ghz improvement where without session-level context, these idle gaps remain indistinguishable from true congestion events.

In the mobility scenarios, our metric reflected throughput changes better than the speed test. Figures 15-18 show our throughput graphed in these scenarios using our application.

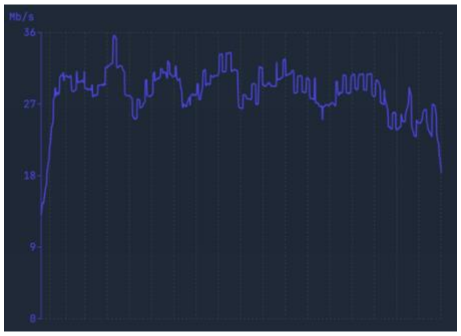

Figure 15. 2.4Ghz "Mobility Far"

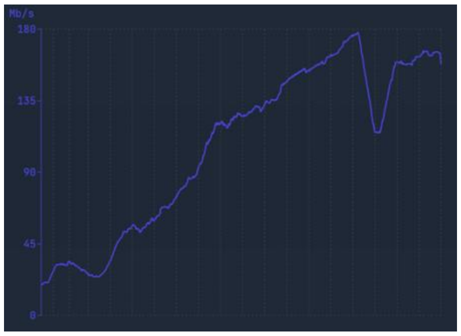

Figure 16. 5Ghz "Mobility Far"

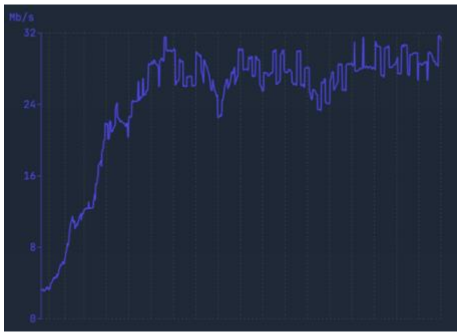

Figure 17. 2.4Ghz "Mobility Far to Close"

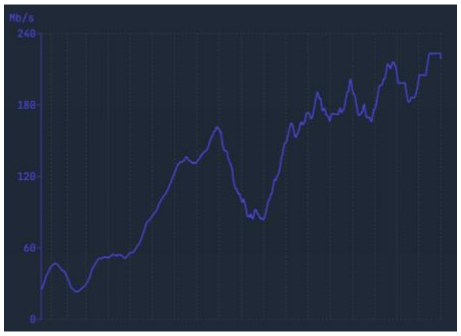

Figure 18. 5Ghz "Mobility Far to Close"

Our packet-level throughput tracks what we expect to see with our throughput during these mobility scenarios. In the "Mobility Far" scenario (figures 15-16), we see a plateau shape which directly mirrors ours physical trajectory. It climbs at the start when we approach the AP, reaches its maximum in the middle when we're closest, and then falls off as we move away again. In the "Mobility Far to Close" scenario, the client started further away and approached the access point over during the test period. We can also see this reflected in upward trajectory of figures 17-18.

Density Algorithm Evaluation

There is no single, objective method for measuring the "density" of an 802.11 network, unlike throughput. To validate that our score reflects network congestion, we test whether it correlates with retry rate, which is a symptom of a dense network [1]. If our density algorithm trends upward with retry rates, it provides evidence that our score captures 802.11 density well.

We captured one-minute traces in the busiest 2.4 GHz and 5 GHz bands at different locations: a dense, multi-story residential street, an internet café, and the Old Venetian Port of Chania. At each location, we took traces at five different times (12pm, 2pm, 4pm, 6pm, and 8pm) to span peak and off-peak usage. We computed our density score and retry-rate on all of these bins to see if a correlation exists.

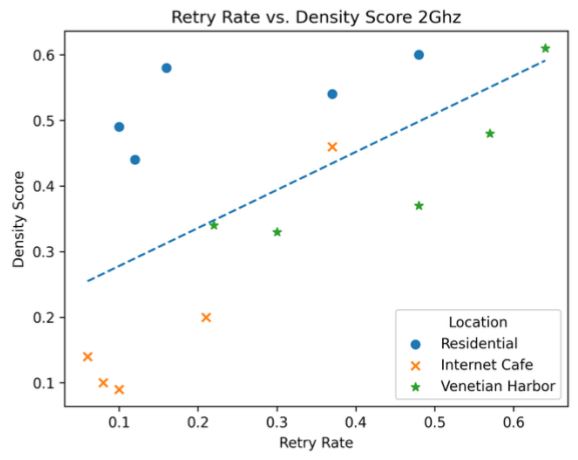

Figure 19.

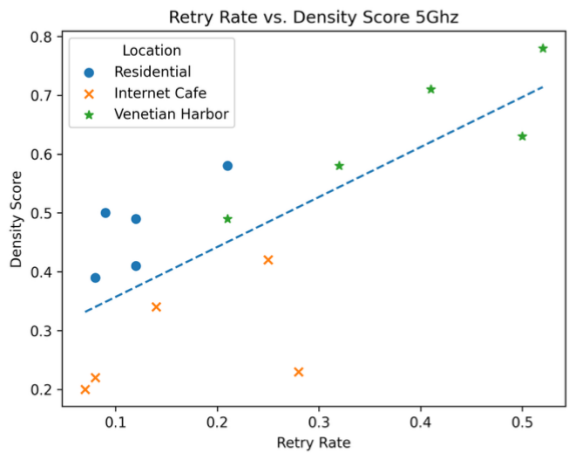

Figure 20.

These results indicate that our density metric is indeed capturing the contention dynamics in real-world 802.11 networks. We found a Pearson's r of 0.61 for the 2.4 GHz band, showing a moderate positive correlation, suggesting that about 37% of the variability in retry-rate can be explained by our density score. We found a Pearson's r of 0.75 for the 5 GHz band, showing a strong positive correlation, indicating that about 56% of the variability in retry-rate can be explained by our density score. These results make sense because the 2.4 GHz is much more susceptible to foreign noise, which would muddy the relationship between our density score and retries. However, 5 GHz has more non-overlapping channels so retries will much more likely be driven by true Wi-Fi contention, pushing our correlation higher.

References

[1] J.-W. v. B. R. S. T. K. and C. H. S. , "Spectrum Utilization and Congestion of IEEE 802.11 Networks in the 2.4 GHz ISM Band," *Journal of green engineering,* vol. 2, no. 4, p. 30, 2012.

[2] OpenJS Foundation, "Process Model | Electron," 2025. [Online]. Available: https://www.electronjs.org/docs/latest/tutorial/process-model.

[3] OpenJS Foundation, "Inter-Process Communication | Electron," 2025. [Online]. Available: https://www.electronjs.org/docs/latest/tutorial/ipc.

[4] OpenJS Foundation, "Node.js v24.4.1 documentation," 2025. [Online]. Available: https://nodejs.org/api/child_process.html.

[5] GeeksforGeeks, "Inter Process Communication (IPC)," 23 April 2025. [Online]. Available: https://www.geeksforgeeks.org/operating-systems/inter-process-communication-ipc/.

[6] KimiNewt, "PyShark | Python packet parser using wireshark's tshark," [Online]. Available: https://kiminewt.github.io/pyshark/.

[7] 7SIGNAL, "WLAN Best Practices Webinar Series: Understanding 802.11 Retries," 25 May 2022. [Online]. Available: https://www.7signal.com/news/blog/understanding-802.11-retries.

[8] R. Nayanajith, "CWAP- MAC Header : Sequence Control," 1 November 2014. [Online]. Available: https://mrncciew.com/2014/11/01/cwap-mac-header-sequence-control/.

[9] S. Ullah, Y. Zhong, R. Islam, A. Nessa and K. S. Kwak, "Throughput Limits of IEEE 802.11 and IEEE 802.15.3," in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, Dalian, China, 2008.

[10] Li, Ping, Yang, Panlong and Yan, Yubo, "Periodicity makes perfect: recovering interfered preamble for high-coexistence wireless systems," *EURASIP Journal on Wireless Communications and Networking,* vol. 2020, 2020.

[11] R. Nayanajith, "CWAP 802.11 PHY – PPDU," 14 October 2014. [Online]. Available: https://mrncciew.com/2014/10/14/cwap-802-11-phy-ppdu/.

[12] C. Lukaszewski, in *Very High-Density 802.11ac Networks Validated Reference Design: Theory Volume*, 2015.

[13] R. Nayanajith, "CWAP – 802.11 Ctrl : RTS/CTS," 26 October 2014. [Online]. Available: https://mrncciew.com/2014/10/26/cwap-802-11-ctrl-rtscts/.

[14] Wikimedia Foundation, Inc., "Short Interframe Space," [Online]. Available: https://en.wikipedia.org/wiki/Short_Interframe_Space#cite_note-:0-1.

[15] G. Combs, "packet-ieee80211-radio.c," [Online]. Available: https://github.com/wireshark/wireshark/blob/4988cca14da57b64a6432d237d3cc208e39e71bb/epan/dissectors/packet-ieee80211-radio.c.

[16] INET Framework, "IEEE 802.11 Frame Aggregation," [Online]. Available: https://inet.omnetpp.org/docs/showcases/wireless/aggregation/doc/index.html.

[17] J. Zahorjan, C. Reis, R. Mahajan, M. Rodrig and D. Wetherall, "Measurement-based models of delivery and interference in static wireless networks," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, 2006.

[18] Acharya, Prashanth, Sharma, Ashish, Belding, Elizabeth, Almeroth, Kevin and Papagiannaki, Konstantina, "Congestion-Aware Rate Adaptation in Wireless Networks: A Measurement-Driven Approach," 2008.

[19] L. Simić, J. Riihijärvi and P. Mähönen, "Measurement study of IEEE 802.11ac Wi-Fi performance in high density indoor deployments: Are wider channels always better?," in *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Macau, China, 2017.