GFiber Wifi Data

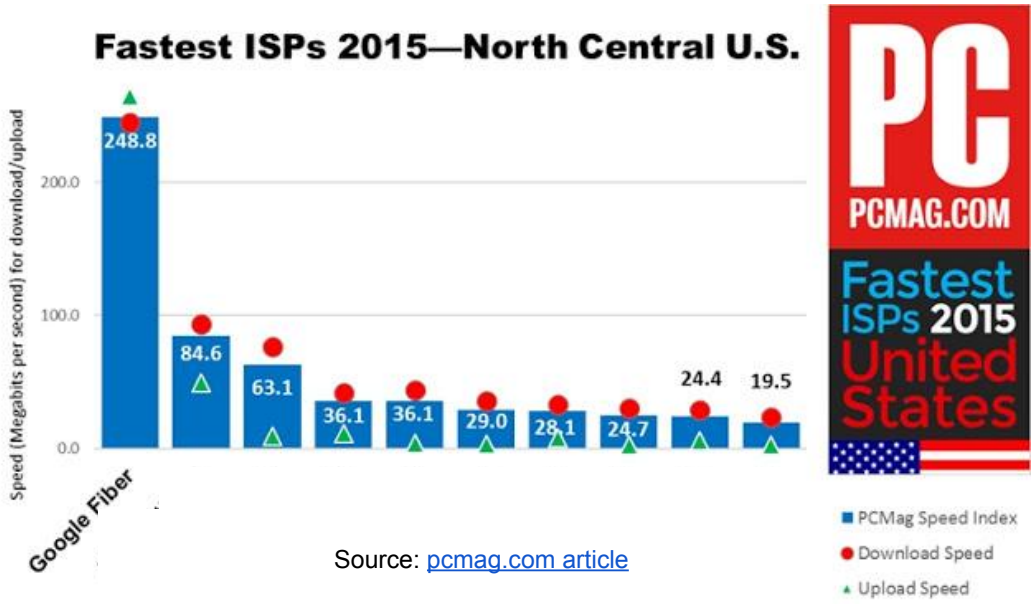
netdev1.1 Seville, Spain

apenwarr@google.com

These are my personal opinions.
They do not necessarily reflect the opinions of my employer.
Not even a little.

Disclaimer: even the parts of these slides that look like facts are actually just my personal opinion. To be facts, I would have to have analyzed the data correctly and then plotted it correctly, and let's be honest, neither of those things are my strong suit.

Who are we?



Source: [pcmag.com article]

Google Fiber is the fastest ISP in the North Central U.S., according to PC Magazine. It's not big enough to count as a national ISP, according to PC Magazine, but if it were, it would also be the fastest ISP in the U.S.

Passive and Active Measurement

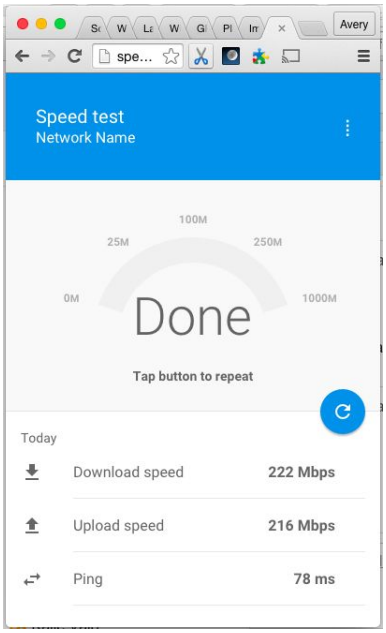
A note on anonymity

- Privacy policy:
  - https://fiber.google.com/legal/privacy.html
- Data stored for a strictly limited time
- Just wifi stats:
  - Don’t log content or Internet endpoints
- MAC addrs are anonymized and IP addrs removed
- Extremely strict access and aggregation controls

Google Fiber collects some data as part of running its network, and you can read about it in our privacy policy (linked above). The data in this presentation is based on the wifi data we collect. Note that, in accordance with our privacy policies, data we collect is time-limited, anonymized, aggregated, and access-restricted according to a bunch of strict rules. Most importantly, while we do collect data about wifi device *types* and performance, as part of these statistics we don’t collect globally unique identifiers that could be used to track a wifi client between homes. Our home equipment also never exports any information about where on the Internet you’re going (ie. the IP addresses of endpoints on the WAN side of the link).

Even the data we do collect, once collected, is kept with strict access controls so only a few people can see it. The exceptions are aggregated statistics, like you’ll see in this presentation, which are obtained by grouping large sets of customers so that there’s no remaining personally identifying information (PII).

HTML Speedtest



What do ISPs use to test performance?

The most basic option is the venerable “speedtest”, like speedtest.net and speedof.me and others.

Unfortunately, traditional speedtests have a few problems, notably:

1) They aren’t optimized enough, so they often can’t keep up with a gigabit, even if the link is fast enough. On GFiber they end up CPU bound instead of network bound!

2) They use flash rather than pure HTML, which means they often don’t work on mobile devices without installing an app.

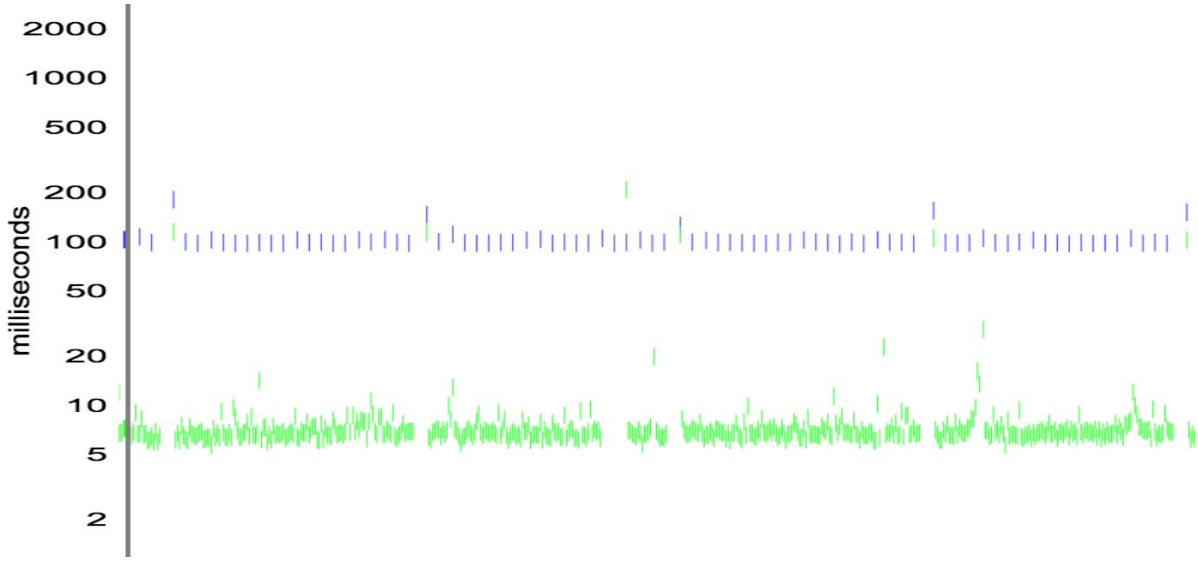
3) They test Internet speed and wifi speed together, confounding the results. It’s hard to tell which one is the bottleneck. (Spoiler: on GFiber the bottleneck is wifi :))

We wrote our own mobile-friendly HTML speedtest to deal with problems #1 and #2. #3 is a little trickier, so we need different tools for that. (You’d think we could solve #3 by just running a speedtest server on our home wifi router, right? Unfortunately not. Our wifi router’s CPU isn’t actually fast enough to generate traffic at a gigabit, even on a wired connection. It can forward a gigabit only because it has a hardware forwarding fast path. So oddly enough, a local speedtest would often be *slower* than an Internet speedtest if we used that method.)

Anyway, we have more tools. Hold that thought.

gfblip.appspot.com

Open source: http://github.com/apenwarr/blip

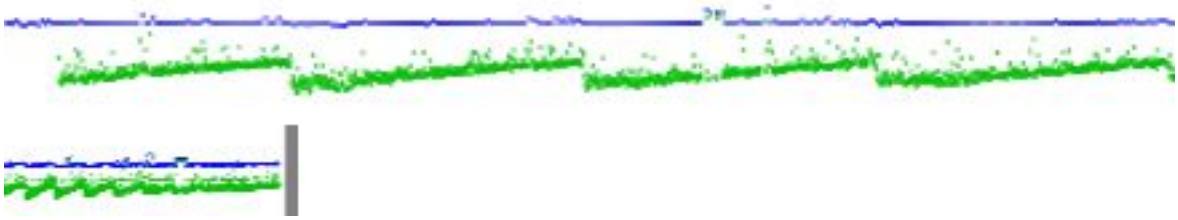


Blip is a pure-html tool that runs a series of HTTP requests and reports the latency of each request. (Each "blip" on the graph is one HTTP request.) Interestingly, once a TCP session is established, every request in an HTTP/1.1 session can reuse that connection, and a short request+response is just one packet round trip, which means the network latency is the same as ICMP ping. What luck! There's some additional latency added by the web browser, javascript, etc, but I think it's reasonable to include that in the chart; after all, they're part of the user-visible latency of viewing a web page.

Note that because blip uses TCP, packet loss looks like "high latency". There's no such thing as a missing blip, just one that TCP had to retry a few times (with a backoff timer).

The green blips are the local router. The blue blips are to "the Internet" (somewhere in California in this case, while I was in New York). If the green blips look fine but the blue ones don't, then the Internet is your problem. If both blips are upset, then the local link (usually wifi) is the problem. If green is bad but blue is good... well, that shouldn't happen. Watch out that the Y axis is logarithmic though, so a particular visual height of delay in the green line means something very different from the same height on the blue line.

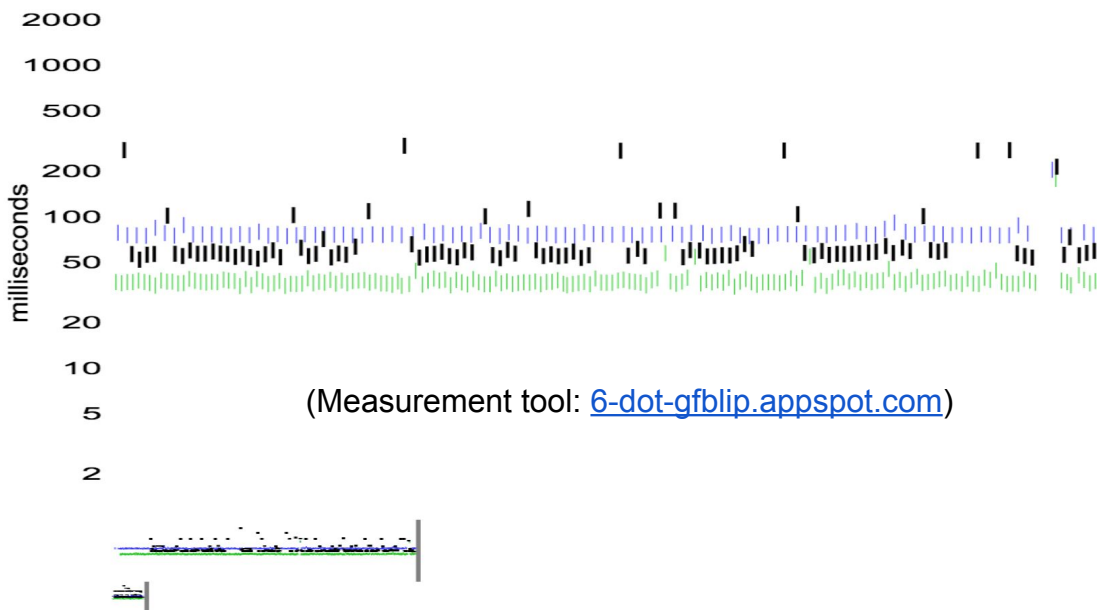
gfblip: weirdly variable ping times



Here's an example of something weird we observed while debugging a particular wifi driver on a particular embedded chipset . Notice how the green blips trend slowly slower and slower, then suddenly jump down again, and repeat.

In this particular case, it turned out to be a timer-related bug in the router we were pinging. It had something to do with interrupt handling latency, I think. Once you get the hang of gfblip, you can identify various different kinds of problems at a glance, which is really handy.

gfblip: flakey DNS



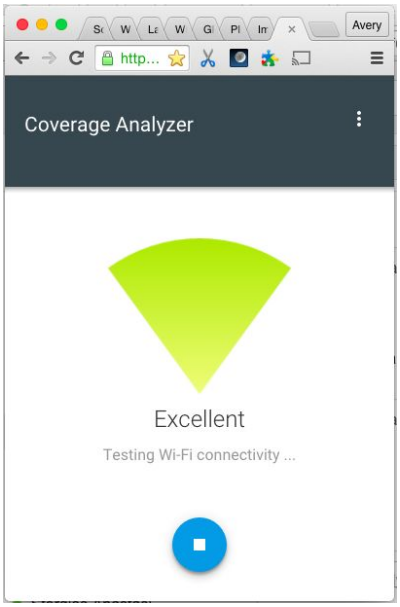
(Measurement tool: 6-dot-gfblip.appspot.com)

A newer version of blip (not quite released yet because I've been too lazy to do the last few steps) is also capable of testing DNS: that's the black blips. Basically, it points at an Internet domain that uses wildcard DNS (*.test.whatever.com) which all points at the same static IP address. Each blip uses a randomly-generated name in place of the *, which forces a DNS lookup and (depending how your browser is implemented) maybe a new TCP connection to be negotiated.

It turns out that *many* supposed wifi problems are actually DNS problems. The above chart shows what that looks like. The green and blue blips are nice and stable, but the DNS server is flakey and very often needs one or more retries before returning an answer. That translates directly to slower browsing performance in real life, no matter how slow your Internet link is. Some especially buggy routers can get into a bad state where every single DNS lookup takes 1000-2000ms! That can be hard to diagnose normally, but blip makes it extremely obvious to see, if you know what you're looking for.

The new DNS-enabled version of blip is not exactly open sourced yet, but you can run it at the link above. Contact me if you want to pester me to open source it; it's not hard, I'm just lazy.

HTML Coverage Analyzer

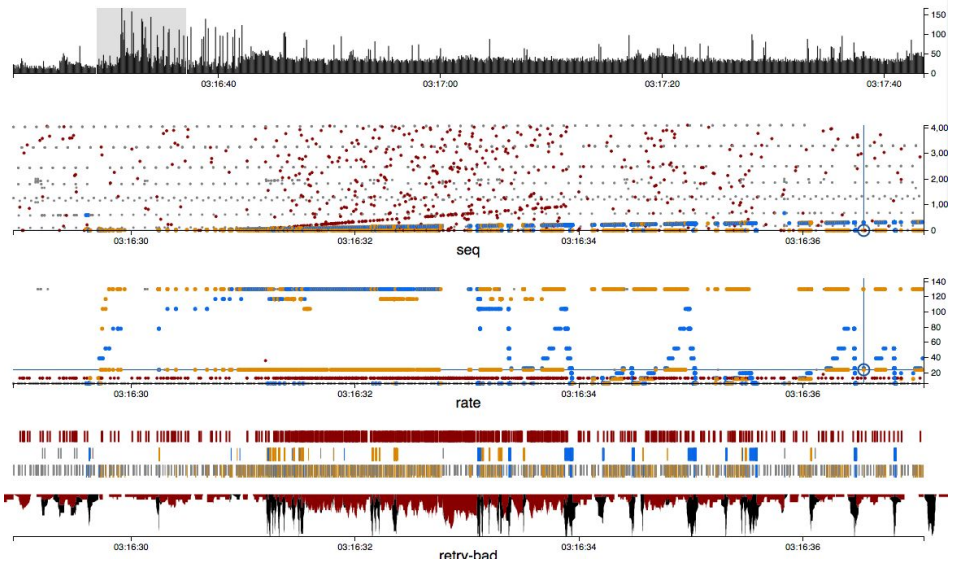


The same team that produced our HTML speedtest also produced the GFiber coverage analyzer tool. It's basically a reskinning of blip to be easier to understand. You can't quite diagnose the same variety of stuff, but it can tell you in real time whether your wifi coverage is good or bad.

The idea is that you can load up this page on your phone or laptop, then wave it around the room to try to find dead spots. The latency-based measurement is very fast, usually reporting changes in less than a second, which makes it really pleasant to extensively test coverage, rather than doing a multi-second speedtest at each location.

Plotting wifi packets: wavedroplet

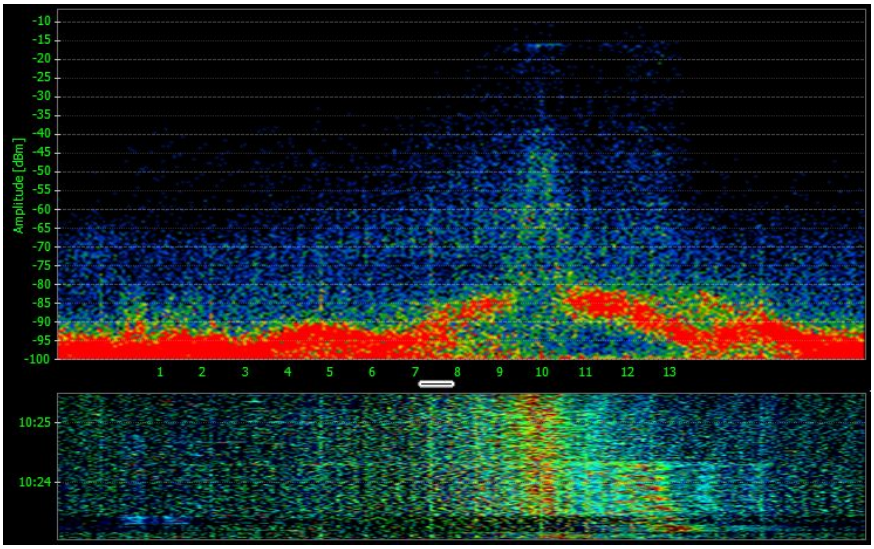
Open source: http://github.com/apenwarr/wavedroplet



Wavedroplet (waveform dropbox applet) is a wifi packet capture analyzer tool that I and a few people have been working on when time allows. When you want to see at a glance what's going on with a particular wifi capture, wavedroplet is good for that. For example, the 'rate' section (blue/orange dots) in the middle of this chart shows some pretty poor rate control behaviour on the part of blue. That seems to be a PID rate control algorithm, which has been discredited for about a decade, but apparently some vendors still haven't gotten the memo. Mr. Orange is using minstrel, by the way. He works a lot better.

Anyway, enough about analyzing pcaps. Ain't nobody got time for that when you have zillions of customers.

Background spectrum analyzer



Here's another thing you can do: a spectrum analyzer captures the wifi frequency spectrum, over time, across all the channels. This is most important on 2.4 GHz wifi, where most of the interference is. This is a visual depiction of a spectral analysis, which is okay for looking at one situation at one time, but like the pcap analyzer, doesn't really help when extended to zillions of customers.

Background spectrum analyzer

Open source: https://gfiber.googlesource.com/vendor/google/platform/+/master/spectralanalyzer/

```
# period of low wifi traffic in a Google office
fft- 1:   94    2             1    1
fft- 2:   95    1
fft- 3:   95    1
fft- 4:   95    1             1
fft- 5:   95                  1
fft- 6:   94                  1              1
fft- 7:   94                                 1    1
fft- 8:   93    1    1        1                   1
fft- 9:   92    1    2                            1
fft-10:   91    3    3        1                   1
fft-11:   89    4    4        1
```

...so we adapted some open source code which lets you take an ath9k wifi chip and use it as a spectrum analyzer.

The output is:
- Each line represents a capture of one channel during the time period
- Each column represents a power level visible on the channel
- Each value is the percentage of time during which the channel was at a given power level.

The sum of each row should be about 100%, other than rounding errors. In this example, mostly the power level is minimal (first column) almost all the time.

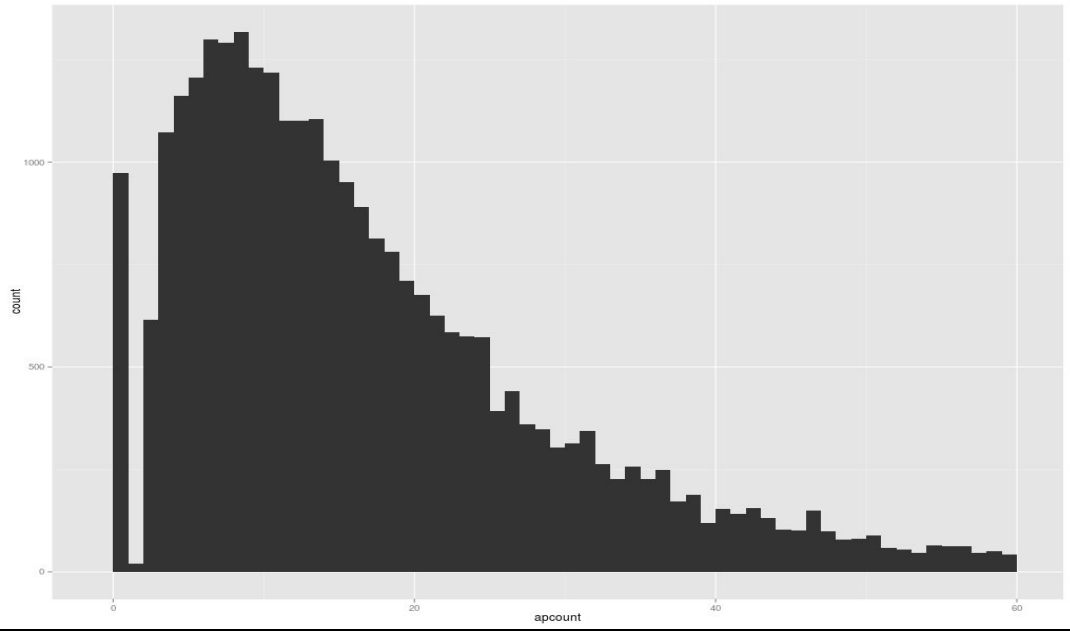
Background spectrum analyzer

```
# period of high wifi traffic in a Google office
fft- 1:  88       4   2   1
fft- 2:  84   1   5   5   2           1
fft- 3:  72   2   5  10   3       1   2
fft- 4:  64   2   6  11   8   1   1   2   1
fft- 5:  59   2   6  13   9   2   1   2   2
fft- 6:  53   2   7  13  11   2   2   3   3
fft- 7:  61   2   6   6  12   3       3   3
fft- 8:  82   2   5       2   2       1   3
fft- 9:  84   2   4       2   1       1   2
fft-10:  87   1   4       1   1       1   1
fft-11:  87       8       2                   2
```

Here's the spectrum analyzer showing much busier channel activity.

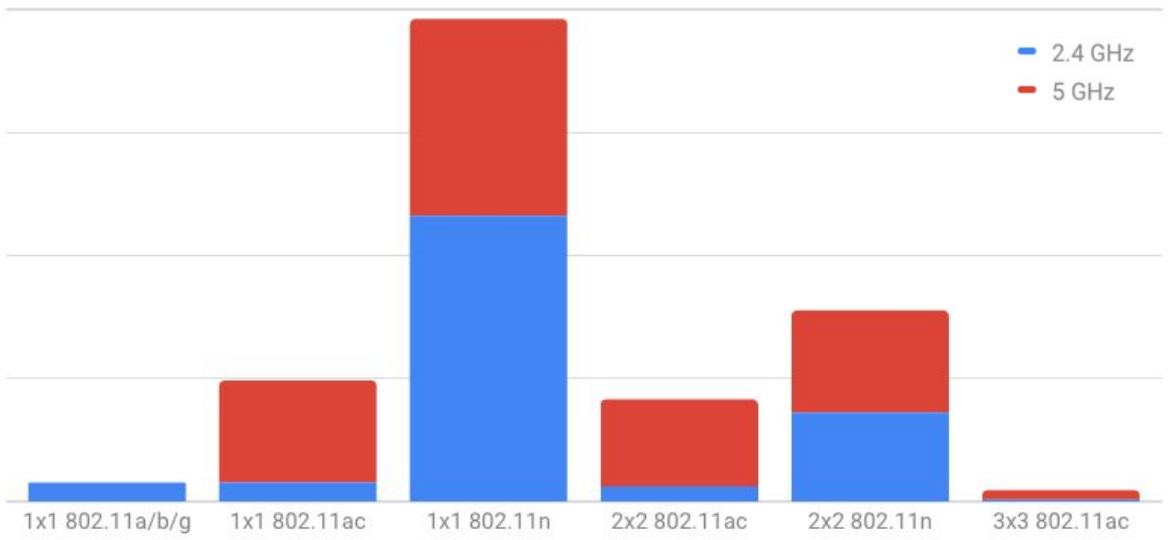
We can capture this sort of data across all devices in our fleet, and correlate it with observed wifi performance. We can also try to use it as a better wifi channel selection algorithm... but we don't yet.

Number of nearby visible access points



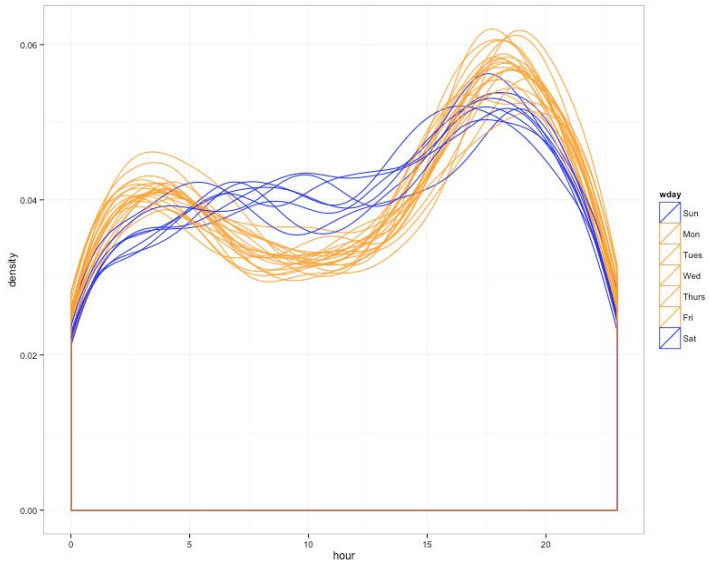
Speaking of collecting data across the whole fleet, here's a similar measurement: a histogram of the number of visible access points, from the point of view of each access point in a subset of GFiber customers' homes. The modal value seems like about 8. But some people really have a *lot* of access points in view!

Real-world device capabilities



Here's a good one: fraction of client devices with particular capabilities. Some things to note:
- There aren't very many 802.11b/g client devices remaining (yay!)
- 1x1 devices are by far the most popular (presumably phones and cheap tablets)
- Many 802.11n-only 1x1 devices are on 2.4 GHz, but 802.11ac devices mostly end up on 5 GHz. (Note that the chart reports that frequency the devices are *on*, not their capability. Arguably, any 802.11ac device on 2.4 GHz is being inefficient.)
- There are virtually no 3x3 devices (almost all of them are recent Macbook Pros). Only 3x3 devices can take advantage of the maximum speeds of your fancypants expensive router. It's not so inspiring.
- All those new 4x4 access points? There aren't any clients at all that support that, across the entire set of customers we sampled. 4x4 can maybe benefit a bit from MU-MIMO, but the advertised maximum speeds are never going to happen in real life.

Number of stations, by time of day (original)



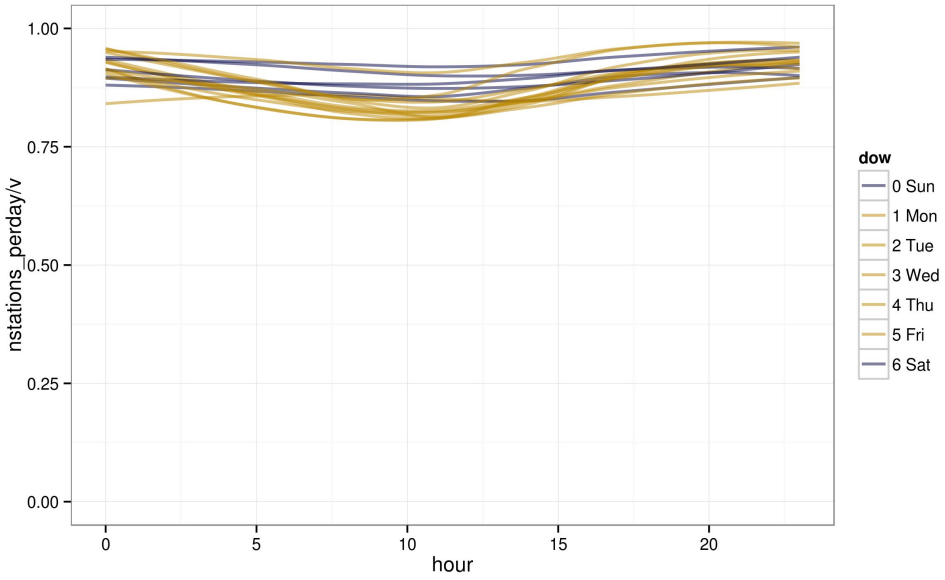
Another interesting chart is the number of client devices that are visible at particular hours of the day. Here's the first chart we made of that, way back when. The blue lines are weekends, and the yellow lines are weekdays. Note how on weekdays, people tend to leave home in the middle of the day (number of devices declines) while on weekends they don't as much. Primetime (evenings) have the greatest peaks however.

Quiz question: why does the number drop so precipitously at night?

Number of stations, by time of day (original)

We had a hypothesis about the reason. I was able to disprove it “scientifically” using the power of Google+ Polls, thus helping that team recoup development costs.

Number of stations, by time of day (fixed)

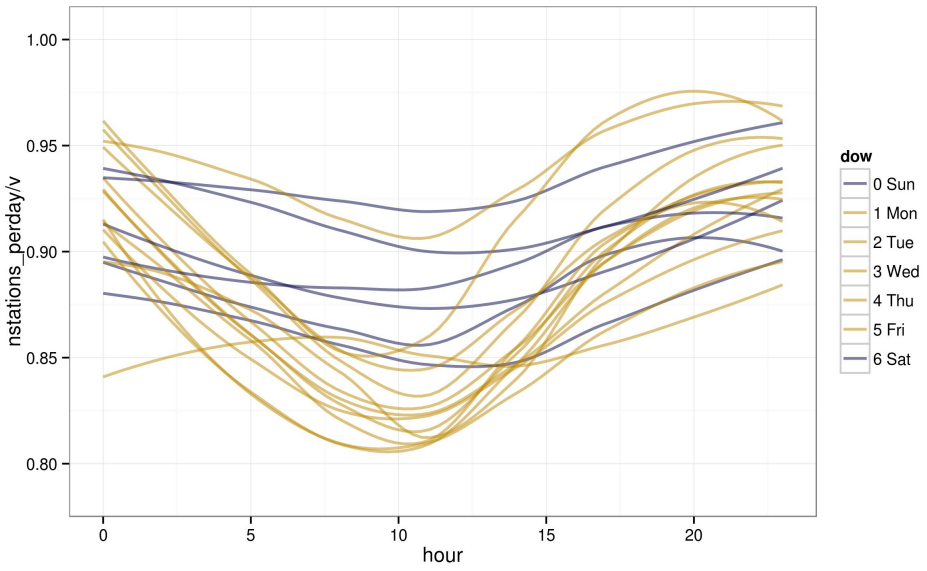


Answer to quiz on previous slide: it doesn't drop precipitously at night, actually. Turns out our original averaging algorithm was wrong. But it was educational for us to experience the process of rationalizing results to meet our worldview.

Turns out the usage patterns are more like this. The midday dropoff on weekdays is still visible, but the midnight dropoff no longer is.

But it's hard to see. Let's zoom the y axis...

Number of stations, by time of day (zoomed)

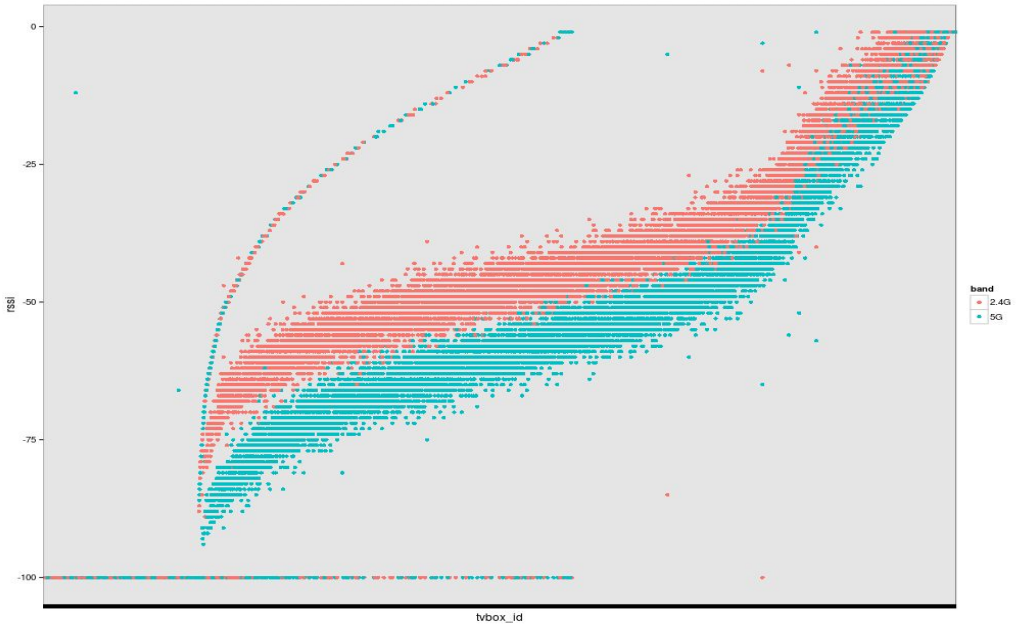


Here's a zoom on the part of the y axis in question. Note that it no longer goes down to 0.0.

Now we can see that there is definitely an overall dropoff overnight and into the morning, but people come back again in the afternoon and max out in the evening ("prime time").

There's one stray orange (weekday) line near the top, which didn't drop off like the others during the day. What's that about? ... turns out it was a weekday holiday! So I guess the system works.

Distribution of RSSI samples for each TV box



I'm going to show some of these graphs mainly because they look cool, not because they're super useful. Sorry.

For this one, I took a sample of GFiberTV boxes, which all contain wifi chips, and had them look for their GFiber Network Box (wifi router), and report back the signal strength they observed over the course of several days. Each point on the X axis is a different TV box; each dot is a sample; the y axis is the observed signal strength. Then I sorted the TV boxes in order by median signal strength, because that was the one that looked coolest when it was done.

Some things to note here:

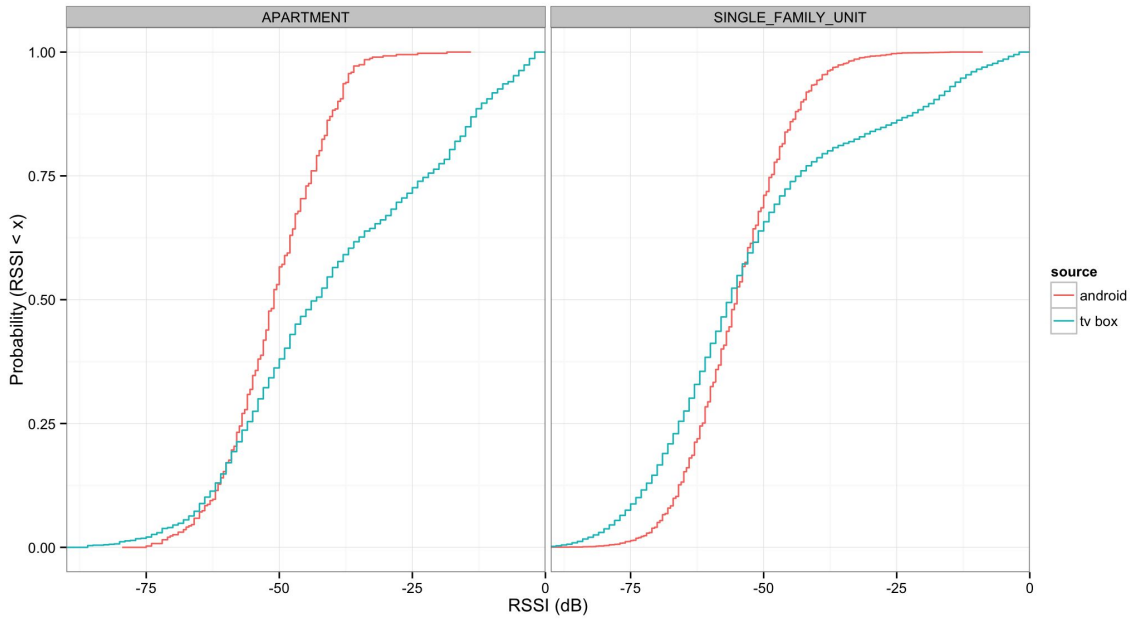
- At each x location, you can see a vertical range of observations. This is, essentially, the error in RSSI measurement. It looks like +/- 5 dB (total range of about 10 dB) for most of the range. Don't trust anyone who tells you about seeing a 3 dB difference unless you average it over a long time!

- The red lines (2.4 GHz) are consistently somewhat higher than the blue lines (5 GHz), though they get closer together at the high end of signal strength (toward the right). This is consistent with a somewhat faster dropoff of 5 GHz at longer ranges, and/or lower quality 5 GHz antennas.

- What's that horizontal line at -100, and the curved line up top? Those are just artifacts of a weird measurement bug. When we got an invalid measurement, I set

the RSSI value to -100 so it would show up, but obviously be separate from the rest of the chart. But that, in turn, caused a drop in the median RSSI, which caused the TV box in question to be shifted left on the x axis from where it normally would be. That curve at the top is just perfectly normal samples that should have appeared somewhere further to the right, where they would have blended in with other TV boxes at similar signal strength. It looks weird, but I kept it because I think it says something deep about statistics, such as “why are these shades of blue and red so much prettier than the ones in the next slide?”

How many TVs at each range?



Here's another way to look at the data, using a cdf (cumulative distribution function). In this chart, a line shifted to the right means higher average signal power.

In this case, the left box is for apartment buildings, and the right box is "single family units" (ie. houses). The blue lines are TV boxes observing their central access point, and the red lines are Android devices with our mobile TV app installed, observing their central access point.

It's interesting to look at the signal strength patterns:

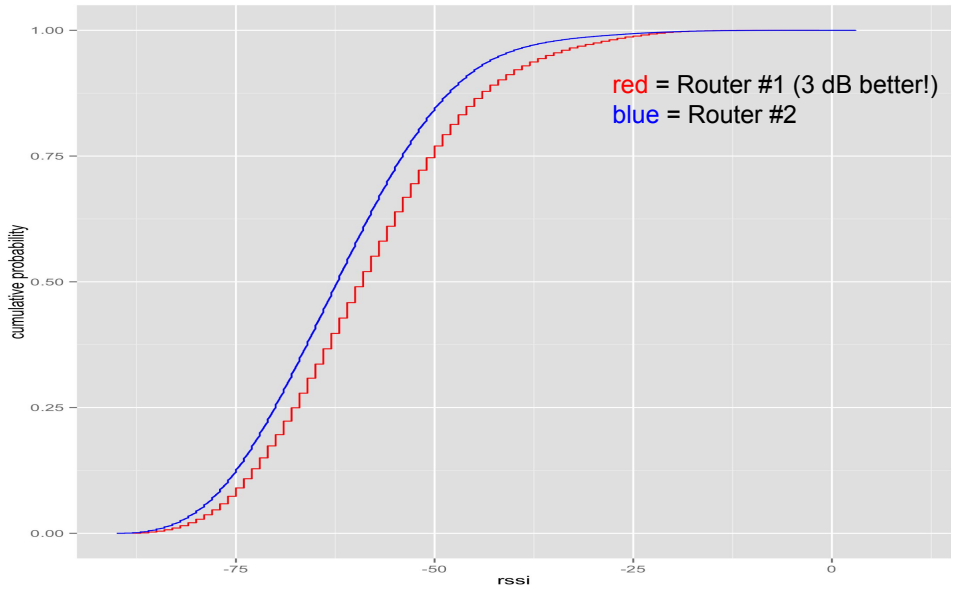
- There are not very many android devices very close to the access point (which makes sense; who sits 10cm away from their access point?). However, there are quite a lot of TVs that get *very* close to their access point. So many, in fact, that it's clear we should disable the wifi in such TV boxes and use a (very short) wired connection instead.

- In apartments, the TV boxes seem to be distributed fairly evenly at various distances. In single family homes, there are some devices close to the access point (gently sloped line toward the right) but more devices are further away (steeper line toward the left). This makes sense, because people in large homes have more TVs, and on average, most of them are not right next to the AP.

- Android devices generally don't show as low an RSSI as the TV box devices do, at the longest range. My theory is that this is because Android phones drop off the wifi

and switch to cellular when their signal gets that weak.

Comparing signal reception on different hardware



Here's another fun example. We make several slightly different hardware variants of routers. We capture the RSSI of observed wifi stations on each one, and can plot them in a cdf (cumulative distribution function), as shown here. In a cdf like this one, a line shifted to the right is "better" (higher signal strength).

This particular example showed that the two routers we were comparing had a 3 dB typical difference in reception quality. The red router was "better", which agreed with qualitative reports that people were getting better range with that hardware than the blue router, even though they performed theoretically the same in a test lab. That led us to investigate the difference in hardware designs, which we might not have known to do otherwise.

(Note: you have to be careful when comparing RSSI like this. RSSI is generally a relative measure, not an absolute measure, and depends on device type and calibration. In this case we knew the wifi chipsets were the same, so we believe the RSSIs are directly comparable like this.)

isostream and isoping

Let’s look at some more active ways of testing customer wifi, rather than just passive observation.

isostream: fixed-rate TCP over a long time

Open source: https://gfiber.googlesource.com/vendor/google/platform/+/master/cmds/isostream.c

isostream (isochronous streamer) is a program that sends TCP data from an access point to a TV box at a particular desired bitrate. This is intended to act like a typical TV video stream.

While testing, it watches for “dropouts” (time periods where the stream inevitably falls behind schedule). Dropouts have three main characteristics:
- width: the duration of the outage (ie. how long until we have caught up again)
- depth: the severity of the outage (ie. how many seconds behind schedule did we fall behind and thus need to recover)
- frequency: how often we fall behind.

This plot shows a few sample isostream sessions in different colours, plotted over each other. You can see that there are very few outages over several hours, but there are definitely some. The worst is the leftmost (purple) outage, which has a narrow width (not possible to measure on this plot, but it happens to have been just a few seconds long) and about 0.7 seconds depth. That means we had to run faster than the nominal bitrate for a while, until we recovered an extra 0.7 seconds of content.

Based on an isostream analysis, we can determine how well wireless TV stream delivery will work given a variety of factors. One of the most important factors is how big the buffer has to be! In this case, at least 0.7 seconds, the depth of the deepest outage. (The math gets a little more complicated if outages are frequent but brief. For example, what if a new outage starts before the previous one has fully recovered?

We might have to plan for outages deeper than the one we have ever actually observed.)

isoping: upstream and downstream, separately

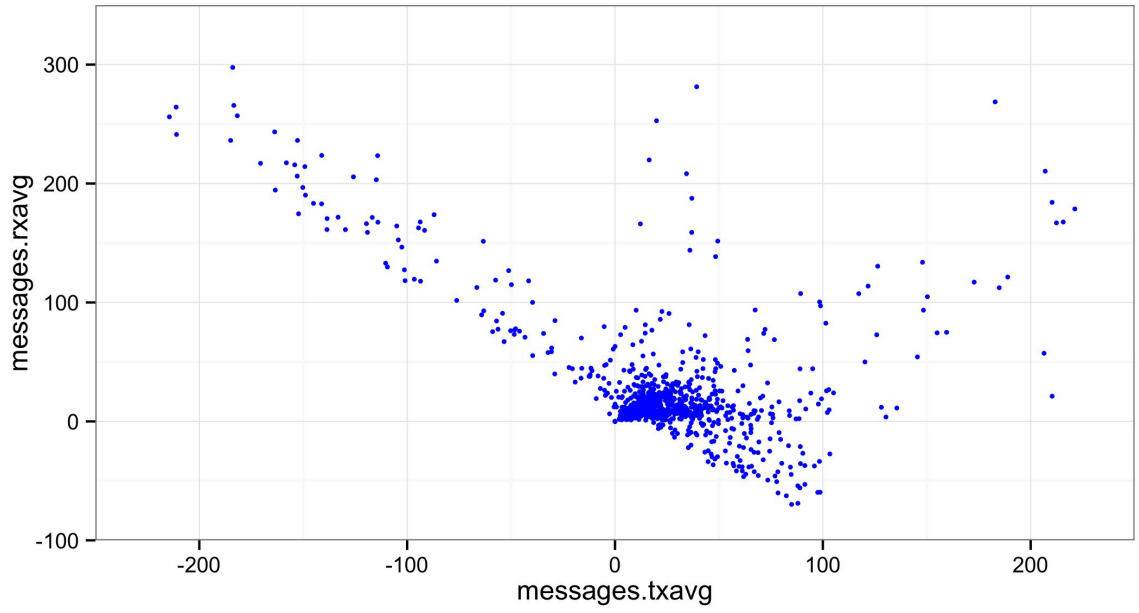
Open source: https://gfiber.googlesource.com/vendor/google/platform/+/master/cmds/isoping.c

- Client sends first packet including interval request
- Both ends send a packet at each interval
  (TTL limited to avoid amplification attacks)
- Content of each packet is a list of receive timestamps
- Allows measuring upstream/downstream latency and loss separately!

isoping is another fun program, whose operation is described non-visually above.

It's hard to overstate the importance of measuring wifi performance separately in the upstream and downstream directions. Almost always, your wifi problem is in one direction or the other, and it saves a lot of effort if you can figure out which one. isoping can also be useful for debugging wired network problems; for example, it makes it easy to see that bufferbloat on a DSL line is usually mainly in the upstream direction.

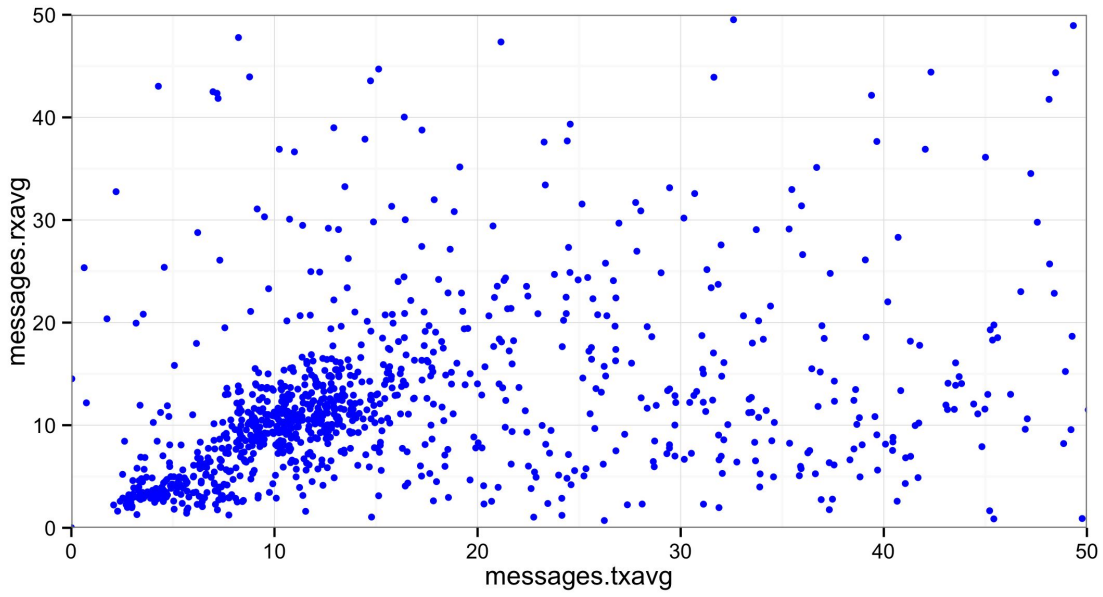
Latency: isoping from GFiber android app



We built an isoping client into our GFiberTV android app. Unfortunately, some versions of it have a bug where it accidentally introduces an offset of multiples of +100ms in one direction and -100ms in the other direction. The net result are the negative values of tx and rx speeds seen in this plot, resulting in the visible trendline of y=-x (down and to the right) for some of the dots.

The effect of this bug is that, for now, we can't really use any values except the ones that are definitely 0..100 on both scales. Almost certainly, some of the values >100ms are real, indicating real latency problems in one direction or the other (possibly caused by congestion, interference, etc), but we don't know which ones, so let's ignore them for now...

Latency: isoping from GFiber android app (zoomed)



Here's a zoomed version of the "interesting" (and I think non-buggy) isoping data.

The main thing we can observe is that tx and rx latencies are fairly well correlated (not so shocking). However, there are definitely many situations where tx is much greater than rx latency, or vice versa. We could someday try to break these situations down by time of day, device type, other background wifi activity, etc, but haven't tried to do that yet.

As we investigate how to keep wifi latency and jitter to a minimum (discussed later), this sort of measurement will be pretty important.

Wifiblaster

Okay, if you thought isoping was cool, you should see wifiblaster!

Wifiblaster: one way tests without installing anything

Open source: https://gfiber.googlesource.com/vendor/google/platform/+/master/wifiblaster/

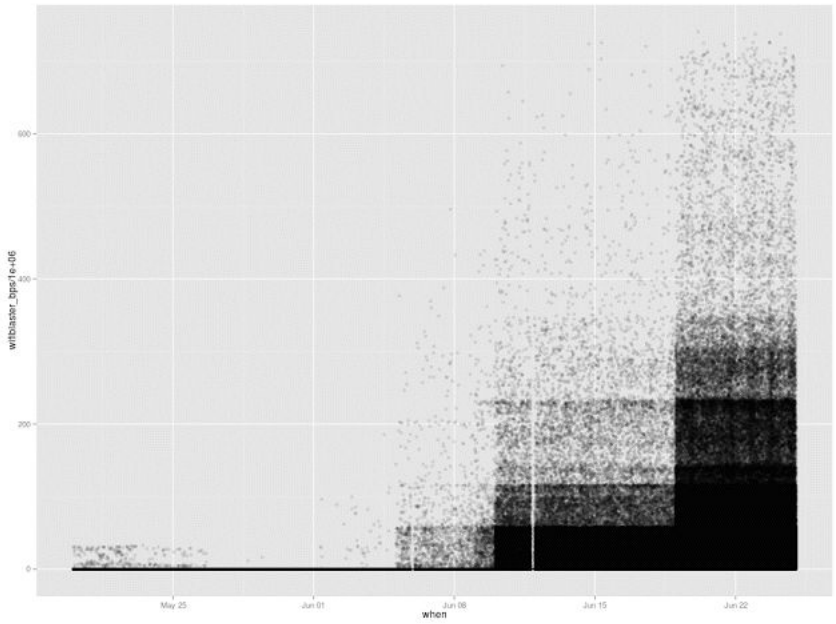
- Uses kernel pktgen
- Send invalid packets (wrong IP, protocol number, etc)
- Station discards them but still sends wifi BlockACKs (even if the kernel stays asleep!)
- Unbiased, periodic, downstream rate measurement

The really cool feature of wifiblaster is that it can work even if the wifi station has no special software installed. Using it, we can measure downstream throughput on any kind of device, even if no users are present. That means we can run tests that aren't biased by time of day, type of device, type of user, etc. This is pretty powerful. The HTML speedtest, for example, gives reliable results, but is highly biased: it only works on devices with a web browser, and it only runs when users run it on purpose. They often run it because they're wondering why things suck so badly, which is a sampling bias. They only run it when they're awake, which is a time of day bias. And so on.

isoping only runs automatically on Android devices that have the GFiberTV app installed. That eliminates the time of day and problem bias, but it only runs on Android devices, which is a severe bias too.

The wifiblaster avoids all that and should be the closest thing to an "objective" wifi speed measurement as we can get.

It’s art!



This is another one of those useless charts I included because it’s interesting. The x axis is time, the y axis is packets per second, and each dot represents one measurement from one device. I used an alpha blend for each dot so that more dots in a particular location make it look “darker” (up to a point).

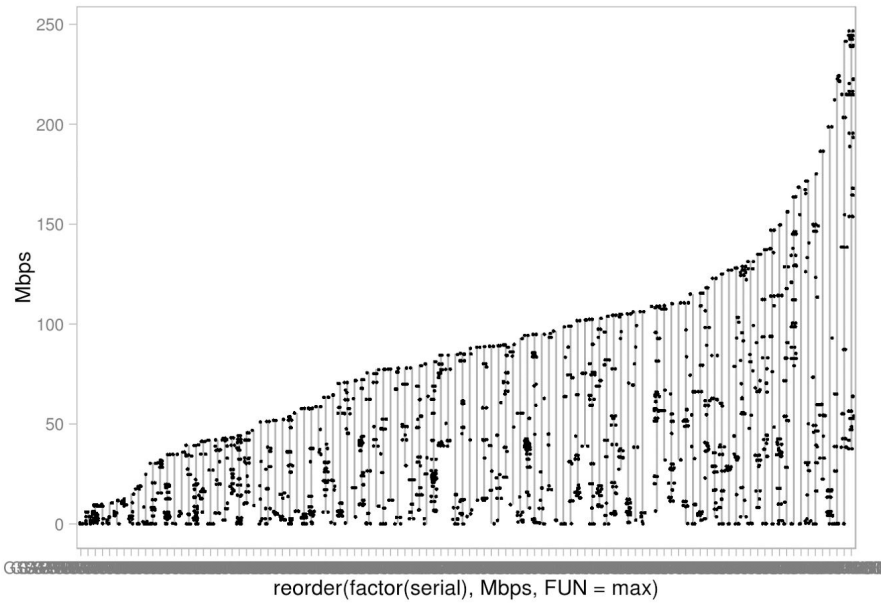
Why the stairstep pattern as we go forward in time (to the right)? Well, because we were activating more and more devices in the wifiblaster test during this time period. Vertical areas which were previously just lightly shaded got filled in, more and more darkly, as we ran more tests per day.

Well, okay, that explains the increase in total darkness as we go to the right. But why the stairsteps?

Good question. Those turn out to be because not all speeds are created equal. There are certain performance maxima: for example 802.11ac 1x1 devices have one maximum speed, 2x2 devices have another, and so on. We might expect any given device to return a “random” speed between 0 and max, for that device. So the 1x1 region has the most dots (because there are more 1x1 devices, but also because 2x2 devices have some of their samples show up in that region) and so on.

Anyway, the point is that it looks cool. I think I should submit it to an art museum somewhere.

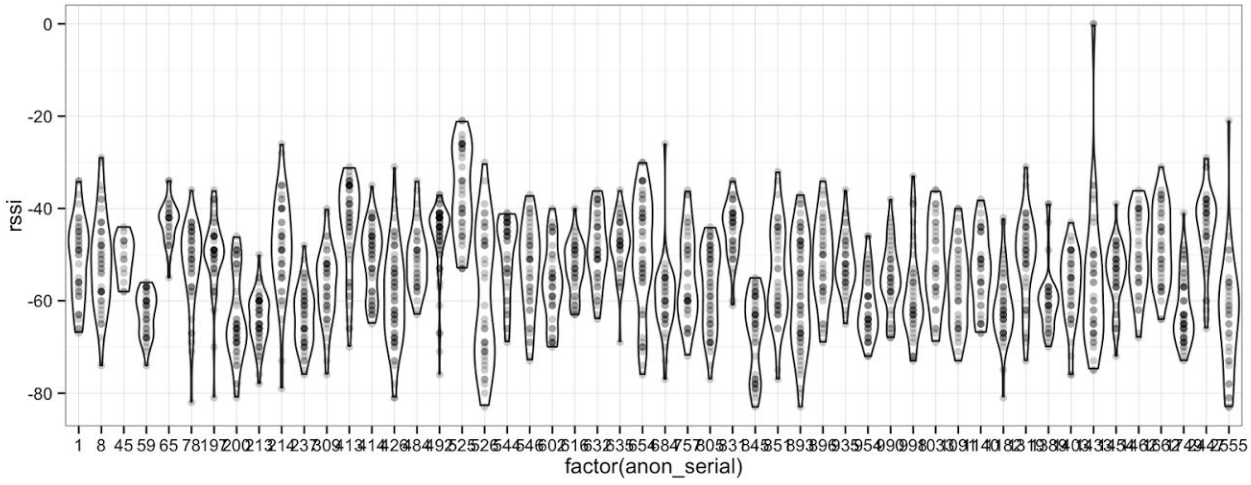
Per-device variability



Here's another way to look at a similar data set. Here, we're back to the x axis being one position per access point (and not sampling all that many access points). Each dot is a sample from one station attached to the given access point. All the samples from a given access point are connected by a vertical line so you can see them visually. We sorted the x axis by the maximum speed ever achieved by that AP, just for fun. (The net result is that a steeper slope means fewer devices had that maximum speed. Naturally, the highest speeds were the least common.)

Some APs clearly have much less variance than others. But since we've mixed multiple stations together in the same chart, the results aren't that meaningful. Just sort of fun looking.

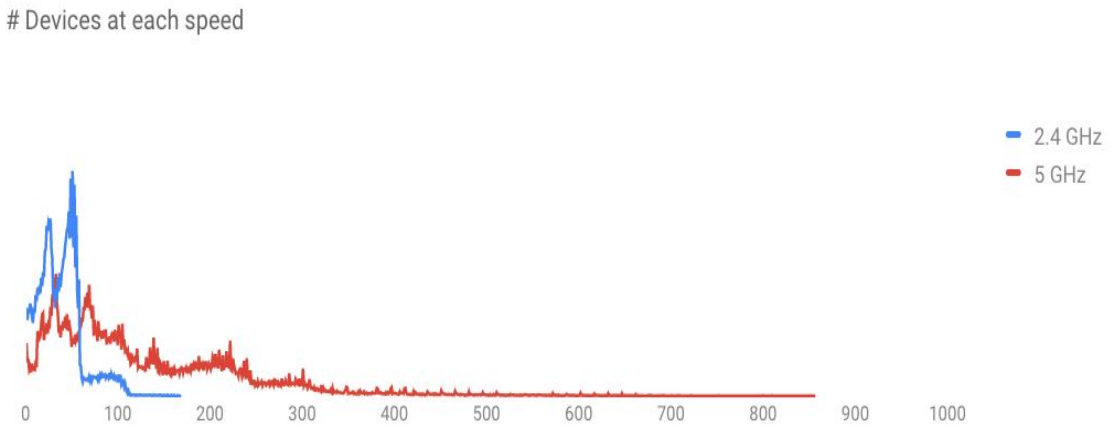
Paramecium plots



Here's the same thing only displayed as if each AP were a single-celled organism, and plotting RSSI on the y axis instead of speed.

(These are technically "violin plots" but when I do them, they never seem to come out looking much like violins.)

Real world wifi speeds

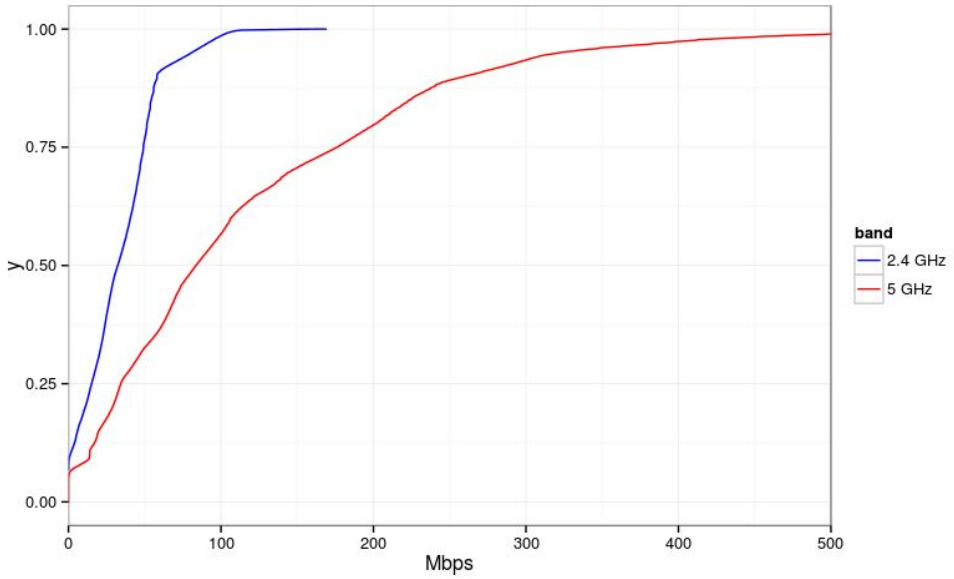


Okay, those previous slides were fun, but not that useful. Here's a simple view of what we actually see in real life. Notice the peaks around the max for different device types (1x1 802.11n, etc).

Major things to note in this graph:

- 2.4 GHz is much slower than 5 GHz
- Peaks for 1, 2, 3 antennas
- Very long tail of fast results up to the (almost but not quite impossible ~850 Mbps), but virtually everyone is much slower than that.
- Trying to optimize for peak performance is not addressing the needs of most of our real customers.

Real world wifi speeds, cdf edition

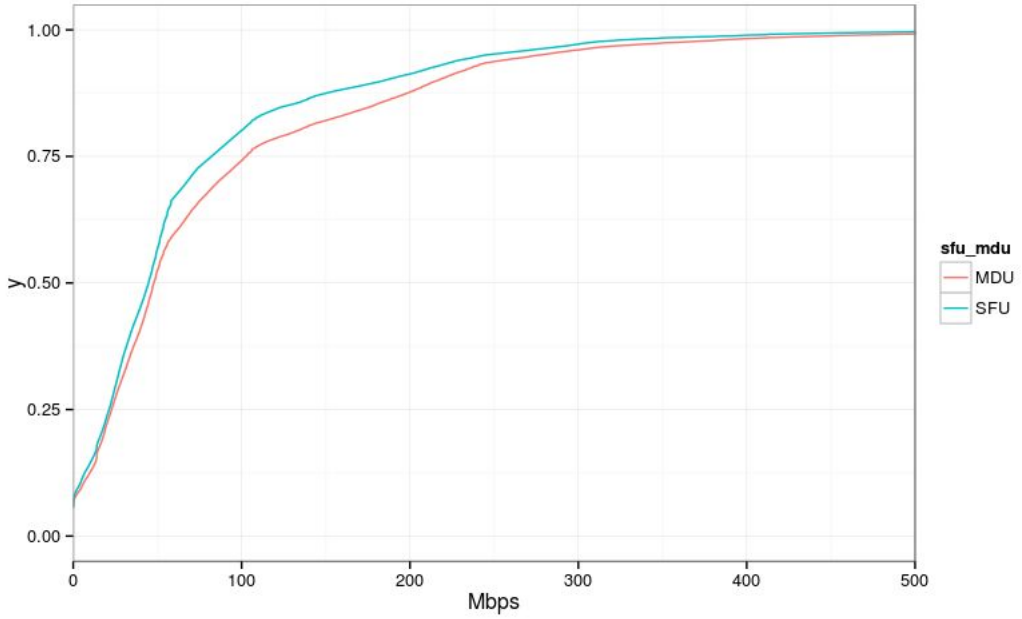


That last plot seems to make a lot of people happy, but it can be a little misleading, as all histograms can be. Here's a cdf (cumulative distribution function) version that tries to be a little more precise. As usual in our wifi cdfs, a line that's more too the right means better/faster.

Not so shockingly, we see that 5 GHz devices are faster across the board than 2.4 GHz devices (ie. there's a higher probability of achieving any given speed). 2.4 GHz gets cut off a bit below 200 Mbps, but realistically, most devices are more like 75 Mbps or less. 5 GHz trails off starting around 200 Mbps, but a few devices are able to get even 500 Mbps and a bit beyond (cut off on this chart for clarity, but we haven't reached 1.0 yet by the time we get to 500 Mbps, so it must happen sometimes).

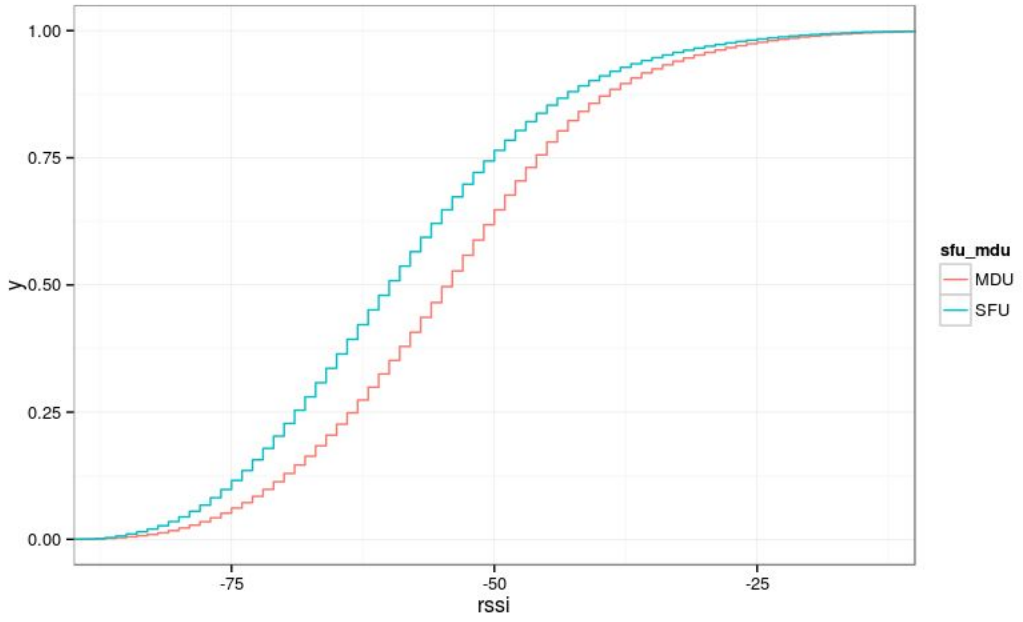
tl;dr use 5 GHz if you want speed!

Real world speed cdf, single-family vs multi-dwelling



Instead of splitting by 2.4 vs 5 GHz, let's split by single homes (SFUs) vs apartments (MDUs). Interestingly, apartments do a bit better on average. I might have expected them to do worse, since apartments tend to be more crowded and thus have more interference. What's going on?

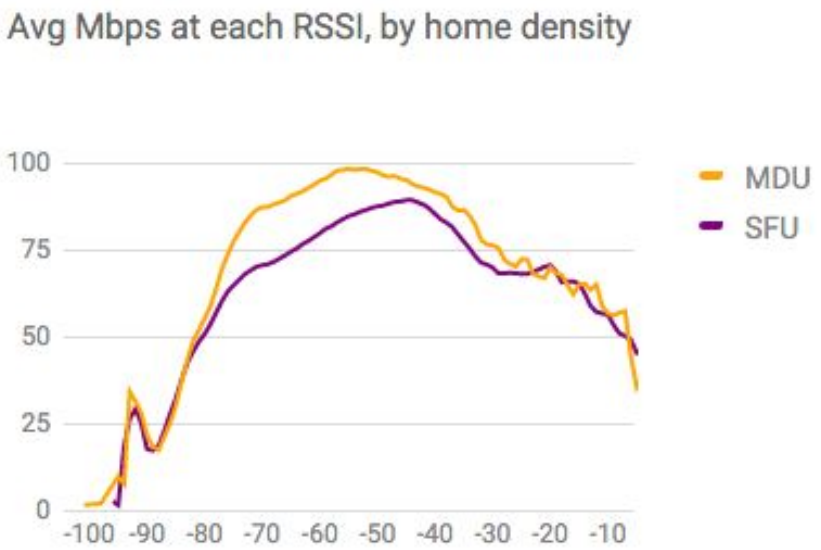
Signal strength cdf, single-family vs multi-dwelling



So let's back off a bit and look at signal strength instead of megabits.

Aha! People in apartments have a higher average signal strength. That makes sense - apartments are smaller, so on average, you'll be closer to your access point.

Speed vs signal, single-family vs multi-dwelling

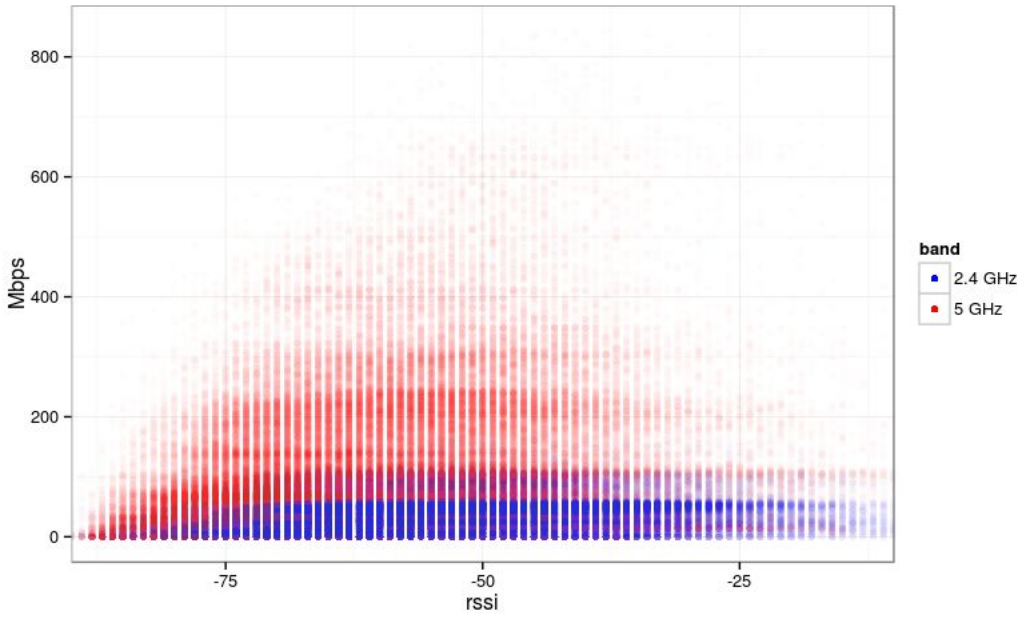


Now let's compare signal strength to achieved speeds.

This is something I can't explain: at any given signal strength, people in apartments seem to get better speeds than people in SFUs. But the numbers don't lie! Probably.

Anyway, it all adds up to being consistent: people in apartments have stronger average signal strength, *and* at any given signal they get faster speeds. There's no conflict there; of course their overall average speeds are also faster. But given that MDUs should have more background interference, I really don't know *why* this plot is the way it is.

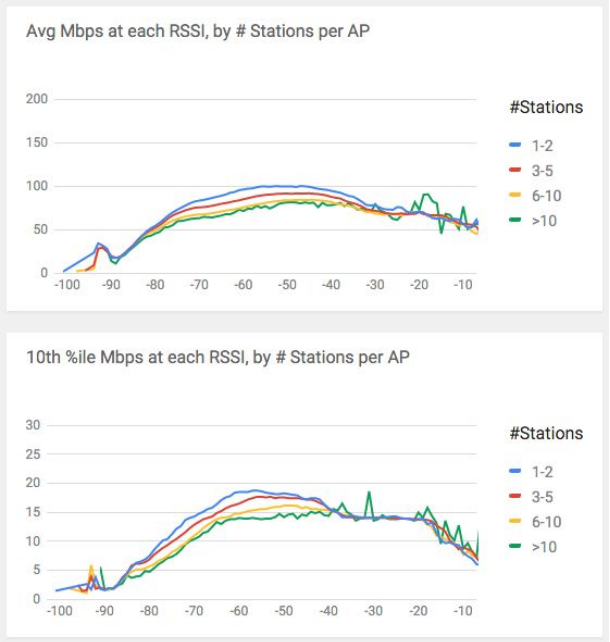
RSSI vs Mbps (individual samples)



This plot is more or less like the last one, except it shows individual samples instead of averages. Like in the museum stairstep plot from earlier, you can definitely see peaks where different types of devices max out.

This visualization is sort of nice because you can visualize simultaneously the number of devices at a given signal strength *and* the achieved throughput. For example, it looks to me like there are fewer 5 GHz devices as you get far toward the right in terms of signal.

Performance drop with more connected stations

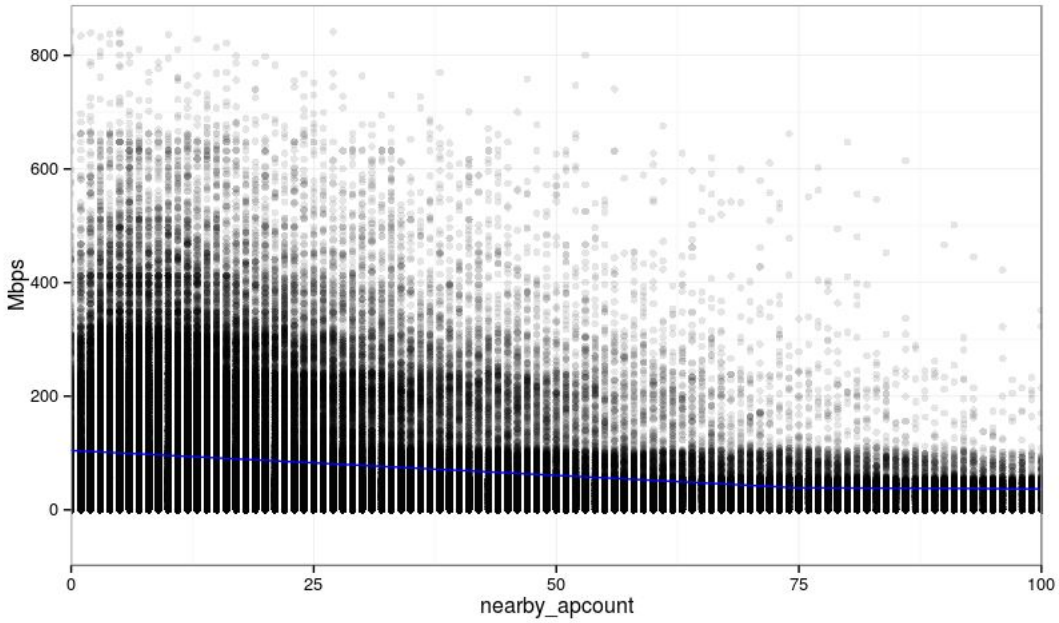


Another quickie: have you ever wondered whether adding more stations to your router makes it slower? Answer: on average, yes, but not by nearly as much as you might have thought.

There are tons of disclaimers here, however. First of all, these are all residential customers. Things are probably very different in an office or at a conference. In a home, most devices (phones, tablets, laptops, TV boxes, fridges) are mostly idle most of the time. At a conference, if it's connected, it's probably in use, so having much more impact.

Still, I use this graph to try to explain to newbies that just having more devices doesn't really make that much impact. It's *active* devices (actually sending/receiving data) that matter.

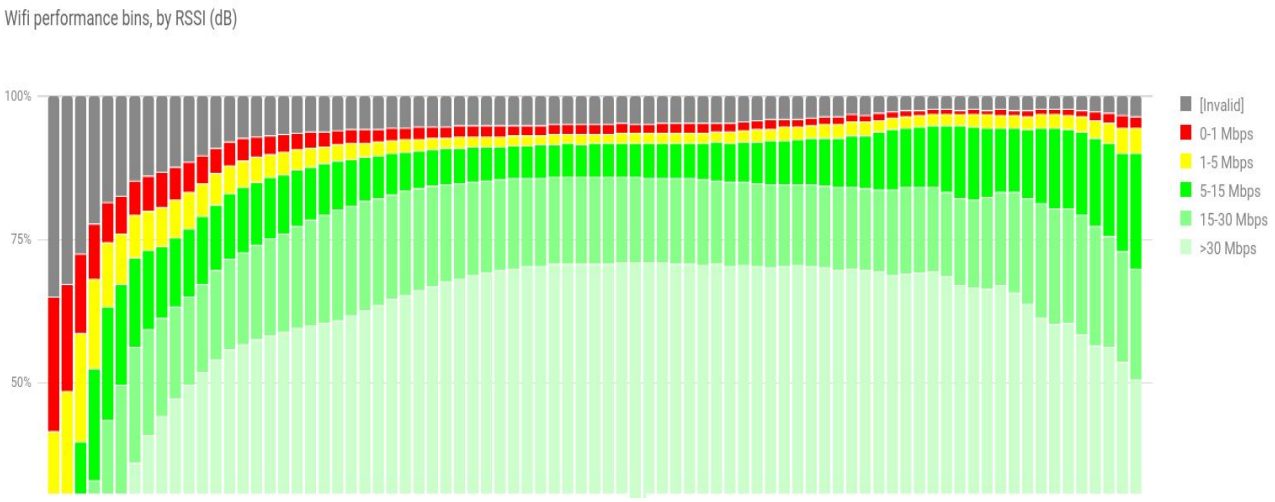
Performance drop with more nearby APs



Similarly, what about the number of nearby access points? Again, there's a performance drop, but it's not nearly as severe as you might imagine. In this plot, we show all the individual samples, where x is the number of visible APs and y is the Mbps achieved. The blue line i the average. It sems like a pretty gentle downward slope if you ask me.

Like the last plot, I use this to try to explain to people that having a zillion nearby APs is not what makes things slow. If all those APs are mostly idle, they don't really matter much. This has huge implications when choosing a good wifi channel to use.

Speed bins



I've been playing with another kind of visualization, "speed bins." The idea is to remove the distraction of talking about those ultra high speeds (500 Mbps or whatever) that don't matter much, and focus on what really matters: how many Internet streaming cat videos can I watch at a time, at any given signal strength? The x axis is RSSI, from lowest to highest, while the y axis is the fraction of samples in a given Mbps range (shown in the legend) The sum of all colours should be 100% at each RSSI level.

For our purposes, let's assume an HD cat video takes about 5 Mbps. If you can stream an HD cat video, that's green. Even more cat videos is lighter green. Yellow would be a low-quality, but still functional cat video, while red would probably be unwatchable. Grey is "we're not really sure but it probably sucks."

Unsurprisingly, as signal strength decreases, your cat video capacity shrinks. More worryingly though, even at high signal strength, we sometimes get wifiblaster results in the yellow, red, or grey regions. We're hoping most of those might be bugs in the wifiblaster test, but probably not all. This is the problem with looking at data: it means you find bugs. Maybe the bugs will turn out to be bugs in wifi drivers, hostapd rekeying, etc.

Wifi Taxonomy

Let’s add one more tool to our tool chest: breaking down all those results by wifi station type, to see if there are per-device-type trends. This is where it gets really interesting!

Wifi taxonomy

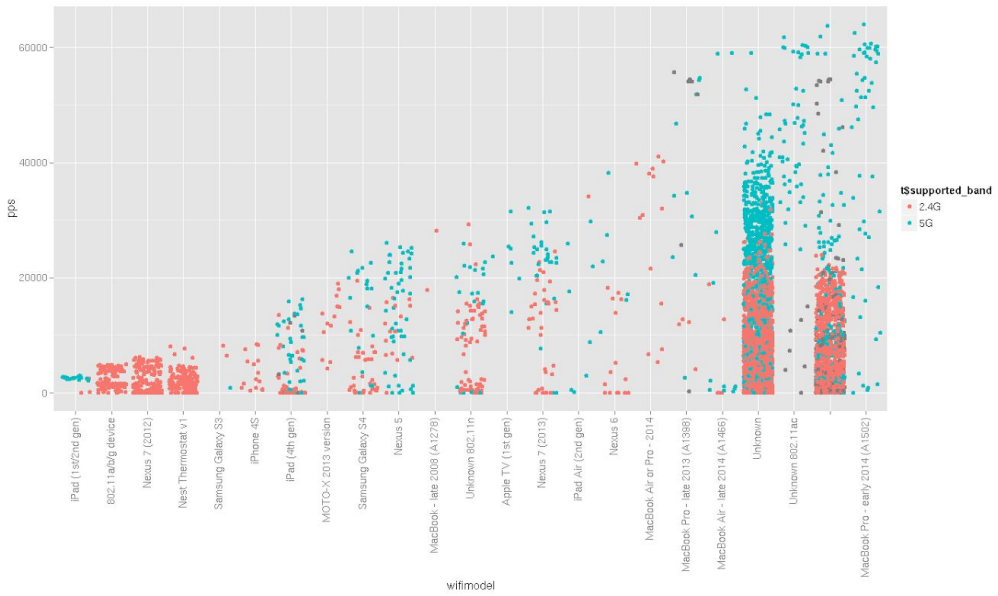
Open source: https://gfiber.googlesource.com/vendor/google/platform/+/master/taxonomy/

```
HT Extended Capabilities (VS): 0x0000
    .... .... .... ...0 = Transmitter supports PCO: Not supported
    .... .... .... .00. = Time needed to transition between 20MHz and 40MHz: No Transition (0x0000)
    .... ..00 .... .... = MCS Feedback capability: STA does not provide MCS feedback (0x0000)
    .... .0.. .... .... = High Throughput: Not supported
    .... 0... .... .... = Reverse Direction Responder: Not supported
Transmit Beam Forming (TxBF) Capabilities (VS): 0x0000
    .... .... .... .... .... .... .... ...0 = Transmit Beamforming: Not supported
    .... .... .... .... .... .... .... ..0. = Receive Staggered Sounding: Not supported
    .... .... .... .... .... .... .... .0.. = Transmit Staggered Sounding: Not supported
    .... .... .... .... .... .... .... 0... = Receive Null Data packet (NDP): Not supported
    .... .... .... .... .... .... ...0 .... = Transmit Null Data packet (NDP): Not supported
    .... .... .... .... .... .... ..0. .... = Implicit TxBF capable: Not supported
    .... .... .... .... .... .... 00.. .... = Calibration: incapable (0x00000000)
    .... .... .... .... .... ...0 .... .... = STA can apply TxBF using CSI explicit feedback: Not supported
    .... .... .... .... .... ..0. .... .... = STA can apply TxBF using uncompressed beamforming feedback matrix
    .... .... .... .... .... .0.. .... .... = STA can apply TxBF using compressed beamforming feedback matrix: 1
    .... .... .... .... ...0 0... .... .... = Receiver can return explicit CSI feedback: not supported (0x00000
    .... .... .... .... .00. .... .... .... = Receiver can return explicit uncompressed Beamforming Feedback Ma
    .... .... .... ...0 0... .... .... .... = STA can compress and use compressed Beamforming Feedback Matrix: 1
    .... .... .... .00. .... .... .... .... = Minimal grouping used for explicit feedback reports: No grouping
    .... .... ...0 0... .... .... .... .... = Max antennae STA can support when CSI feedback required: 1 TX ant
    .... .... .00. .... .... .... .... .... = Max antennae STA can support when uncompressed Beamforming feedba
    .... ...0 0... .... .... .... .... .... = Max antennae STA can support when compressed Beamforming feedback
    .... .00. .... .... .... .... .... .... = Maximum number of rows of CSI explicit feedback: 1 row of CSI (0x
    ...0 0... .... .... .... .... .... .... = Maximum number of space time streams for which channel dimensions
    000. .... .... .... .... .... .... .... = Reserved: 0x00000000
```

We call our feature for identifying device types “wifi taxonomy.” Classically, this sort of classification feature has been called “fingerprinting” (like the OS fingerprinting feature in nmap), but in recent years, the term fingerprinting has started to refer to uniquely identifying users (“browser fingerprinting”), rather than just identifying classes, which is what it used to mean. We are explicitly not trying to identify individual users here, and we go through some contortions to make sure we don’t do it by accident, which you can see if you read our code (linked above). To make this distinction super clear, we call our feature “taxonomy”: a structured breakdown of devices into categories. This is not to be confused with my upcoming wifi taxidermy service, which is different.

Anyway, the basic idea is to look at the wifi management frames and DHCP packets from a given device type, particularly the vendor IEs. This seems to give a remarkably precise way to differentiate between types of devices. Then the only trick is to figure out what type of device each combination of identifiers refers to. Our taxonomy database is at the link above; feel free to borrow it or contribute new signatures to help narrow things down further.

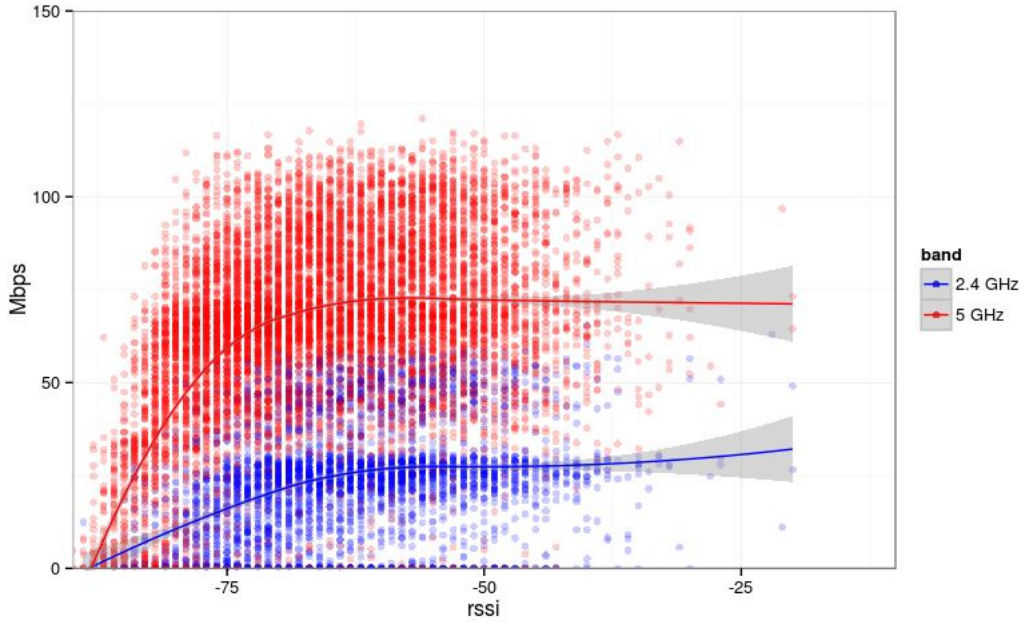
Taxonomy breakdowns



Here's a super simple dotplot of some wifi taxonomy breakdowns we did in the early days. Each bar is a specific device type (the two most-filled-in bars are "unknown" and "null," respectively, oops). The y axis is packets/sec. Red is 2.4 GHz, and blue is 5 GHz; you can see that 5 GHz is faster.

Nowadays we have a lot more known device types, but they don't fit as neatly onto a bar chart.

Real world wifi speeds: particular phone model

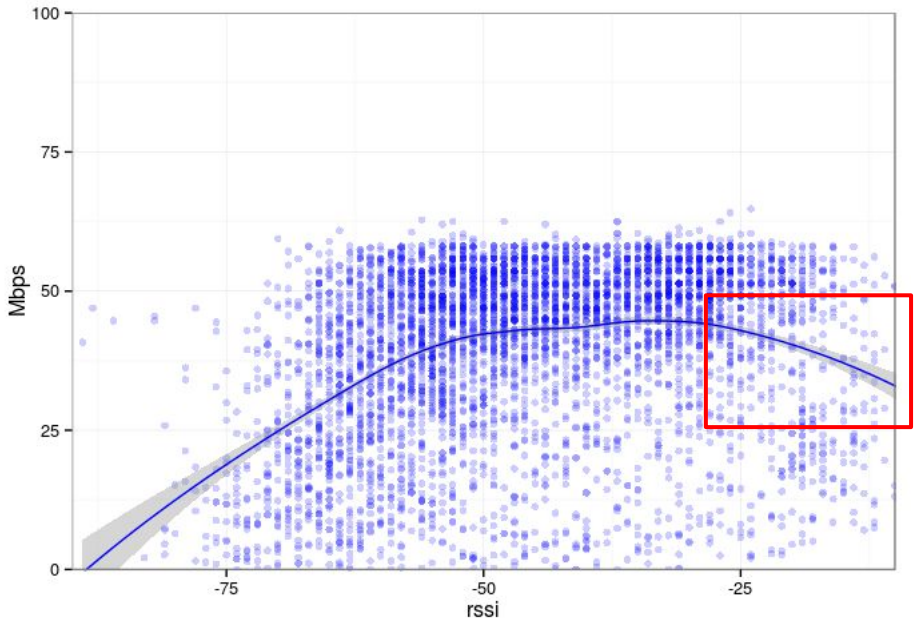


Let's look at a particular type of device. These are the wifiblaster speeds (y axis) for different signal levels (x axis) for one particular popular phone model. A few slides back, we showed this graph across all devices in the fleet; this one filters down to just one phone type. The most obvious difference is that it's easier to see a clear trendline. When you mix a lot of device types together, it's normal for a given signal level to result in many different Mbps levels, depending on CPU speed, antenna positioning, number of antennas, and so on. With a particular device type, these things are a little more constrained.

Abstractly, you can see that at low signal levels, the speeds drop off; as signal gets to -60 dB or so, speeds don't seem to get much better on average, which suggests that they're limited mainly by CPU. (This could be CPU or buffer-size limits on the phone or on the AP; since I know a lot of devices can go faster than that with our AP, I'd guess it's the phone.) You can see two distinct maximum levels at 2.4 GHz, around 30 and 60 Mbps; these correspond to 20 MHz and (much less common on 2.4 GHz) 40 MHz channels, respectively. 5 GHz maxes out at about 120 Mbps for an 80 MHz channel.

The gray region around the trendline (especially toward the right) is the error margin. There are fewer samples over to the right, so the error margin goes up.

Real world wifi speeds: Chromecast



Here's the same picture for a different device: Chromecasts. Again, we can see a distinctive trendline, but this time it's a little different.

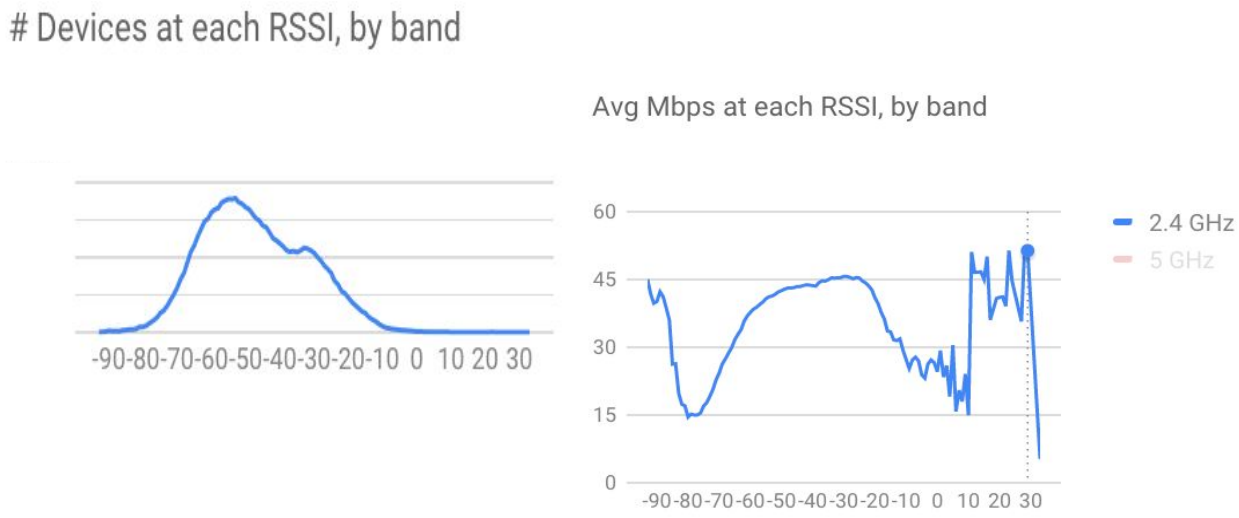
First of all, there are very few samples as signals below -70 dB or so. This suggests that, unlike phones, people tend to put their Chromecasts at locations with decent wifi reception and leave them there. (This is pretty easy to believe since unlike a phone, Chromecasts are useless without wifi reception.)

Secondly, notice a dropoff in the trendline as we move to very *high* signal powers (the red box). It's a bit hard to see from the dotplot, since we sometimes get good speeds at high signal, but the trendline doesn't lie; as the signal goes up, you can see a higher fraction of devices getting poorer transfer rates. What's going on here? Let' s look at it more in the next slide.

By the way, Chromecast 1.0 supported only 2.4 GHz, which is why all these dots are 2.4 GHz.

Too much power: Chromecast and minstrel-blues

Minstrel-blues from Linux Plumbers Conf 2014: http://www.linuxplumbersconf.net/2014/ocw/proposals/2439



Let's drop out the dots and focus on two trendlines: the number of devices at each power level, and the throughput at each power level. Let's also remove the filtering on the x axis, which shows some decidedly unrealistic RSSI levels, all the way up to +30 dB. What's going on there? And why the dropoff in Mbps above -30 dB or so?

Let's back up a moment and look at the left-hand chart, the relative number of devices at each power level. The distribution looks more or less gaussian, which isn't too surprising (people are most likely to be an average distance from their AP), except for a hump near -30 dB. And there are a pretty surprisingly large number of devices above -30, even up to -10 and smoothly declining all the way down to +30. The smoothness of this plot supports the claim that the power level values are probably not blatantly false. (Although you can imagine various types of errors that could lead to mistakes.)

Also, I can tell you that non-Chromecast devices don't show that extra hump at -30 dB. Most devices have just one hump in the middle. Why?

We don't know for sure, of course. All we have is our numbers. But my hypothesis is this: there are two main places you might put your TV or A/V receiver: either in a cabinet or on a shelf right next to your wifi AP, or... not there. If you put it in the same cabinet as your AP, your AP could end up *very* close to your Chromecast. Maybe just a few centimeters away. If you play with putting a wifi client directly on top of (touching?) an AP, you can sometimes get it to report extremely high RSSI levels in the -10 to 0 range. I'm pretty sure Chromecasts are special among device types

because they're so small and easy to pack into a crowded cabinet, sometimes right next to the AP.

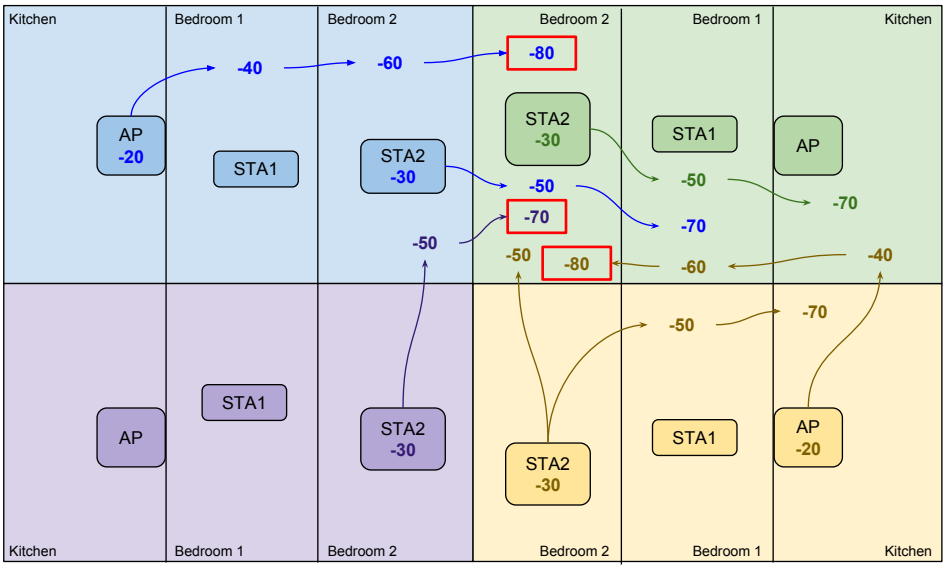
Back to the chart on the right. The very high speeds achieved at signal levels >0 dB look suspiciously like the speeds achieved at much lower levels; that leads me to believe the (very small) number of samples at those high signal powers are just driver bugs, or hardware bugs, or something else lying to us about power level. The speed dropoff from -30 to -10 looks pretty believable though. Here's my hypothesis for that: when your station is too close to the AP, you get what I like to call the "screaming in your ear" effect, which is to say, when someone is right next to your ear, it's a lot easier to hear them if they speak quietly rather than yelling at full blast.

Most wifi APs right now, including ours, just always scream at full blast. The minstrel-blues rate control algorithm, designed by Thomas Hühn, intentionally tries lower power transmissions to see if they work equally or almost as well as higher powered ones. Interestingly, while his algorithm was originally design to sacrifice a bit of personal throughput in order to improve global throughput for everyone (a tricky game theory problem), in this particular case there is no compromise: sending at lower power would make things work better for you, regardless of what anyone else does.

We're planning to integrate minstrel-blues as an a/b test in an upcoming release of our software to see if this chart changes shape.

Cloud autochannel selection? and channel overlap

Open source channel selection: https://gfiber.googlesource.com/vendor/google/platform/+/master/waveguide/



Here's another quick example: a dense customer site (apartment buildings) where pretty much everyone was using our service. For some reason, this site had an unusually high complaint rate. We eventually traced the root cause to unusually thin walls (in terms of wifi propagation, anyway): basically, in every apartment, there was an unusually high level of wifi interference from neighbouring apartments, especially in the back bedroom of each suite. Most interestingly, because of the particular (and consistent) layout of these apartments, the AP was always located in the kitchen area facing the outside of the building, while the back bedrooms had adjoining walls with other apartments. In an unusual twist, better signal propagation was resulting in *worse* wifi range!

In the end, we made things a lot better by doing something unusual: we switched to partially overlapping wifi channels. The traditional wisdom is you should use non-overlapping wifi channels, like 1, 6, and 11. But we had surprising success using channels like 4 and 9. What was going on?

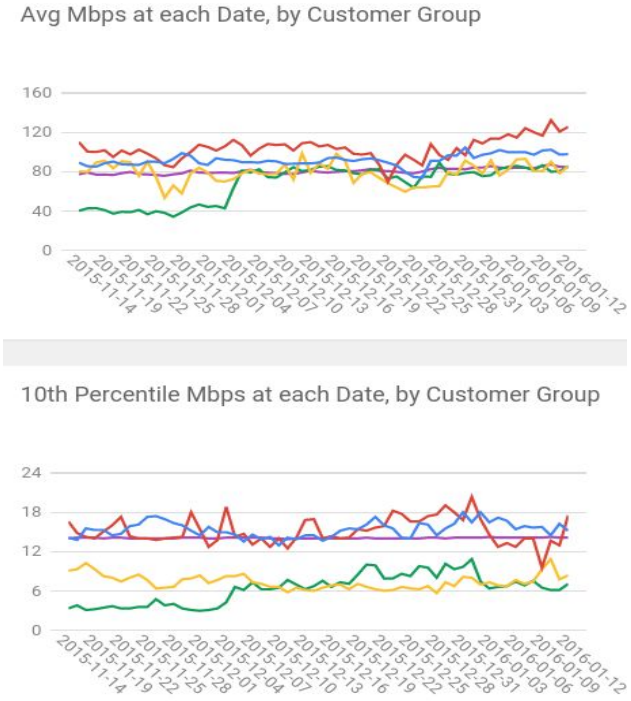
After a lot of investigation, our hypothesis for this comes down to the way channel busy detection works in wifi. Basically, if a packet preamble can be decoded (usually down to -80 or -90 dB), we consider the channel busy. If we can't decode the signal, but there's signal power on the channel anyway, it's typical to consider the channel busy if the signal power is > -62 dB or so. This leads to an odd effect: a channel that would be considered busy if you're on the same channel as the interference (eg. if you're both on channel 6) might *not* be considered busy if you're on an overlapping channel, like 4, and the interference power is between the magic thresholds of -62

and -90 dB! We found that in situations where stations in the back bedroom refused to transmit because the channel was busy almost all the time, we could get them to transmit regardless by using overlapping channels, thus squeezing through enough extra packets to get the cat videos flowing smoothly at last.

So far, the way we dealt with this situation was manually. But we're now thinking of additional options:
- Would using minstrel-blues here have freed up enough air-spacetime (eg. by using less power to talk to devices in the kitchen and bedroom1) to accomplish the same thing?
- Can we improve our autochannel selection algorithm to take this partial-overlap trick into account? When should we do it vs. fully avoiding overlap?
- When we run almost all the APs in a whole building or complex, would a global autochannel optimization algorithm work better?

We have the data collection ability to try a lot of different options here and see what works best.

Before and After



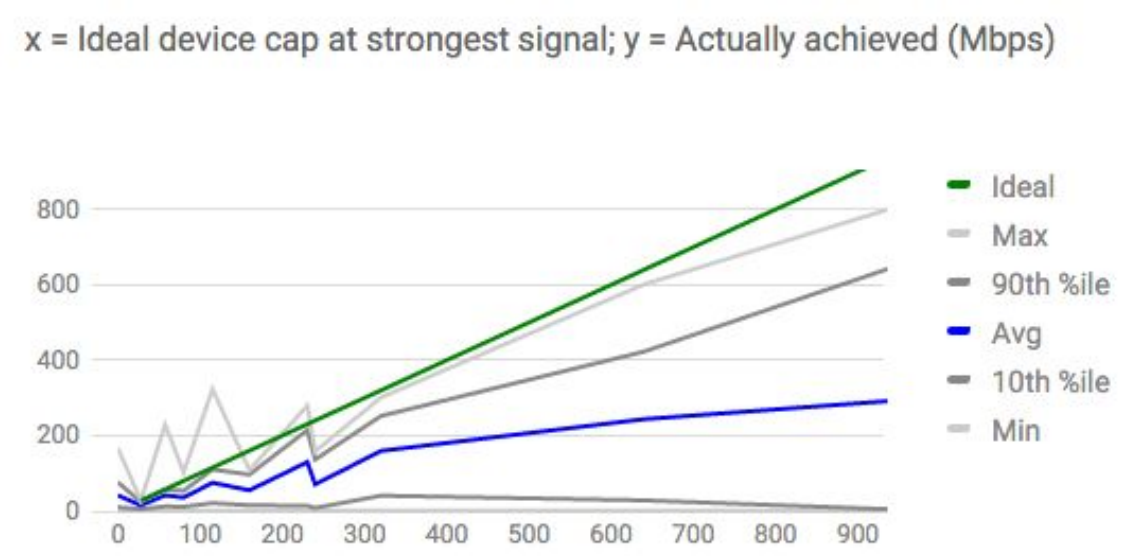
This chart shows a few different customer subgroups and the average (top chart) and 10th percentile (bottom chart) performance over time. Look at the green line: the day we switched to all those partially overlapping channels is the day the average jumped from way-below-average to average. The 10th percentile also got a lot better, though it still trails other (presumably less dense) areas. There may not be anything we can do about that; crowding is crowding.

The good news is this provides evidence that all these numbers aren't just wasting our time. We really can identify performance problems, try things, and see that the charts go up or down, correlated with the real-life experiences our users had while we were on site. That means we have the possibility of doing more automated optimizations without humans having to be physically present.

Speed prediction

Now let’s look at some really “big picture” aggregations, combining wifiblaster and taxonomy, and see what we can do.

Wifi speed prediction (omitting RSSI)



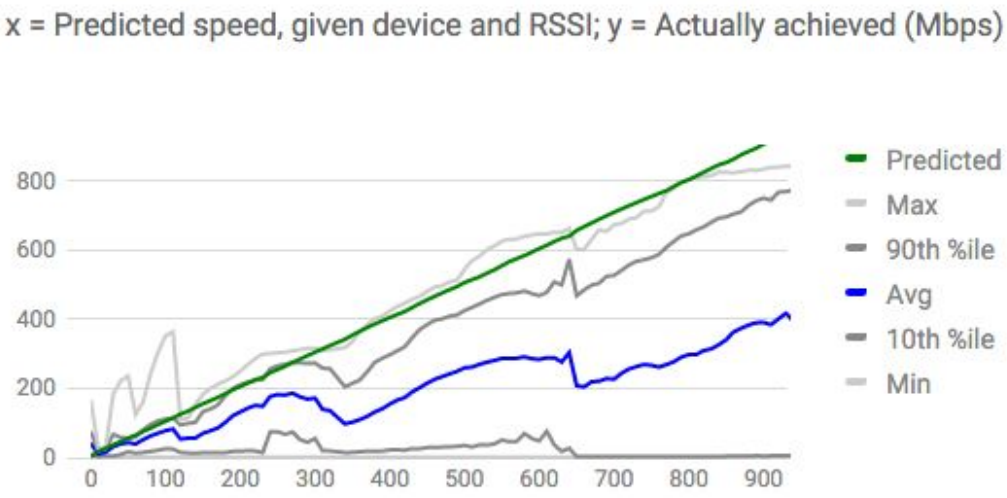
When improving wifi drivers, what we want is to see whether client devices are getting the speeds they “should” be getting, in various situations. And we can try different heuristics to see what gets us closer to ideal. But what is ideal?

Using some of the taxonomy data, we can break devices into their theoretical capability, based on number of antennas, supported wifi standards, etc. We put that on the x axis. Then, for all the samples for a given device capability combination, we calculate the different percentiles of real achieved rates and plot those on the y axis.

The green line is y=x, that is, what you’d expect for devices that achieve their “ideal” speed. It lines up pretty well with the Max line (ie. the max achieved speed at any given capability level). The average speed achieved is less than half of ideal, though, which isn’t great. Even the 90th percentile is pretty far from the ideal line.

...but we’ve left out signal strength, which has an obvious effect on speed. It’s not really fair to say a device at the farthest edge of wifi range should be running at its maximum “capable” speed.

Wifi speed prediction (including RSSI)



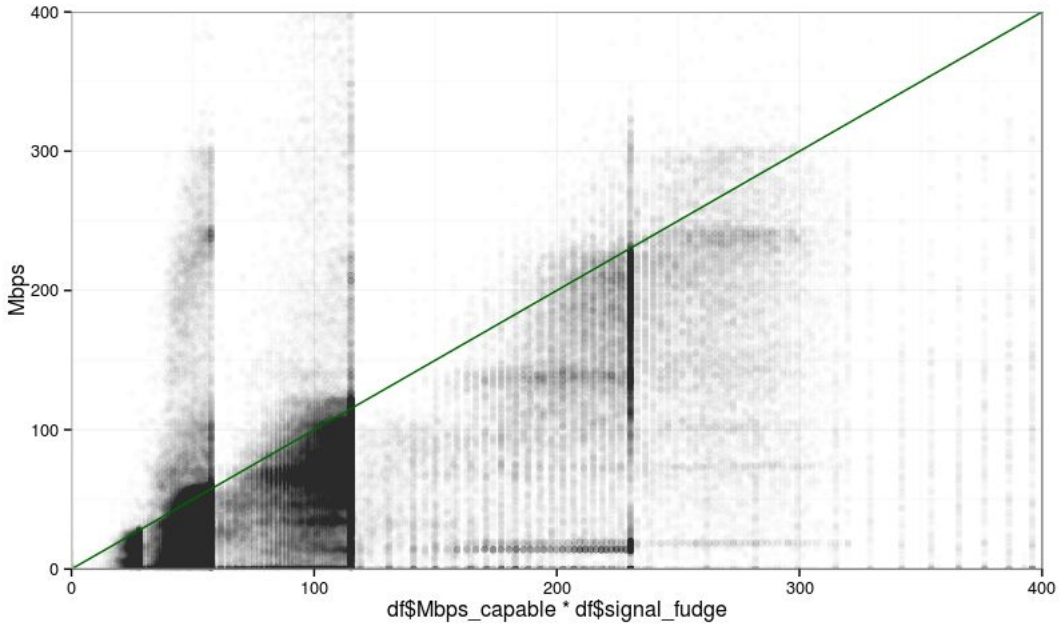
Here's the previous chart, with predictions (x axis) adjusted for RSSI. Now at least the 90th percentile is looking pretty respectable, and even the average has perked up quite a bit; now it's maybe half of predicted (and often more than that, up to 250 Mbps or so), which is quite a bit better than the previous slide.

The 10th percentile and below are looking very bad; this might be bugs in the wifiblaster test, or indicate compatibility problems with certain device types, etc. We aren't sure yet. The point of this graph is it's an optimization goal: our wifi driver teams are working to get the blue line up toward the green line.

GFiber currently doesn't make too many different kinds of devices, but you can imagine that when we launch a new kind of device, we're going to compare them on a chart like this to see which one works better under which conditions.

Wouldn't it be cool if you could test an experimental change to your rate control algorithm and find out what effect it had on this chart?

Wifi speed prediction (including RSSI)

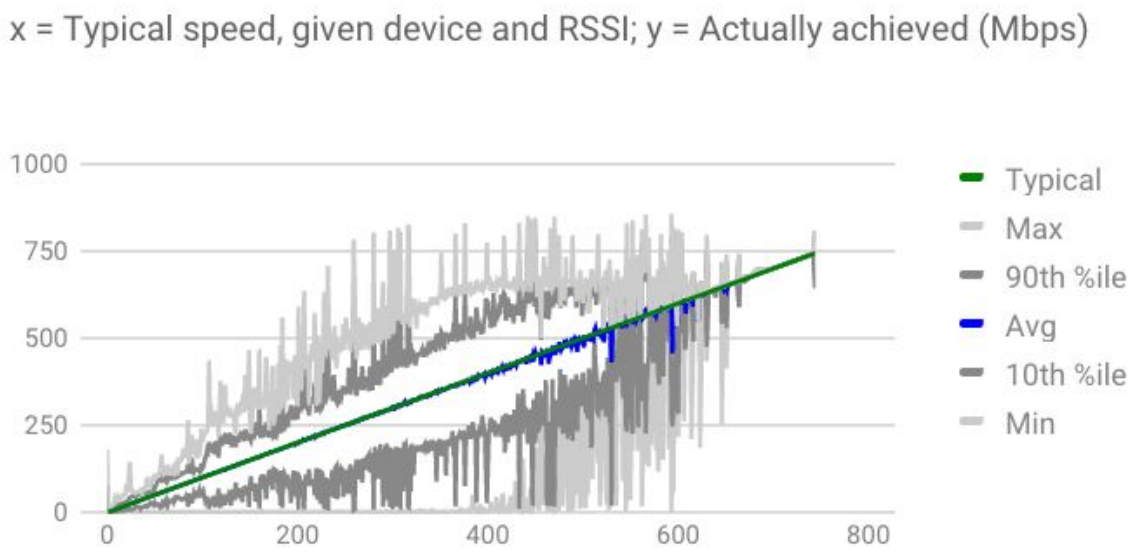


Here's the previous chart again, except using actual dots for the samples, rather than averages. This can help identify outliers or trends that simple averages don't show.

One thing that seems clear here is that some devices we think can only do 60 Mbps or so seem to be getting results way above that, up to 300 Mbps. That's kinda weird. Maybe our taxonomy isn't perfect.

It's also clear that at any given prediction level (x value), speeds tend to be scattered fairly evenly below it. Why? We don't really know. On the other hand, as the speeds get higher (>150 Mbps on the x axis), note how the y value spread seems to decrease; a device that's supposed to be 250 Mbps seems to get a lot of values i the 150-300 range, and not so many in the 0-150 range. That seems better than the slower devices that seem to get a lot of poor results. Why? We don't know yet. Eventually, we'll have to pick out particular device types using taxonomy and see if some work great and others are more suspicious.

Wifi speed prediction based on typical values



Okay, so the previous charts show that today's achieved wifi speeds are not always very close to the ideal, and we don't know why, but we're working on it. (And I guess if they were all operating at the ideal rates, we'd all be done and could go home until the next 802.11 spec came out.)

Another problem we have is trying to help customers figure out if their device is going as fast as it should be. When they phone us and complain about speed, what speed *is* a good one? Should we spend our time trying to debug wifi problems, or should we accept that those speeds are pretty normal for that sort of device, and spend time educating people about wifi device capabilities and which device types go faster?

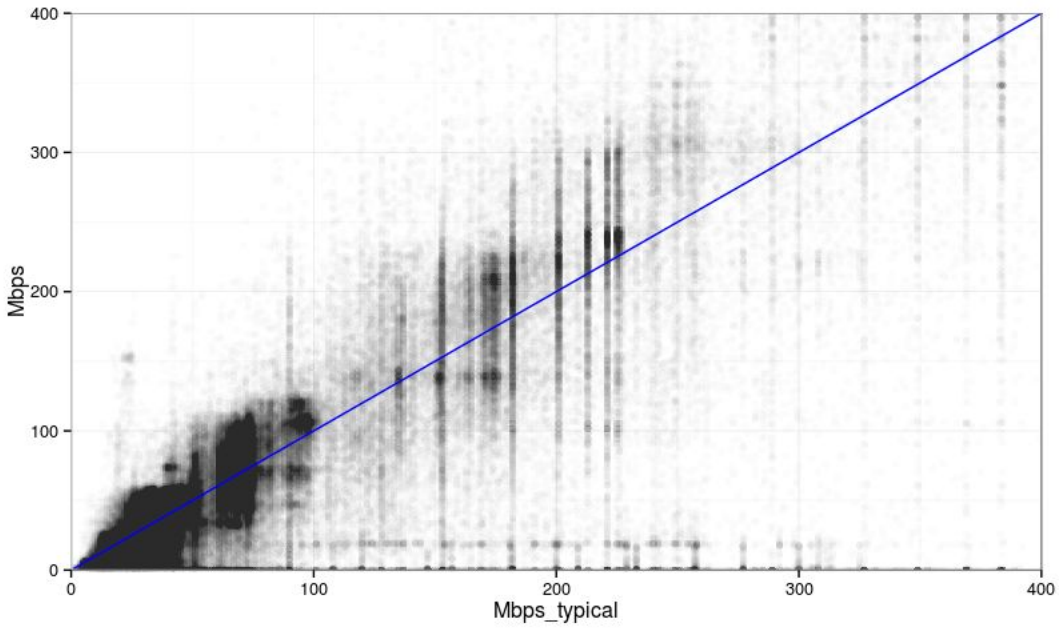
To answer that question, we need to know what speeds are "normal." For that, we take the empirical values from the previous charts, break them out for each taxonomy value, and construct an "expected speed" curve for each taxonomy over the whole RSSI range. Then, for any given taxonomy and RSSI, we have an average "typical" speed, which we plot here on the x axis. The y axis is, as before, the different percentiles of the real achieved speeds.

The green line is this new "typical" calculation, y=x, Not surprisingly, it overlaps almost perfectly with the blue average line, since it was constructed directly from the average. The small discrepancies come from the fact that the x averages are constructed for each taxonomy separately, while the y averages are across all device types.

Now we're getting somewhere! Not only does the average line up nicely, but even the 10th and 90th percentiles are within a 2x factor of the prediction. If someone phones us about a particular device type, and it's above average, we probably shouldn't spend a lot of time fussing with channel changes or moving the AP around inside the home. If it's below typical, that suggests that there's something we can do to get them fixed, so we should spend more time on that.

We don't do it yet, but in theory we could even show pretty graphs to our customers to indicate how their devices are doing relative to the average. We're not yet sure whether that's too complicated for people to understand though, so we have to do more usability research.

Wifi speed prediction based on typical values



Here's the previous graph again, but with the percentiles replaced with the raw data.

Unlike the "prediction" graph we showed previously, this graph vs "typical" speeds is much less disconcerting. At most x levels, the y range is not too wide, most of the time. It's pretty clear that most (though certainly not all) samples are clustered around the blue y=x line. This is a pretty good sign that our averaging algorithm is working.

Making wifi better

Finally, after all that data, let's talk about a few concrete things we can do (or have already started doing) to make people's wifi work better.

Replacing your router almost always helps

P(problem with router A) = 20%

P(problem with router B) = 20%

P(problem with router A **and** B) = 0.2*0.2

= **4%**

I wish I were joking, but this one comes up a lot. If you assume independent sets of bugs from one access point to another - which is not quite true, but not too far off, then your chance that a given bug affects two different randomly selected routers is pretty low.

So a pretty good strategy is:
- Trying installing router A.
- Does it work? Great, stop.
- Does it not work? Okay, replace it with router B.
- Does it work? Great, stop.
- ...and repeat.

If each router has a 20% chance of failing, you're down to only a 4% chance of failing by the time you try the second one. Wow! I think this explains a lot of the bimodal (one star or five star) wifi router reviews. When wifi works, it works pretty well. When it doesn't work, it's super annoying. Which one you get is essentially random chance.

We run into this in our own deployments. We launched 1.0 router hardware and 2.0 router hardware. Naturally, when you number things like that, people assume the 2.0 router hardware is better, so when they have a problem with the 1.0 router, the first thing they want to do is throw it away and get 2.0 hardware. Amazingly, because of the above formula, this almost always works.

What's less obvious, though, is that the opposite would also probably work. But

nobody ever tests that direction. So you end up with the self-fulfilling prophecy that 2.0 routers are always better than 1.0 routers.

(Admittedly, I prefer to believe the 2.0 router is better because that's the one I worked on. And all the statistics say I'm right! I should get a promotion.)

Adding wifi extenders can make things worse

P(problem with router A) = 20%

P(problem with router B) = 20%

P(problem with router A **or** B) = 1 - (1-0.2)*(1-0.2)

= **36%**

Here's the converse situation. As people's expectations for wifi get more and more extreme, I believe we're going to need multiple wifi routers in each home to make sure there's excellent speed and coverage across your whole palace. And you can buy wifi extenders in stores already today.

So why are wifi extenders and repeaters so universally seen as buggy and unreliable? Well, partly because they *are* typically buggy and unreliable, for lots of reasons that I don't have time to go into here. But even if they weren't bad, the statistics would make them seem so!

With a normal router, you might have a disappointingly high bug rate, but if you try a couple of different routers you'll probably be fine, like we talked about in the previous slide. The probability that your bug affects both router A *and* router B is very low.

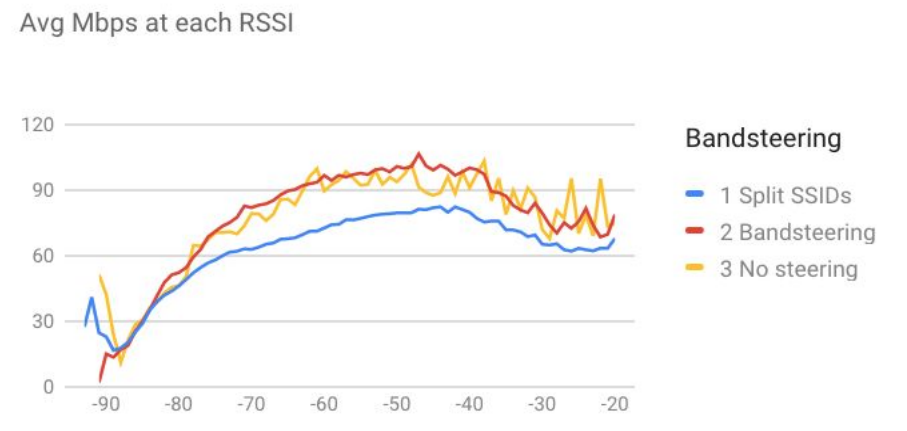
When you have an extender though, your life sucks if ther's a bug in router A *or* router B, because you'll be using one or the other at different times. (And with pure-wireless repeaters, using router B means all your traffic has to bounce through A, so a bug in either one means failure when you try to use router B.) These failures compound as you add extenders.

If you're going to make consumer-grade wifi extenders work reliably, at some point you have to deal with this problem. One way is to get the per-router bug rate down to a very low number; naturally, a lot of the work we're doing, and a lot of the statistics in this presentation, are aimed at that. But another way is to make it so that you can

“route around failures.” If you have 5 access points in a mesh, and 2 of them are buggy, can you disable those 2 and make do with the remaining 3? That’s where the magic lies, if we can make it happen.

Bandsteering (2.4 vs 5 GHz)

Bandsteering patches: https://gfiber.googlesource.com/vendor/opensource/hostap/+/master



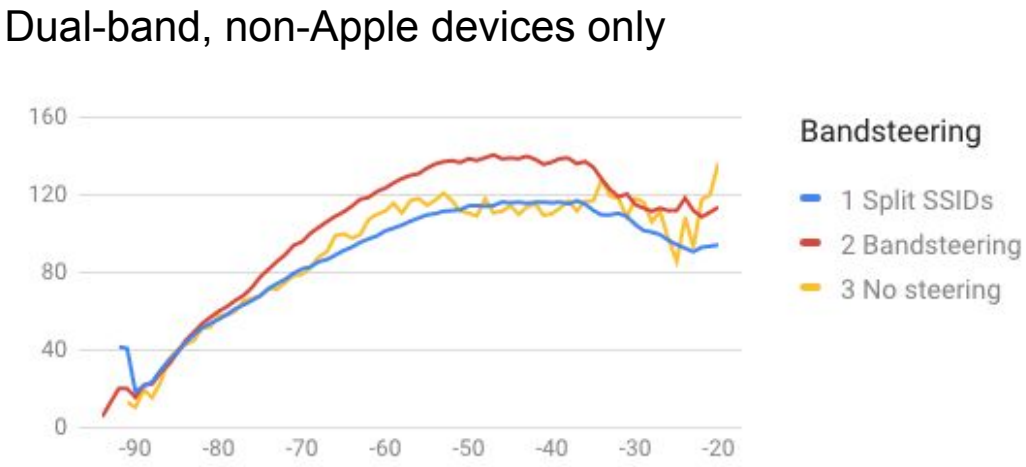
Here's a quick example of a simple improvement we made to hostapd (and haven't upstreamed yet, but it's open source).

One trend we saw over and over in these slides is that 5 GHz is faster than 2.4 GHz; no real surprise there. But we noticed that a lot of 5 GHz capable-devices end up joining our 2.4 GHz network even when a faster 5 GHz network is available. To fix it, we want to "steer" these clients toward 5 GHz.

There are three approaches: 1) Have two different ("split") SSIDs, one for each band, and have a person decide; 2) Have hostapd try to refuse connection on 2.4 GHz so they'll have to use 5 GHz; and 3) Do nothing and trust that the client will be smart enough.

In this chart, you can see the success rates of the three options, broken down by signal power. Amusingly, option 1, to let the humans do it, is pretty much worse across the board. It turns out most humans have no idea whether 2.4 or 5 GHz is better, so they just pick one and stick with it, and it's often the wrong one. It's hard to see here, but #2 really is better than #3, just not by a huge amount. Let's look at a clearer picture next.

Bandsteering (2.4 vs 5 GHz)

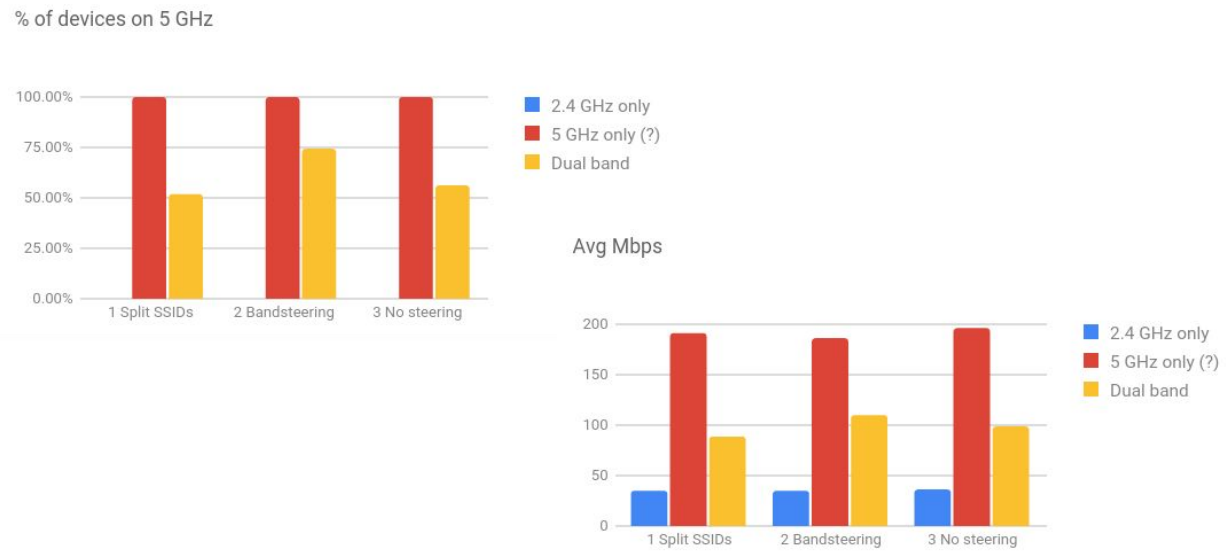


Aha. In this plot, we filtered out all 2.4 GHz-only devices - since they obviously can't be steered - and also Apple devices, because they tend to make pretty good guesses on their own.

Now hostapd-driven bandsteering is a clear win at all signal powers.

Bandsteering (2.4 vs 5 GHz)

Non-Apple devices only



Or let's look at a simpler plot, ignoring signal power since it doesn't seem to have any effect (a mode which is better at high power seems to also be better at low power). We'll leave out Apple devices again because they make pretty good guesses on their own. This lets us focus on the large number of other devices, which need more of our help.

The left chart is the percentage of devices that are on 5 GHz, in the different modes. Clearly, 0% of 2.4-GHz only devices are on 5 GHz, so there's no blue bar. I don't know what a 5 GHz-only device is exactly; it's something that shows up in our taxonomy as only ever being on 5 GHz. I guess maybe that means it supports both bands but is 100% reliable at picking 5 GHz, which is great, I guess; anyway, 100% of those devices are 5 GHz, which isn't too shocking. Dual-band devices are where the differences lie. What's interesting is that even with our bandsteering turned on, we seem to only be getting about 75% of devices pushed to 5 GHz. This could be bugs in our detection (maybe they don't support 5 GHz after all?), or bugs in our steering (maybe we should "push harder"? We err on the side of allowing a device to connect if it seems too insistent on 2.4 GHz), or bugs in the device itself (maybe it just refuses to accept any hints and really really wants 2.4 GHz).

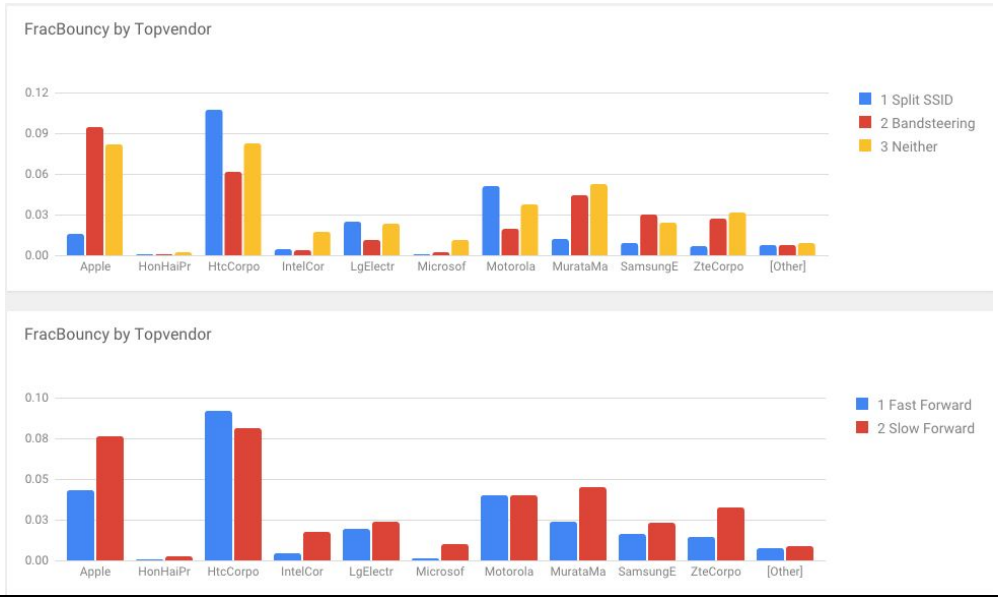
Anyway, that seems to suggest there's a pretty large number of devices that could still be steered, if we figure out how, and that might mean much better average speeds.

I do feel a bit bad about using Average Mbps in this chart; averages don't tell the whole story. In particular, we don't know if bandsteering maybe increases the

average while making the 10th percentile worse for some reason, or if the average slightly increased while the 90th percentile got much higher, or what. So we need to look closer at this.

The good news is that bandsteering really does seem to make things better already, even though I guess it could be even better still.

Bounciness analysis



We're looking at another way of measuring wifi "quality" beyond just speed: a factor I call "bounciness," which is the tendency to jump back and forth between 2.4 and 5 GHz bands. When one or the other wifi radio is acting funny, or there's a bridging bug, or a client device is buggy, it seems to bounce between bands more often.

This breakdown by device OUI shows that some vendors are clearly more "bouncy" than others. I think there are still some confounding factors in here, though: for example, I suspect some device types bounce between bands on purpose, even when things are fine, just to see if the grass really is greener on the other side. If it isn't, they bounce back. We need to do a lot more exploration here to be sure.

Other Experiments

Wifi80211g - force all wifi down to 802.11g

Wifi24G80211g - force 2.4G wifi down to 802.11g

Wifi5GForce20M/40M - force 5 GHz to 20/40 MHz

- May help in very dense areas

Enable/disable **WifiBandsteering**

WifiPrimarySpreading - switch primary 20 MHz channel

NoAutoNarrow - use 40/80 MHz channels even when crowded

WifiRekeyNever - disable peer/group rekeying

NoFastForward - disable hardware forwarding

NoSTP - disable spanning tree protocol on br0

WifiNoBgScans - disable offchannel scanning

We have the ability to try various A/B tests and see the impact they have on performance. Some of the ones I'm most interested in are:

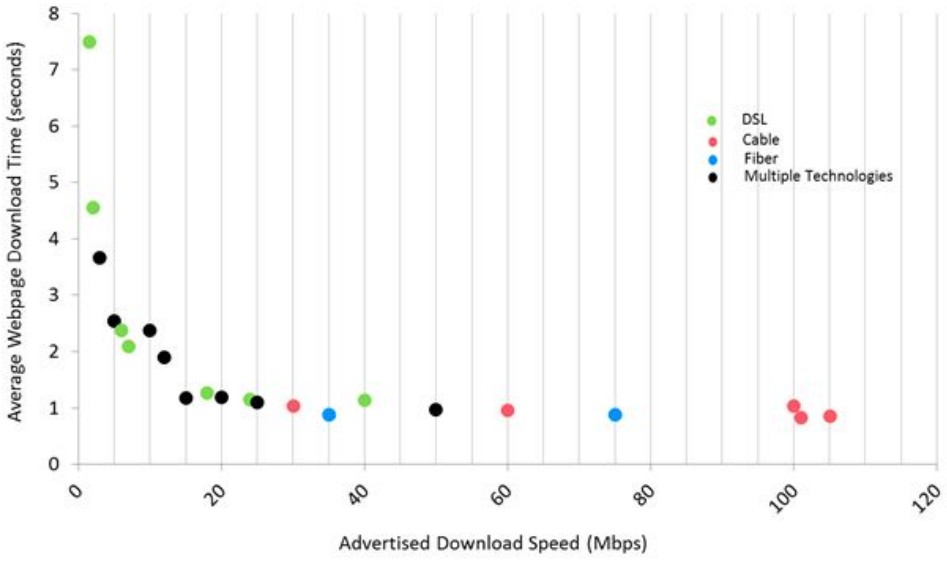
- Experiments with different channel selection algorithms (and how often should we consider re-selecting a channel?)

- Forcing 40 or 80 MHz wifi even when some devices are claiming "40 MHz intolerant" or when there are overlapping channels. I theorize that modern rate selection, like minstrel-ht, should be able to automatically send on a 20 MHz channel when that works better even if we are theoretically using a 40 MHz channel, so that forcing things always down to 20 MHz "just to avoid interference from partially overlapping channels" is not necessary. In fact, it might make things worse by getting *less* effective utilization of the crowded 2.4 GHz band. But we won't know until we try it out and measure the results carefully.

- When the channel in use contains more than one 20 MHz channel (two for 40 MHz, four for 80 MHz), right now hostapd prefers to pick the same "primary" 20 MHz channel as other APs using the same wide channel, because in theory it works better. But does it really work better, especially in crowded environments? I'm not sure it does. We should try it both ways and measure the results.

We need to improve latency

from FCC broadband report



Another thing I want to look at a lot more closely is latency, not just throughput. In wifi, latency is a huge problem because the transmit rate is so wildly variable, which makes algorithms like fq_codel not really good enough to manage queue lengths without some extra work.

This plot is from the FCC. It shows that the time it takes to load a typical web page does get faster with higher throughput, but only up to a point. Beyond about 50 Mbps, it seems to stop improving, and even beyond 15 Mbps or so the improvement is very small. After not very much throughput, the limiting factor becomes latency.

Of course, online gaming, voice, and video chats also really want latency and jitter to be low. Just using simple packet prioritization is too simplistic, just like it is on wired networks. We need something fq_codel-like that will work with wifi. Tim Shepard (who also has a talk at this conference) is working on the problem with us.

Passive latency measurement

- Measure time delays:
  - SYN to SYN-ACK: Internet-facing RTT
  - SYN-ACK to ACK: wifi-facing RTT
- Allows measuring success of fq_codel, per-station queuing, etc.

But wait! How will we know if a given change *really* results in lower end-to-end latency and jitter in the field? We need some kind of measurement we can use, which doesn't require any special software to be installed on client devices (ie. just as the wifiblaster test doesn't). And ideally, we need the test to run often, but only when the device is already awake, and use the same UDP or TCP protocol that data is already using, to discourage anyone from prioritizing eg. ICMP packets above everything else to make latency look super low.

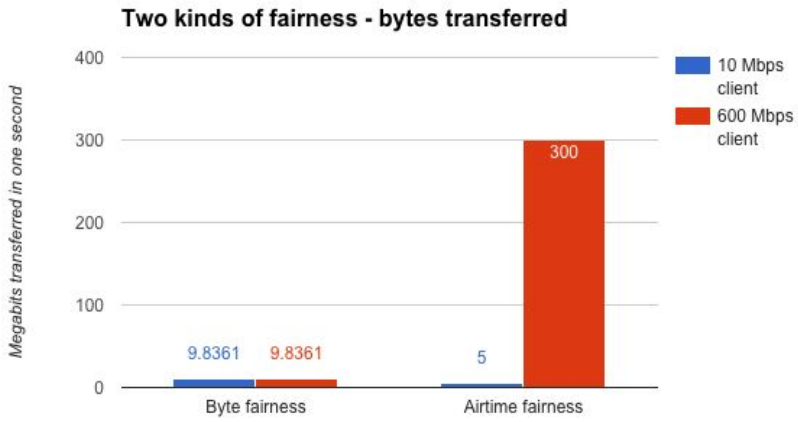
Our idea (not implemented yet) is to monitor the three-way handshake at the beginning of a TCP session. Using the difference between an outgoing SYN and the returned SYN-ACK, we can see Internet-facing latency; using the difference from the SYN-ACK to the ACK, we can see latency back on the LAN-side wifi link.

Of course, on an idle link, which you'd expect your link to be since you haven't finished initiating the TCP session yet, the latency should be very low. The trick here is that we expect there to be various short-lived TCP sessions that get initiated even while background traffic is present; sometimes even when the background traffic is caused by a different station. So laptop A starts downloading a big file (and the beginning of that download showed a very low latency handshake). But then phone B browses a few web pages while the download is in progress, and laptop A's queue affects it. We should see the latency effect on phone B's TCP handshake.

The nice thing about this method is that the initial TCP handshake is generally done before hardware fastpath forwarding is enabled, which means we should see these

three packets getting passed down to Linux, even if the rest of the session just goes through a separate hardware forwarding engine. I think.

Airtime fairness



* Without airtime fairness, slow clients can ruin it for everyone. :(

By the way, another big area of interest for us is airtime fairness: making sure that if one device has a crappy signal and another device has a great signal, we divide up the throughput on a basis more like airtime than total bytes. If we implement airtime fairness in a driver, we should see the average Mbps increase, in situations where there are stations with different RSSI levels present.

Of course, there are lots of tradeoffs here: like with progressive taxation in income tax, the first few megabits can be a lot more valuable than the last few, so it might be better overall to take 10 Mbps from the rich station and give 1 Mbps more to the poor station. Thus perfect airtime fairness might not be desirable either. That could be tricky because if we nudge away from perfect airtime fairness, that could actually *decrease* average Mbps. Thus we wouldn't be able to just use average Mbps as an indicator. Maybe 10th percentile Mbps would increase though? A tradeoff that increases 10 Mbps but makes average Mbps decrease might be a reasonable one.

Yakthulhu

(and Questions?)

The yakthulhu was commissioned by my manager, Denton Gentry. It's a baby cthulhu knitted from yarn made from real yak hair. It symbolizes the yak shaving expedition that our wifi optimization has turned into. One day, I hope to unwind the frightening stack of digressions that this presentation has become, and find that, by carefully paying attention to the endless fiddly quantifiable details, we've finally made wifi great.

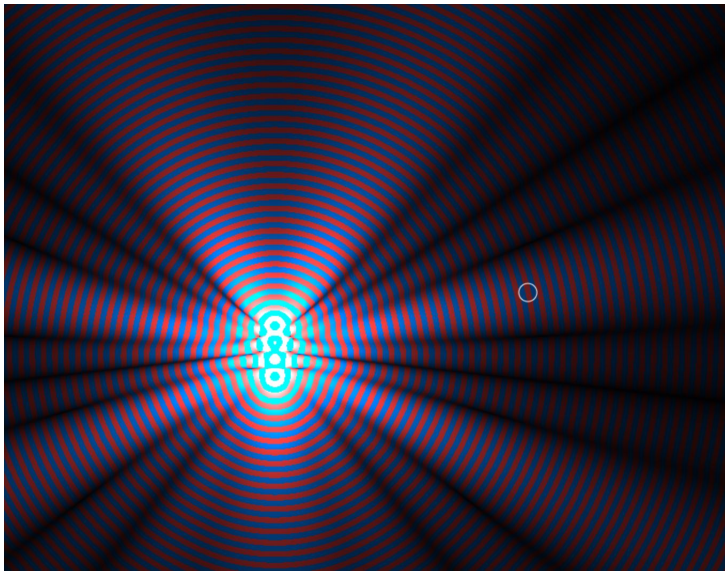
Bonus Slides

Beamforming

Better SNR even for clients with fewer antennas!

↓

Potentially longer range (but not yet)



Anti-beamforming (MU-MIMO)

Only useful with multiple active clients.

(Wireless TV?)