Consumer Grade Wifi & Mesh

@ Battlemesh v10

apenwarr@gmail.com June 2017

READ THE SPEAKER NOTES
(youtube recording)

Opinions & alternative facts herein do not represent my employer's opinions and/or real facts

I've worked on fairly-high-volume ISP wifi deployment and network maintenance, so I know some stuff about wifi and what can go wrong. I've also looked into small-scale mesh-like systems in residential use cases: adding extra wifi access points in a single home, which all route back through the home's wired internet connection. Just doing those super-small scale "mesh" deployments, in large numbers for paying customers who call technical support when there's a problem, taught me a lot about what works and what doesn't.

I'm hoping some of what I learned from this work will be useful to the communities building much larger free mesh deployments, where each network is much more complicated but the user feedback channels are often missing.

What is “consumer grade?”

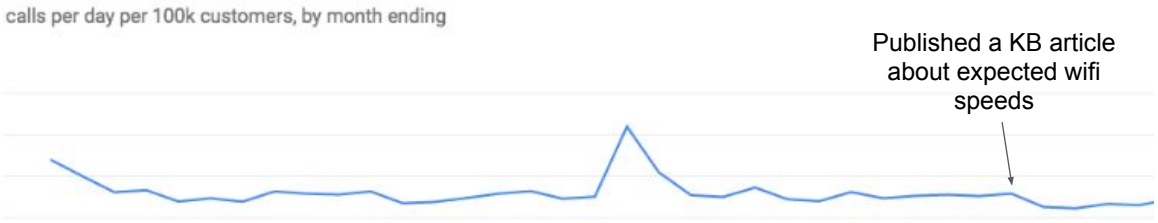
- Something that works even for people who don’t know how to use openwrt.

  (Hardly anyone, right?)

Since I’m talking about “Consumer Grade Wifi” I guess we’d better start by defining what I mean by “consumer grade.” Consumers are non-technical, can’t install their own router firmware(!), don’t even know what openwrt is(!!), and just want the Internet to “work.” This is tricky because, as we all know, the Internet does not in fact work most of the time. So our job is to create a compelling illusion that it works.

Most importantly, typical consumers can’t be assumed to know how to fix things when they go wrong. People have been trained over the years to power cycle their routers in case of a problem, which is the most you can expect (and it means they’ll power cycle the modem at the slightest provocation, whether it’s likely to fix anything or not).

Consumers want boring things



Here's an example of what consumers care about. As an ISP, we get a lot of phone calls from customers every day, with various technical questions and problems. One of the most common is to complain that "the Internet is way slower than I'm paying for." Our particular ISP advertised gigabit speeds, which is completely true (we delivered gigabit speeds through a wired link into the home), but nobody, including us, actually has wifi equipment capable of going at a gigabit, and nobody wants to hook their computer up through a wire anymore.

As a result, we got many phone calls asking for help debugging why, say, their 1x1 802.11n phone couldn't get gigabit speeds.

This is a graph of several years of our wifi-related phone calls. You can see that there was a fairly high rate at the beginning (where we launched something or other), which trailed down to a steady state. Let's not talk about that big spike in the middle; it makes me twitchy. What I want to point out is that we worked on wifi for several years, improving reliability, speed, configurability, and so on, but none of it had much direct impact on the number of support calls. Happy customers might have been happier over time (we did have statistics that they complained less about wifi bugs overall), but they still phoned us at about the same rate...

...except for one major event toward the right, which I pointed out here. That's when we published a knowledgebase (KB) article about wifi speeds and how to tell approximately how fast your device is capable of going. That one article (out of many in the KB) resulted in a sustained reduction of 15% or so in wifi calls.

As software developers, we imagine that the biggest benefits come from complicated algorithms, advanced technologies, and big breakthroughs. But sometimes end users are much better off because you add another page or two of documentation.

Who's the competition?

Competition is "whatever people would do if they don't use your product."

- ~~Other mesh projects~~
- ~~Ethernet/Fiber/Cable/DSL~~
- Cheap LTE

Coming from the world of capitalism and corporations, we spend a lot of time thinking about competitors, and how we're doing compared to them, and how we can improve so we do better than them. I realize most people here work on community mesh projects and don't like the common capitalism-duel-to-the-death model very much. On the other hand, this conference is called "Battle" mesh, after all. As the lore goes, earlier editions of the battlemesh conference had actual competitions between different mesh software projects to see which one was better.

When done in that kind of healthy way, competition helps everybody improve, and so it benefits everyone. That's the kind of competition I'm talking about here, where I want to apply some "capitalist" models to the way we think about competition.

The best definition I ever heard about competition - unfortunately I no longer remember what book I read it in, but it might have been "Crossing the Chasm" or Positioning: The Battle for Your Mind" - is this: your competition is what people are considering using instead of your thing. It's very important to understand this. Many entrepreneurs get confused when they get started: they'll find other companies working on similar products, say, and consider those companies their competition. They usually aren't, because if a customer hasn't heard of that other company, the customer is not considering them. If you get to them first, the customer will not have an existential debate about whether to use your product or other company's product. They will, most of the time, just debate whether to keep doing what they're doing, or switch to your thing. Your toughest competitor is usually the status quo.

So I propose that we redefine the “battle of the mesh”; if you want a real battle, it shouldn’t be a battle between mesh projects or mesh algorithms or mesh communities. It should be a battle between the real competitors: mesh vs. whatever else customers might do to get online.

The next obvious competitor is wired Internet links. Are customers choosing between mesh and, say,, paying for a cable modem connection? Well, maybe, in some places. But the world is changing fast; people don’t like wires. If anything, your competition is wifi connected via a cable modem, not the cable connection itself. And that’s an easier competitor, because their wifi has most (but not quite all) of the same problems as mesh wifi.

But, you might say, our community mesh is mainly to help people in areas where wired infrastructure like that is unavailable, or only sparsely available. Good point! In that case, wired links are not your competition either.

The real competitor is LTE (cellular data). It’s available almost everywhere, and getting cheaper all the time. Some new market entrants (for example in India) are selling very cheap plans targeted at people earning only a few dollars a day, and getting really popular. Virtually everyone in the world might be *considering* buying an LTE phone and plan. That’s the primary competitor for your wifi mesh.

LTE is easier than wifi, and getting cheaper

	LTE	**Wifi Mesh**
Coverage area	High signal power	Many cheap nodes
Authentication	SIMs, auto roaming	Captive portals
Mobility	Keep your IP address	IP address might change
Speed	Predictable (reserved timeslots)	Variable throughput and latency
Technology	Expensive base stations -> fancy hardware	Cheap -> last-gen hardware
Management	Central, global, professional	Hope
Service	24 hour repair trucks	Your 12 year old
Phone calls	Actually work	Laggy (but with video)

LTE has a lot of advantages over wifi, listed here.

The big and unfortunate disadvantage is that it's highly centralized, both in terms of technical systems (providers often route all traffic back to a central point in a city or region before sending it back out again, increasing latency if you want to talk to your neighbour), and politics (companies "own" a chunk of public spectrum that they then rent back to you). This kind of centralization, and more things like it, are shaping the evolution of the Internet. It used to be a peer-to-peer network where anyone could provide a service without anything telling them what to do. More and more, you have to ask permission from one of a few giant, mostly unregulated, Internet providers and service providers before you can do what you want.

I know most of the people here, working on community wifi mesh networks, know all this, and respond with a goal of greater openness and less centralization. That's the sales pitch (and market positioning) that works for you. But it's way too complicated for most people. What most people care about, mostly, on a day to day basis, are things like what's in my table here. Does it work? Does it work everywhere? Does it connect automatically without hassling me? Is the speed decent? If something goes wrong, will someone fix it for me?

Next, let's break down some of those in more detail.

Avery's laws of wifi reliability

Replacing your router:

- Vendor A: 10% broken
- Vendor B: 10% broken
- P(both A and B broken):
  10% x 10% = 1%

Replacing your router (or firmware) almost always fixes your problem.

Adding a wifi extender:

- Router A: 90% working
- Router B: 90% working
- P(both A and B working):
  90% x 90% = 81%

Adding an additional router almost always makes things worse.

First of all, reliability. All wireless networks, both LTE and mesh, go down sometimes, but I'm willing to bet that your wifi network is flakier than your phone's LTE connection. And of course, at Battlemesh we're all sitting in a room with dozens of experimental misconfigured wifi routers offering open networks that may or may not ever successfully route back to the real Internet. What makes a network reliable or unreliable?

After a few years of messing with this stuff (and being surrounded by tons of engineers working on other distributed systems problems, which turn out to all have similar constraints), I think I can summarize it like this. Distributed systems are more reliable when you can get a service from one node OR another. They get less reliable when a service depends on one node AND another. And the numbers combine multiplicatively, so the more nodes you have, the faster it drops off.

For a non-wireless example, imagine running a web server with a database. If those are on two computers (real or virtual), then your web app goes down if you don't have the web server AND the database server working perfectly. It's inherently less reliable than a system that requires a web server, but does not require a database. Conversely, imagine you arrange for failover between two database servers, so that if one goes down, we switch to the other one. The database is up if the primary server OR the secondary server is working, and that's a lot better. But it's still less reliable than if you didn't need a database server at all.

Let's take that back to wifi. Imagine I have a wifi router from vendor A. Wifi routers

usually suck, so for the sake of illustration, let's say it's 90% reliable, and for simplicity, let's define that as "it works great for 90% of customers and has annoying bugs for 10%." 90% of customers who buy a vendor A router will be happy, and then never change it again. 10% will be unhappy, so they buy a new router - one from vendor B. That one also works for 90% of people, but if the bugs are independent, it'll work for a *different* 90%. What that means is, 90% of the people are now using vendor A, and happy; 90% of 10% are now using vendor B, and happy. That's a 99% happiness rate! Even though both routers are only 90% reliable. It works because everyone has the choice between router A OR router B, so they pick the one that works and throw away the other.

This applies equally well to software (vendor firmware vs openwrt vs tomato) or software versions (people might not upgrade from v1.0 to v2.0 unless v1.0 gave them trouble). In our project, we had a v1 router and a v2 router. v1 worked fine for most people, but not all. When v2 came out, we started giving out v2 routers to all new customers, but also to v1 customers who complained that their v1 router had problems. When we drew a graph of customer satisfaction, it went up right after the v2 release. Sweet! (Especially sweet since the v2 router was my team's project :)). Upgrade them all, right?

Well, no, not necessarily. The problem was we were biasing our statistics: we only upgraded v1 users with problems to v2. We didn't "upgrade" v2 users with problems (of course there were some) to v1. Maybe both routers were only 90% reliable; the story above would have worked just as well in reverse. The same phenomenon explains why some people switch from openwrt to tomato and rave about how much more reliable it is, and vice versa, or Red Hat vs Debian, or Linux vs FreeBSD, etc. This is the "It works for me!" phenomenon in open source; simple probability. You only have an incentive to switch if the thing you have is giving *you* a problem, right now.

But the flip side of the equation is also true, and that matters a lot for mesh. When you set up multiple routers in a chain, now you depend on router A AND router B to both work properly, or your network is flakey. Wifi is notorious for this: one router accepts connections, but acts weird (eg. doesn't route packets), and clients still latch onto that router, and it ruins it for everyone. As the number of mesh nodes increases, the probability of this happening increases fast.

LTE base stations also have reliability problems, of course - plenty of them. But they usually aren't arranged in a mesh, and a single LTE station usually covers a much larger area, so there are fewer nodes to depend on. Also, each LTE node is typically "too big to fail" - in other words, it will annoy so many people, so quickly, that the phone company will need to fix it fast. A single mesh node being flakey might affect only a smaller region of space, so that everyone passing through that area would be affected, but most of the time, they aren't. That leads to a vague impression of "wifi meshes are flakey and LTE is reliable", even if your own mesh link is working most of

the time. It’s all a game of statistics.

Solution: The Buddy System

Let your friend tell you if you're making an ass of yourself.

- Router A: 90% working
- Router B: 90% working
- P(either A or B working):
  (1-0.9) x (1-0.9) = 99%

Auto-shutdown when flakey is essential.

In the last 15 years or so, distributed systems theory and practice have come a *long* way. We now, mostly, know how to convert an AND situation into an OR situation. If you have a RAID5 array, and one of the disks dies, you take that disk out of circulation so you can replace it before the next one dies. If you have a 200-node nosql database service, you make sure nodes that fail stop getting queries routed to them so that the others can pick up the slack. If one of your web servers gets overloaded running Ruby on Rails bloatware, your load balancers redirect traffic to one of the nodes that's less loaded, until the first server catches up.

So it should be with wifi: if your wifi router is acting weird, it needs to be taken out of circulation until it's fixed.

Unfortunately, it's harder to measure wifi router performance than database or web server performance. A database server can easily test itself; just run a couple of queries and make sure its request socket is up. Since all your web servers are accessible from the Internet, you can have a single "prober" service query them all one by one to make sure they're working, and reboot the ones that stop. But by definition, not all your wifi mesh nodes are accessible via direct wifi link from one place, so a single prober isn't going to work.

Here's my proposal, which I call the "wifi buddy system." The analogy is if you and some friends go to a bar, and you get too drunk, and start acting like a jerk. Because you're too drunk, you don't necessarily know you're acting like a jerk. It can be hard to tell. But you know who can tell? Your friends. Usually even if they're also drunk.

Although by definition, not all your mesh nodes are reachable from one place, you can also say that by definition, every mesh node is reachable by at least one other mesh node. Otherwise it wouldn't be a mesh, and you'd have bigger problems. That gives us a clue for how to fix it. Each mesh node should occasionally try to connect up to one or more nearby nodes, pretending to be an end user, and see if it can route traffic or not. If it can, then great! Tell that node it's doing a great job, keep it up. If not, then bad! Tell that node it had better get back on the wagon. (Strictly speaking, the safest way to implement this is to send only "you're doing great" messages after polling. A node that is broken might not be capable of receiving "you're doing badly" messages. You want a watchdog-like system that resets the node when it doesn't get a "great!" message within a given time limit.)

In a sufficiently dense mesh - where there's always two or more routes between a given pair of nodes - this converts AND behaviour to OR behaviour. Now, adding nodes (ones that can decommission themselves when there's a problem) makes things *more* reliable instead of *less*.

That gives meshes an advantage over LTE instead of a disadvantage: LTE has less redundancy. If a base station goes down, a whole area loses coverage and the phone company needs to rush to fix it. If a mesh node goes down, we route around the problem and fix it at our leisure later.

A little bit of math goes a long way!

Auto-configuration (forget about LANs)

Run a hidden SSID that only allows a restricted set of servers (sniproxy)
...at a low speed.

Let any device connect to that SSID and download its configuration.

If configured correctly, the device will connect to a “real” network.

--> Easier cloud configuration workflow

Now that we’ve totally solved the reliability problem (ha ha), let’s change gears for a bit. How do we connect and authorize a new device?

With LTE, it works like this: you buy a device, insert a SIM card, and it just works. Behind the scenes, it uses the SIM card to decide what networks it’s willing to talk to, then gets its network configuration from the base station, then sets up a network circuit.

When I buy a new wifi device from the store, say a Chromecast or an Echo or a wifi extender, that’s not what happens. Even if there’s an open wifi network right there, the device sits around and does nothing until you configure it. Usually the way they let you do this is to have the device advertise a separate wifi network that you have to join (losing your connection to the Internet), then go to a magic IP address (192.168.1.1), then set up the “real” wifi, and then you change your terminal back to the main network. This isn’t so bad if you know what you’re doing, but it’s tedious for us techies, and highly confusing for everyone else.

What would be better is if every device could connect up to the Internet right after it’s powered on, and download its configuration. Meanwhile, your laptop or phone, also on the Internet, could post configuration settings somewhere on the Internet for your new device to pick it up.

The concept of a LAN - a local network with a local subnet where all your personal devices are directly reachable by each other, but not by anyone else - is getting

increasingly obsolete. First of all, any phone on LTE can't talk to a "local" device; but also, we all move around a lot more than we used to, and we expect to be able to access or files, web pages, printers, and other services regardless of what subnet we're on. One of the few things still relying on the LAN concept is device configuration, and it needs to go away.

My proposal here is to define a quasi-standard for *all* wifi devices, where there is a well-known (hidden, open) SSID that's always available for configuration purposes. Access points always publish this SSID, and client devices always try connecting to it (or maybe they try any open SSID). The hidden SSID could restrict access to only https on a restricted set of services (to be agreed upon somehow by the community), and limit throughput, so that nobody providing this SSID would need to worry about abuse. (Someday, I hope everyone can just make all our wifi routers wide open and not worry about abuse. But today is probably not that day, so this is the tradeoff.)

I think this idea could spread pretty fast if we got it into, say, openwrt and convinced a few hardware vendors to enable it by default.

Streaming video & rebuffers (isostream)

That covered network reliability and network setup. Now let's talk about what people actually want to do with their Internet connection, once it's working, and what prevents them from doing what they want.

Most data sent on the Internet today is video, especially streaming video like Netflix or Youtube, so the ability to deliver uninterrupted streaming video streams is key to user satisfaction. Other than raw throughput, the key measurements in a video stream are **time spent prebuffering** at the start, and **frequency of rebuffering** during playback. Prebuffering means you have to wait longer before you can start watching. Rebuffering means the video stops playing and you have to recover before continuing. Both are annoying.

The two factors are very related. If it weren't for rebuffering problems, you wouldn't need prebuffering in the first place. If you have infinite prebuffering (like if you download the entire video in advance), you will never need to rebuffer.

What these factors actually reflect is that available throughput is actually variable, not constant. Sometimes you might get 5 Mbps, but sometimes you might get 1 Mbps, and sometimes it will even drop to zero. Video playback software needs to anticipate variable throughput and compensate (either by buffering enough in advance, or being able to downshift to lower-rate video in a hurry, or probably both).

We're not the people who will fix the video playback software. But as network providers, what we can do is try to build a network that can deliver **consistent and**

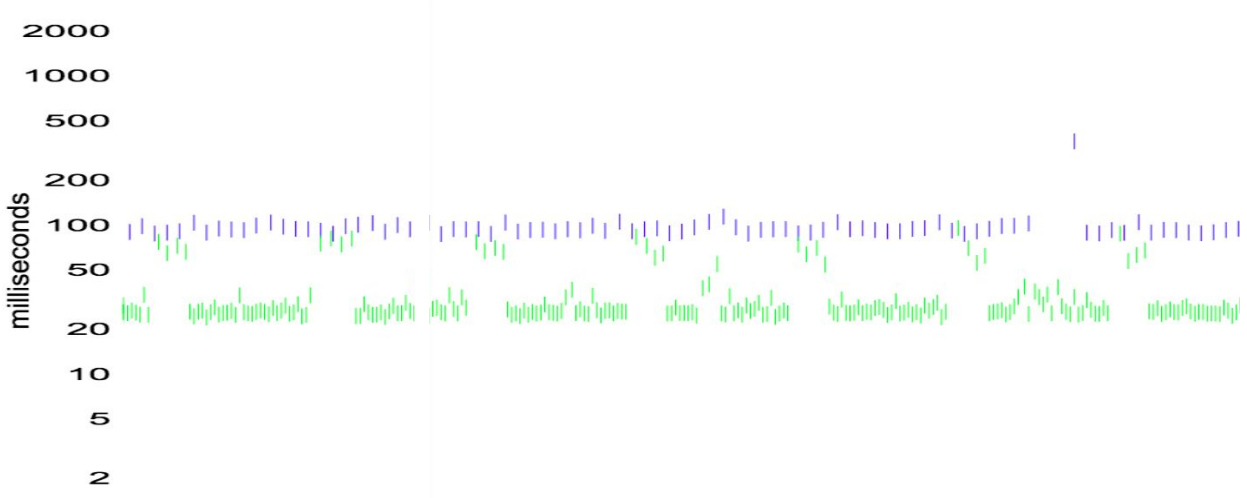
slow-changing throughput. For video, it's actually massively better to have a constant 3 Mbps available than to have a link that goes 5 Mbps some of the time, but 1 Mbps for a few seconds every now and then. That's because with a constant bitrate available, even dumb video playback software can figure out what to do. With the bitrate fluctuating wildly, you depend on the video playback people to try to estimate what's available. And let's face it, if *you* don't know how to estimate your own mesh throughput reliably (and I bet you don't), then the video people are probably going to do even worse.

This slide happens to contain a plug for my isostream program (link above), which sends a constant-rate TCP stream "as if" it were being sent by an especially dumb single-rate video streaming system. It then tracks dropouts (periods in which we fell behind real time), their depth (how far behind we fell, a negative number of seconds) and their width (how long we stayed behind before catching up, a positive number of seconds). Networks with occasional performance drops will end up looking like this chart, which shows several isostream sessions superimposed. Purple has the worst dropouts, but light green and bluey-green also had a couple. Based on isostream data, you could predict the amount of prebuffering needed by your link in order to avoid rebuffers during these dropouts. (For example, purple needed at least ~0.7 seconds of prebuffering in this example, if we make the simplistic assumption that the worst observed dropout is as bad as the worst likely dropout.)

Sporadic mesh routing changes are a huge problem because they can change the available throughput suddenly, and (depending on the routing algorithm) sometimes frequently, which is the worst case for streaming video.

By the way, queue systems like fq_codel intentionally detect "thin" streams (like fixed-rate video streams that don't try to use all the available bandwidth) vs "thick" streams (like http downloads of big files as fast as possible), and try to prioritize thin over thick. The result is that sporadic http traffic has less impact on the available throughput for long-term video downloads, causing things to change more gently and thus causing fewer rebuffers. If you can turn on fq_codel on all your links, definitely do it! (But unless you're also using per-station wifi queues - coming up in another slide - don't set the target latency too low. Wifi is weird about latency. For video streaming what you need is fq_codel's auto prioritization feature, more than its minimum latency feature.)

Live video & jitter (gfblip.appspot.com, isoping)



That was streaming video, which is responsible for most Internet data transfer today, and probably always will be (most people are mostly passive consumers, most of the time, which is good because otherwise nobody would have time to read anybody else's content).

A related, yet surprisingly very different, problem is real-time audio and video. Streaming Netflix and Youtube isn't really "real time", because you can add prebuffering to fix most of your problems, at the cost of a bit longer startup time. But if you're trying to have a phone call or video chat, you can't just prebuffer 5 seconds of video, or it'll be unusable. You really want 200-300ms at the most.

300ms should be pretty achievable, right? You can literally send a signal over the Internet to anywhere in the world in less than 300ms. Problem solved, let's all go home!

Well, hold on. There are two problems that ruin it for everybody: bufferbloat and jitter. You probably already know about bufferbloat. If you don't, look it up! It's important. The short version of bufferbloat is that if you have a huge queue, a bottleneck (like a slow wifi link), and a typical network stack, the queue will spend most of its time full, and your minimum latency will be defined mostly by the amount of memory you give the queue, rather than the actual time it takes to get a packet from one place to another. The more memory you have, or the slower the link, the more latency rises. Nowadays there are a few mitigations for this, but they aren't enabled by default on most platforms! My favourite ones are fq_codel (on routers) and TCP BBR (on

endpoints). Use them.

A related problem is jitter. Jitter is the variation in latency, which for a live stream, can be even worse than consistently high latency. (Just like in the previous slide, variable throughput was worse than consistently lower throughput.)

This slide shows one of my two latency measurement tools, gfblip, which is a javascript program that should run on any device with a web browser (including phones, tablets, laptops, etc). It connects to two servers, green (something "nearby") and blue (something "further away") and repeatedly http-pings the server and plots the round trip time. Note that the y axis is a logarithmic scale rather than linear, so green is lower latency than blue, but the big jumps in green would correspond to much smaller jumps in blue. Anyway, reading from the blip chart, you can see that green is typically about 30ms, but sometimes jumps to around 90ms. Blue is typically about 100ms, but sometimes jumps to maybe 120, and exactly one blue blip is all the way up at 400ms or so.

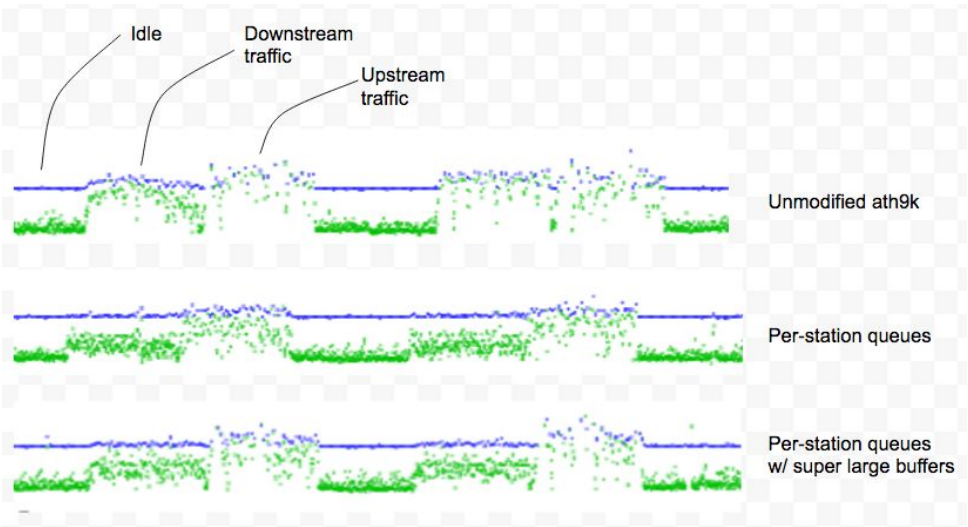
Quiz question: if I'm having a real-time video chat, how much lag will my audio have when talking to green or blue, respectively?

Answer: Many people will calculate that it is (30ms green or 100ms blue) plus a few milliseconds for the frame rate (30fps = 33ms/frame), so about 63ms for green or 133ms for blue. But they've forgotten about jitter. When streaming live video or audio, terrible things happen when a frame arrives later than expected: you have to insert "silence" (or garble, or frozen video) and then either throw away the data when it finally arrives (so you don't fall behind) or else intentionally fall further and further behind after each incident (effectively this becomes prebuffering, so late deliveries should get less and less common because you have higher and higher artificial latency). Different video/audio chat systems use different tradeoffs. But it's safe to say that if you're talking to green and don't converge on the ~90ms (occasional high blips) instead of ~30ms (most common low blips) as your baseline delay, you're going to get a lot of dropped frames. For blue, that ~400ms blip is a real pain, because it's so rare. The conversation might work great *most* of the time, but every now and then, blam, the video freezes temporarily. People hate that. The poor videoconference software has to choose between glitching the video every now and then (in the hope that the big blips are rare) or increasing the constant video delay (because it assumes the big blips are not rare enough to be ignored).

As network designers, it's our job to start thinking about jitter. This blip chart is actually a pretty good one, gathered from an uncongested wifi network (with a wired backhaul) back at my office. At the battlemesh conference where this slide was presented, latency and jitter were all over the map, usually < 150ms but sometimes rising to 4000ms. Throughput was wildly variable too. It was absolutely impossible to have a video or audio call using Hangouts or Facetime.

P.S. I also made a command line tool, isoping, which measures latency, jitter, and also packet loss, even more accurately. As a bonus, it can measure latency in the receive direction separately from the transmit direction, which is insanely handy for debugging wifi. In wifi, one of the two directions is almost always the source of most of your problems, and it helps to narrow it down right away. (Unfortunately, isoping can't run in a browser because it uses UDP, which is not widely supported in browsers. Maybe some webrtc tricks could work, but then it wouldn't work on all platforms...)

Airtime fairness and latency control



Latency effect on station A of simultaneous traffic to/from station B

And this is where we come to airtime fairness and the newfangled per-station queuing(*) that has recently made its way into the Linux kernel (at least for ath9k, and soon ath10k, and hopefully eventually many other wifi drivers). I won't go into a lot of detail about how it works (although it's really interesting; read Toke's et al's paper about it). I'll just show a quick example of the outcome.

This slide is a set of three blip charts with different latency control settings on the (ath9k) access point. The first one is plain ath9k. The second one uses per-station queues. And the third one uses per-station queues with artificially inflated maximum queue size (to show how the new feature mitigates bufferbloat). In each of the three charts, we ran the same test: latency to station A when station B is idle, when station B is downloading, and when station B is uploading, respectively.

In the original driver, when station B started downloading, latency to "green" rose by about 10x (remember, it's a logarithmic y axis, so a bit hard to see). That's a change from about 15ms to about 150ms. Latency to "blue" rose by about 1.5-2x (from about 100ms to about 200ms). When B was uploading instead, performance was completely destroyed, because B hogged as much airtime as it wanted.

With per-station queues enabled on the AP - no extra support is needed on the stations themselves - we get the second chart. During the download section, you can see that blue experiences no visible extra latency at all (in reality it's probably ~5-10ms higher than the baseline ~100ms). Green shows an increase, but because of the logarithmic y axis, that's actually the same delta: about 5-10ms higher, but on

top of 15ms, so it's still only 20-25ms instead of the 150ms we saw before. In the upload direction, things aren't perfectly controlled - remember, we actually didn't change any code on the stations, so they are hogging whatever airtime they want - but because ACKs get somewhat reduced in the return direction, it's still considerably better than with no latency control at all.

The third plot is mostly of academic interest. With very large queues available, there isn't much difference from what happens with very small queues (which makes sense, because the latency control feature intentionally tries not to fill up the queues, preventing bufferbloat problems). The difference is in the upload direction, which we can't control very well. Using larger downstream queues allows more ACKs to build up in the queue and get forwarded all at once, which allows the upstream direction to blast more stuff than we'd like.

Anyway, that's a very long winded way of saying: y'all, we did it! There is no excuse for your mesh's per-hop latency and jitter to be more than two-digit milliseconds. 4000ms is just not okay. Turn on these features!

[Airtime fairness is also nice: it makes it so that one faraway station running at low speed can't crowd out other stations unfairly. But that's actually less important than keeping latency under control, and it's actually a separate feature, so I won't go into it here.]

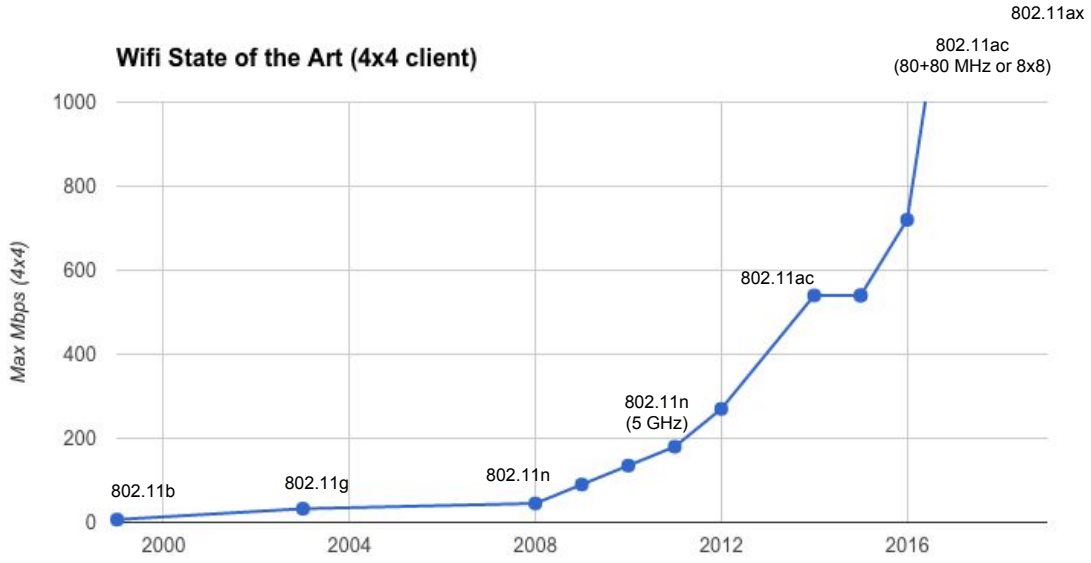
(*) Google Fiber sponsored research into and partial development of the per-station queuing feature.

The Good News

When I do presentations at work, my manager used to warn me, “Avery, everybody likes your presentations, but please try not to be too depressing.” So far, this presentation has been a bit depressing. We talked about how LTE is better and easier, how adding repeaters makes reliability worse, how variable throughput screws up video streaming, and how variable latency screws up live audio/video. Not a great track record so far.

To balance it out, I’m going to tell you what I think is good news. There are fewer slides in this section.

Wifi is getting faster, fast (so is LTE)

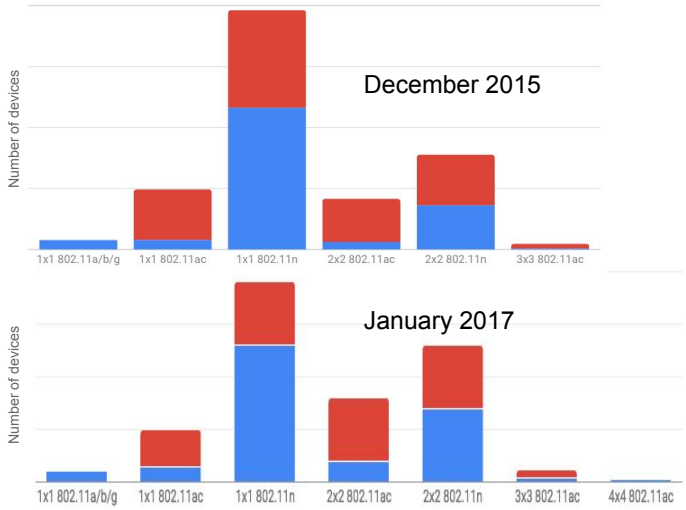


First of all, some straightforward good news: wifi is in fact getting faster. Quickly. In fact, the rate at which it's getting faster seems to be getting faster.

Okay, cool. But I still need to make some disclaimers:
- LTE is also getting faster, at about the same rate.
- This assumes 4x4 MIMO wifi clients, which mostly do not exist in the real world (although you can buy them if you try really hard)
- I calculated UDP bitrates rather than TCP, which is slightly cheating (TCP needs a return channel, UDP doesn't, and wifi is half duplex so that return channel costs you).
- These rates are the true maximum achievable speed at very short distances, like 5-10 feet away. That really does happen sometimes, but not so often. (Still, the speeds at longer distances are increasing along a similar curve, so if you delete the y axis labels, the chart would be about right.)

On the other hand, it could be worse. These rates are definitely lower than the "advertised rates" (also known as "marketing lies") you will find on the cardboard box your routers come in. The rates in this chart are the non-lying, actually achievable real world UDP bitrates. There are actually wifi products that can do >1 Gbps from 5 feet away now, no kidding. Welcome to the future!

(But plenty of devices still suck)



Um, I know this is the good news section, but I thought I'd include an overview of about what fraction of devices support what features and what the trendline is. Sorry for the microscopic font sizes.

This data is from Google Fiber's network on the given dates. Blue corresponds to 2.4 GHz, and red is 5 GHz. The two y axes are normalized, so the tallest bar (1x1 802.11n) actually represents a different absolute device count in each chart. What matters is the relative bar heights in each chart, not the absolute height.

The main thing to observe is a noticeable increase in 2x2 802.11ac and 2x2 802.11n connections, relative to 1x1 802.11n, from December 2015 to January 2017 (about 1 year difference). The main thing that happened is that some major phone manufacturers started including 2x2 MIMO in almost all their new phones, and phones are a huge fraction of wifi activity nowadays.

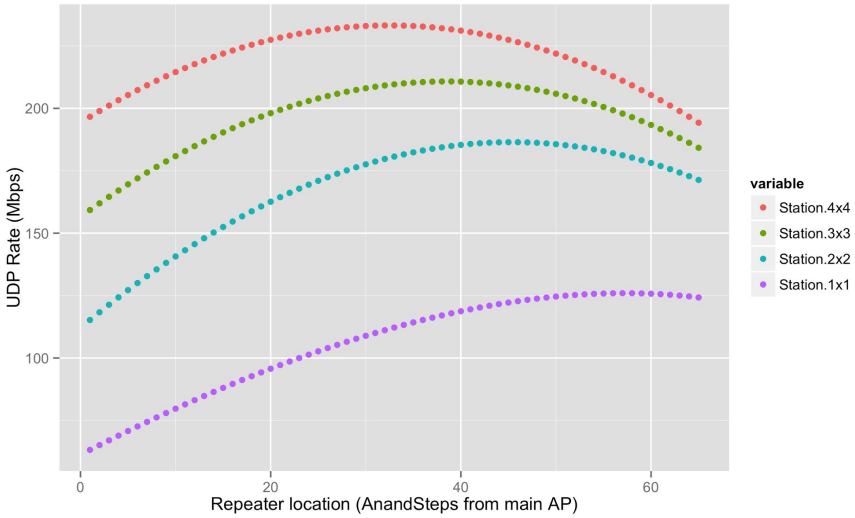
(The fraction of devices on 2.4 GHz is surprisingly high for the 802.11ac category, but at least for 802.11ac-capable chips, that's mostly caused by 5 GHz-capable devices that make poor roaming decisions.)

So, bad news: the previous chart of 4x4 performance isn't too meaningful yet since almost no client devices have 4x4.

But good news: 1x1 devices are rapidly getting eclipsed by 2x2 and higher. 2x2 is twice as airtime-efficient as 1x1, which makes a huge difference with congestion. (It

also slightly improves maximum range, via antenna diversity.)

Throughput: (4x4) Repeater placement matters



Here's a slide I thought was really important but I couldn't think of a better place in the presentation to include it. Sorry if it seems out of place.

These are simulation results for how far apart to place your mesh nodes. It's a little confusing, but it'll be important a few slides later, so let's try to go through it.

First of all, my unit of distance measurement in wifi is what I call AnandSteps, named after my former intern who I made do all these experiments. An AnandStep is how far Anand goes when he takes one step. It's an intentionally vague distance. You've never met Anand, and sometimes he walks fast and takes big steps. Sometimes he takes small steps. Sometimes he goes around corners. The point is to not get too hung up on exact distances, because with wireless systems, exact distances are often less important than other random characteristics of the environment (reflections, background noise, walls, etc). So let's just say that AnandSteps are a linear-ish measure of distance.

In this chart, there are three nodes:
- Primary wifi AP (4x4 MIMO): 0 AnandSteps
- Wifi repeater node (4x4 MIMO, echoing between station and primary AP): "x" AnandSteps
- Wifi station (1x1 to 4x4 MIMO): ~70 AnandSteps

There is one line for each type of station (1x1 to 4x4), but in all cases, the APs are 4x4. The line corresponds to what happens as you move the wifi repeater from

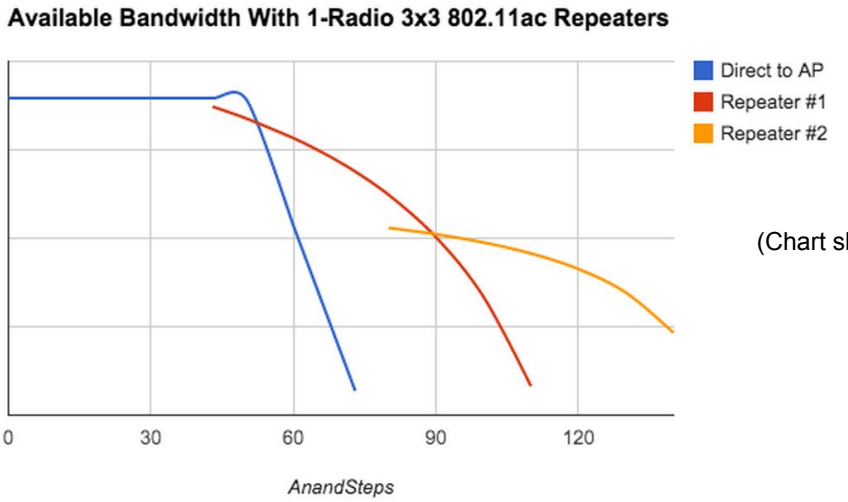
location 0 (right on top of the primary AP) to location 65 (very close to the station). The y axis is achievable throughput from the station to the primary AP (and thus presumably the Internet).

Unsurprisingly, with the repeater at location 0, performance sucks; you might as well have no repeater. As the repeater moves toward the right (further from the primary AP, closer to the station), performance goes up, peaks, and then starts declining as the repeater gets “too close” to the station (and thus “too far” from the primary AP).

Here’s the really important insight that we’ll need later: a 1x1 station is near peak performance when the 4x4 repeater is right next to the station. That’s because 4x4 MIMO gets 4x the performance of 1x1. So you want most of the distance travelled to be at 4x4 (between the two 4x4 APs) rather than at 1x1 (between the station and some other device). With a 2x2 station, the ideal repeater location is maybe 2/3 of the way along; closer to the station, but not right on top. With a 4x4 station, the ideal distance is roughly halfway, which makes sense since both hops are 4x4.

I don’t think I can emphasize this enough. **4x4 MIMO repeaters help even with 1x1 stations.** They help a lot. We’ll talk more about this in a bit.

Performance effect of adding a repeater



(Chart shows 2x2 802.11n client)

Here's another slide using some of the same simulation/experimental data. This time we use 3x3 repeaters (inconsistent, I know, but I prefer to recycle my old charts rather than drawing new ones all the time!) and the x axis is now the location of the station, rather than the repeater. At each point on each line, we assume the repeater(s) are placed "reasonably well" (not quite optimally, but somewhere not too far off) according to the chart in the last slide.

In this chart, we choose to simulate a 2x2 802.11n client (which is pretty common today) to show some interesting effects.

The first interesting effect is the blue line. That's what happens if the client talks directly to the primary wired AP. Notice that up to 45 AnandSteps or so, the speed is at max. That's because 802.11n doesn't take advantage of extra signal-to-noise ratio (SNR) beyond a certain point; this wastes spectrum. An 802.11ac client, with the mysterious new MCS8 and MCS9 encoding levels(*), wastes less airtime if you're lucky enough to be close to the AP.

Beyond a certain distance, direct (blue line) throughput starts dropping off fairly fast. This shows an advantage to adding a repeater (red line, Repeater #1). At certain distances, you're better off doing two short hops rather than one big hop. And as you get even further away, a second repeater (yellow line, three hops) starts to give better speeds.

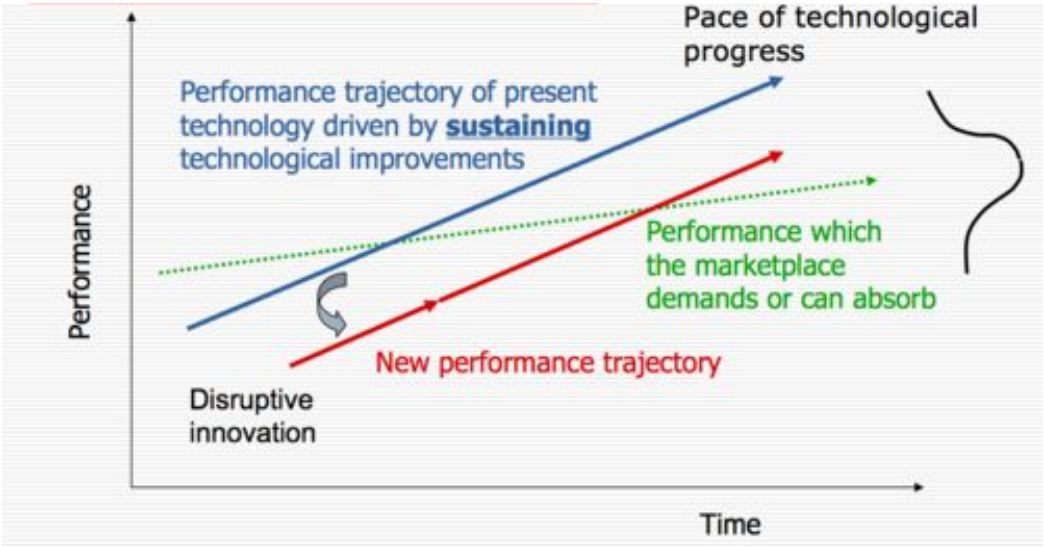
Note: the typical math that says throughput drops by 1/2 with each repeater you add

(ie. 1/2^n) is just wrong; it's more like 1/n. On the other hand, the increase in useful range from adding each repeater is fairly disappointing: about 50% more range. You can stretch it more, but then your throughput drops off *really* fast and performance sucks. It's better to have a dense network of many repeaters rather than repeaters stretched over too-long links.

(*) Note that 802.11ac MCS8 and 9 are not the same as 802.11n MCS8 and 9. In 802.11n, the standards people committed a major error, defining MCS0-7 as 1x1, MCS8-15 as 2x2, MCS16-23 as 3x3, etc. This left no room to add new modulation levels at each number of antennas. In 802.11ac they realized their mistake, undefined MCS8 and up, then redefined MCS8 and 9 as new 1x1 encoding levels. In 802.11ac, you always state the number of MIMO streams separately: 1x1 MCS9, 4x4 MCS4, etc.

The Wireless Innovator's Dilemma

(Image via coderinsights.com)



I asked at battlemesh how many people had heard of the book The Innovator's Dilemma. It was surprisingly few, at least compared to business/corporate-type people I usually talk to. Anyway, I recommend that book. I read it far too late in my career, and it's almost magical at explaining strange phenomena in the computer industry that otherwise go unexplained. This book is the origin of the term "disruptive innovation." In the spirit of all good clichés, that term used to have a really useful meaning that has since been lost. In particular, they wanted to compare disruptive innovations with the much more common "sustaining innovations." The big difference is that the existing industry leaders are really good at sustaining innovations. Making x86 chips faster, or SSDs or hard disks store more stuff, or whatever, are all things that market leaders are good at. What goes wrong is when something **worse** becomes **good enough.** This confuses the leaders every single time. Think of SSDs vs spinning hard drives (mostly different manufacturers) or ARM (slower but power efficient) vs x86 (fast but inefficient).

What happens is that the market leader ignores the new thing until it's too late. When ARM came out, it was tiny, slow, cheap, and low-power. You'd never imagine trying to make a "real" computer with it, because the computer would suck. A bit later, people tried to do it anyway, and sure enough, it sucked. But ARM slowly ate up the low-end areas of computing where low-cost, sucky computers were sufficient. And every year, ARM chips got a little faster. One day, they got so good that you could build a smartphone, and then smartphones got more common than desktop PCs, and now x86 makers should be worried.

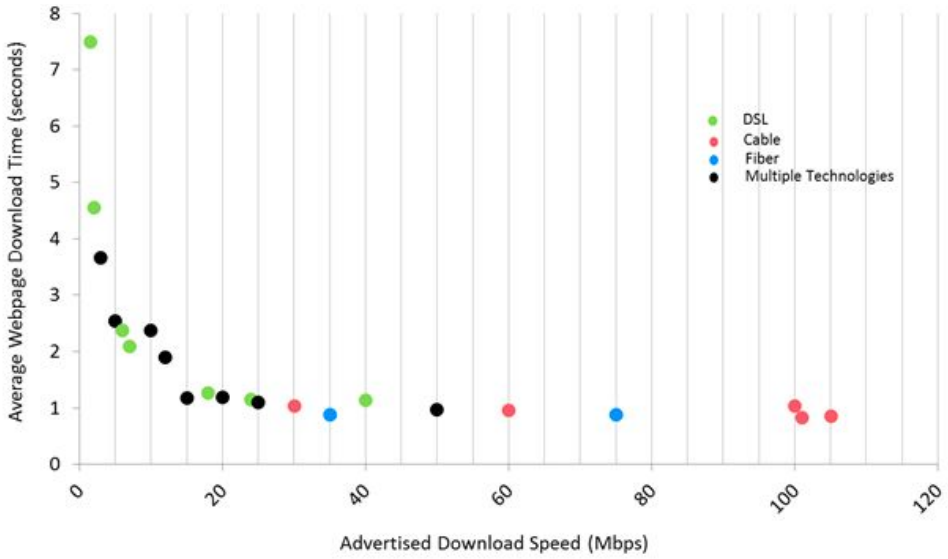
The essential ingredients for an Innovator's Dilemma are:
- Two competing technologies, which both get better over time
- Customer requirements, which gets higher over time
- The first technology exceeds customer requirements, but the second one is not good enough
- But the customer requirements are increasing **slower than the technology is getting better**

When this happens, it's almost inevitable: eventually, the second technology will exceed the customer requirements. When that happens, suddenly you don't need the first technology anymore.

I want to claim that LTE vs Wifi Mesh is an Innovator's Dilemma. LTE and Wifi are getting faster very fast, maybe at the same rate, probably both according to some variant of Moore's Law. But customer demand is increasing slower than that. There's only so much throughput a consumer needs. Once you have 60fps, 3D, 4k or 8k live video, that's pretty much all you can consume. And that's only, say, 100 Mbps or so per stream. I mentioned a couple of slides ago that wifi (albeit not a mesh, and not at long distances) can already do more than 1 Gbps with the latest chips. And theoretical maximum throughput demand isn't even the whole story...

Bandwidth demand is limited

from FCC broadband report



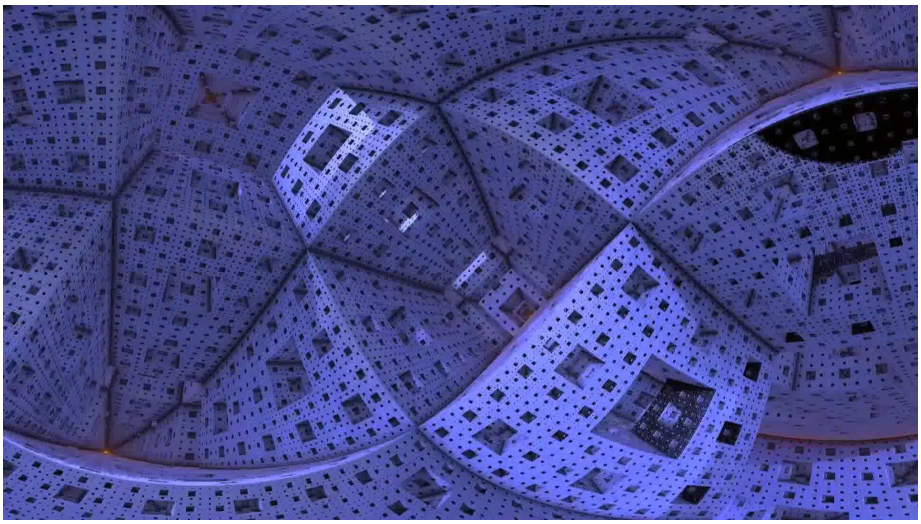
Here's one of my favourite charts, from a report put together by the FCC (see link above). It shows that around 25 Mbps or so, average web page load times stop improving. That's because web pages tend to have a lot of back-and-forth requests going on, and the load time ends up dominated by latency instead of throughput.

25 Mbps! LTE can do that now in most places. Wifi can do it pretty reliably, on 5 GHz, within a reasonable distance from the AP. The meshes I tried at Battlemesh v10 mostly couldn't keep up with 25 Mbps, but then again, most of them are still 802.11n or 802.11a/g based. 802.11ac is twice as fast! We're almost at peak.

And anyway, a decent 1080p video stream can be obtained in about 5 Mbps, which we can do easily right now, as long as we're careful about throughput variation, latency, and jitter, like we discussed in earlier slides.

Six magic ingredients (airspacetime)

- Time
- Space
- Spectrum
- Modulation
- Beamforming
- **MIMO**



[The "Beamforming" link above is to my javascript beamforming simulator. It's fun.]

In networking we talk a lot about throughput and latency, but in wireless, those aren't the fundamental measurements. Slightly more advanced discussions talk about airtime - the amount of time your radio is on the air, blocking out other people - which is useful because it helps figure out total throughput when there are multiple stations. For example, if you have two faraway stations talking at 1 Mbps and a nearby one capable of 100 Mbps, what total data transfer do you get if they're all sharing the link? In one of the more reasonable worldviews (airtime fairness), the answer is:

1 / [1/(1Mbps) + 1/(1Mbps) + 1/(100Mbps)] = 49.75 Mbps

Counterintuitively, you have to do it that way. You can't just take the average of the three bitrates:

~~(1Mbps+1Mbps+100Mbps)/3 = 34 Mbps~~ [DON'T DO THIS]

because that's not meaningful physically.

However, airtime is not the whole story either. A simplistic view of airtime is that if one node is sending, no other node can send at the same time. But that's not true! Besides taking turns ("time" multiplexing), a wifi node in Germany can transmit at the same time as one in Canada. That's "space" multiplexing. Nodes can also be on different channels ("spectrum" multiplexing).

There are also some more complex effects. Like we saw a few slides ago, adding new modulation levels (MCS8 and MCS9) can pack more bits into the same amount of time and spectrum; let's call that increasing "density" via better modulation, up to

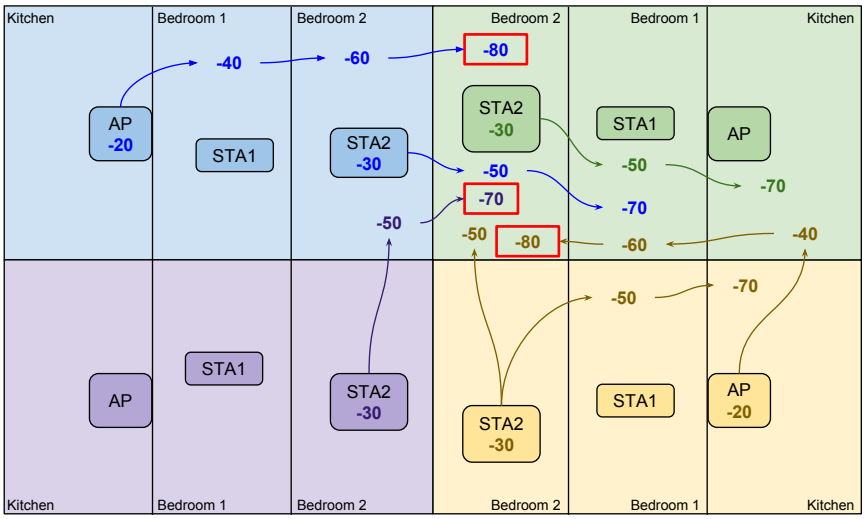
the Shannon limit. You can also squeeze a little harder on the Shannon limit by improving signal-to-noise ratio (SNR) at the receiving end via beamforming, further increasing density. And weirdest of all, if you have multiple antennas on the sender and receiver, and the signals from each sending antenna follow a different path to the antennas at the receiver, you can use MIMO to send multiple non-overlapping signals, through the same space, at the same time, on the same channel.

[Even weirder: you can't do MIMO with a wired link, unless you have multiple wires. But nobody has yet found a theoretical limit for how many wireless MIMO streams a particular volume of space can support. High-end wifi right now can do 4x4, and soon 8x8, but that's not even close to the maximum limit. That means wireless could someday actually go *faster* than wired links using the same spectrum. Maybe insanely faster.]

I call this multidimensional expression "airspacetime." To depict it, I downloaded an image from Google Image Search. It is apparently a hypercube, viewed through what is apparently a hyper-fisheye lens, so that you can see it on your mere 2-dimensional display.

If you want to understand how wireless meshes could eventually rule the world, you need to be thinking of airspacetime, not airtime. The next slides will show what I mean.

Transmit power control saves **space**



This is a map I've used in previous presentations. It represents the layout of a simplified apartment complex. For our purposes, the important thing is that the signal in one apartment interferes with the reception in the neighbour's apartment. In fact, even Yellow's AP talking to Yellow's STA1 causes interference with Green's STA2. This is an airspacetime conflict that reduces total throughput available to all four apartments in the diagram.

We can do better, at least a little better. Thomas Hühn's Minstrel-blues transmit power control(*) algorithm transmits at lower power to nearby stations, and higher power to faraway stations, thus reducing the "space" used when talking to the nearby stations (STA1 in the diagram). Reducing space reduces airspacetime, which frees up airspacetime for other people to use, at no real cost.

This sounds obvious, yet in the wifi world it previously ranged from rare to unheard-of.

(*) Google Fiber sponsored work to bring minstrel-blues to ath9k (and hopefully eventually ath10k)

Wider channels save **time**

- We should use channels 1, 5, 9, 13 instead of 1, 6, 11
- Sending 1M on an 80 MHz channel takes the **same spacetime** as on a 40 MHz channel, **but less time**

  "Bandwidth is the ultimate perishable resource; use it right now, or it's gone forever." -- Stuart Cheshire

Saving the space part of airspacetime is fairly straightforward (once you accept that reducing power means you take less space). This one is slightly trickier.

My claim is that if you send the same data over a double-wide channel (40 MHz instead of 20 MHz, say), and thus take half the time (because thanks to the extra spectrum, it goes at twice the speed), then you are overall better off.

But didn't we just finish saying that airspacetime is one single thing? If we use twice the spectrum and half the time, the total volume of airspacetime is the same, right?

Well, yes. But there's a catch. I once saw a talk by Stuart Cheshire in which he said something like, "Bandwidth is the ultimate perishable resource; use it right now, or it's gone forever." Unlike the other elements of airspacetime, the "time" component is special. If nobody uses a particular timeslot, it's wasted. Forever.

Let's imagine we have a channel capable of 16 Mbps (2 MBytes/sec). Station A wants to transfer 20 MBytes. Station B wants to transfer 2 MBytes. If they share the big channel, the total will take 22 MBytes / 2 MBytes/sec = 11 seconds. But if we give them each a half-sized channel, each 8 Mbps (1 MByte/sec), then the transfer will instead take 20 seconds on channel A, and (simultaneously) 2 seconds on channel B. Channel B will be idle (wasted) for the other 18 seconds. The total airspace used will be the same, but the airspacetime consumed will be double what it was with one wide channel.

Weird, right?

Now, people who have being doing this mesh stuff for a while will rightly complain, “Hold on, what about hidden nodes?” That’s a good point. (Follow the link if you don’t know what hidden nodes are.) However, it’s becoming less of a good point over time. First of all, there are not enough wifi channels; you will have hidden nodes no matter what you do. Secondly, 802.11n and later use aggregation (much more efficient channel utilization), which reduces the impact of hidden nodes. Thirdly, a lot of problems with hidden nodes were actually caused by naive rate control algorithms(*) that died badly when they encountered interference. Nowadays, most chipsets have that problem fixed. Hidden nodes are not as big a problem as they used to be.

But even more importantly, this advice - using fewer big, wide channels instead of more narrow channels - decreases channel utilization. If we assume, as discussed in the earlier slide, that user demand for throughput is increasing slowly, but capacity is increasing quickly, then we can also assume that channel **utilization** (the % of time the channel is in use) will eventually start to fall. As utilization falls, the probability of a collision with a hidden node falls even faster. If we trust in this effect, then we should trust that using wider channels, and thus decreasing utilization, will make it come true even sooner. And that’s why the IEEE wifi people are not crazy to be talking about 80 and 160 MHz wide channels that eat almost the whole 5 GHz spectrum allocation all at once. It’s just good math!

This leads me to one more side note: this is good advice even on 2.4 GHz, where traditionally 40 MHz wide channels are almost impossible due to historical channel allocation mistakes. Typical industry practice has slowly converged on using channels 1, 6, and 11 as the “main” 2.4 GHz wifi channels. This is based on the ancient 802.11b modulations, which could sometimes use up to 22.5 MHz per channel (about 4.5 channels wide). Modern 802.11n and later modulations only use 20 MHz for channel, which is why the 802.11 spec says to space wifi 4 channels apart (20 MHz each): we no longer need to care about 22.5 MHz. It would actually be best if we followed the same rules on 2.4 GHz. That would give us several benefits: first, if only channels 1 and 5 are used in the bottom half of the range, we could combine them and make a 40 MHz channel; second, in countries where wifi channel 13 is allowed, this new spacing fits channel 13 without overlap, and that could combine with channel 9, giving a *second* 40 MHz channel.

I’ll probably never be able to convince people to stop using 1, 6, 11. But if you’re writing a channel selection algorithm, I suggest helping out the world a bit by at least using 1, 5, 11 (North America) or 1, 5, 9, 13 (in regions with channel 13) instead, where channel 6 isn’t already being used.

(*) Very briefly: a naive rate control algorithm will transmit at a slower rate if packets

*are getting lost, and a faster rate if everything is perfect, based on the assumption that if packets are getting lost, there must be some background noise. That makes sense for pure white noise, but it doesn't work with bursts of interference (such as from hidden nodes): if there are short bursts of interference destroying some of your packets, you will actually do better by *increasing* the transmission rate, because then your packets will be smaller, and thus less likely to overlap with an interference burst! Modern rate control, such as minstrel, now does this.*

MIMO is free!*

- More MIMO streams take **less spacetime** for the same content
- ...although you need fancier silicon
  (but chips keep getting cheaper)
- Moore's Law still continues here

* *Except it costs money*

Let's talk a bit more about MIMO, because MIMO is completely amazing. I don't think enough people realize how amazing MIMO is.

If you have 2x2 MIMO, you can fit twice the bits in the same amount of airspacetime compared to 1x1. If you have 4x4 MIMO, that's four times the bits. And so on. And there are no known limits to how far you can scale it, other than some pretty minor things (eg. your antennas need a certain minimum distance between them). That makes increasing MIMO usage "free" in terms of airspacetime: you can keep using more MIMO without interfering with anybody else.

There are some restrictions though:

- The more MIMO streams you have, the more math is needed in order to do the modulation. I'm not sure, but since receiving the signal effectively involves inverting an NxN matrix, I think the computing power required may increase slightly faster than O(N) with the number of streams. Luckily though, Moore's Law seems to be continuing in the world of massively parallel digital signal processing (despite having slowed down in single-threaded processing). There's a lot more room for more MIMO streams.

- Antenna placement makes it really tough to fit more antennas in tiny mobile devices like phones. Phones are probably going to be stuck at 2x2 for a while. Tablets and laptops have lots of room for more antennas though, if demand gets high enough (or costs drop low enough, which they will).

Still though, remember from a previous slide that even if your phone never hits 4x4 or 8x8 MIMO, you'll still benefit if your wifi mesh uses 4x4 or 8x8. What that means is we want a world where there are lots and lots of APs, maybe even one per room, in order to deliver the signal as close as possible to the end-user device so that we waste as little airspacetime as possible talking to cheap devices with fewer MIMO antennas.

Multi-radio repeaters are mostly obsolete

Quiz: which is better?

1. A 2.4 GHz 2x2 radio + a 5 GHz 2x2 radio

   -- or --

2. One 5 GHz 4x4 radio

   ?

And now, let's tie it all together. Based on what we just learned, which one of these two situations is better? Two 2x2 radios or one 4x4 radio?

When I asked this question at Battlemesh, someone pointed out that if your client device is 2.4 GHz-only, then you really need the 2.4 GHz radio. Okay, fair enough. But those devices are slowly declining. (If you really need them, and slow LTE isn't good enough for them, you could strategically add a few 2.4 GHz radios here and there to cover the ancient devices. You don't need too many 2.4 GHz APs since most devices will migrate to 5 GHz, leaving only a few left fighting for spacetime.) And for the purposes of this slide, let's think of just a mesh backhaul.

The other people who answered also picked #1 though, for reasons that all revolved around trying to avoid channel overlaps and interference.

Here's my claim: long term, the correct answer is #2. Each "stream" costs about the same, whether it's divided into two 2x2 radios or one 4x4 radio. The total compute power (4 streams) and total antenna cost (4 antennas) is the same in both cases. But in case #2, you have twice the airspacetime density of case #1. You can pack twice the bits in the same amount of airspacetime! That means you can have either twice the total throughput, or half the utilization (and thus far fewer collisions with hidden nodes). It should be no contest, as long as we get all the necessary parts (especially rate control and transmit power control) working.

The net result is that the whole concept of multi-radio repeaters is becoming obsolete.

It’s just kinda wasteful to be thinking about old-fashioned wavelength division multiplexing (multiple channels), rather than dumping everything onto one giant channel, with a single radio, and using time division multiplexing instead. As an added bonus, this makes the electronics, mesh routing, and channel selection simpler.

The exception is if you really want/need to take advantage of multiple completely different spectra, like 900 MHz, 2.4 GHz, 5 GHz, 30 GHz, 60 GHz. Right now that needs multiple radios. Maybe someday, if software-defined radio (SDR) gets cheap enough, even that can all be done with a single giant MIMO radio. Well, at least we can dream.

Summary

- The next battle isn't between meshes, it's between mesh and LTE.
- We're at a supply/demand intersection point.
  If there was ever a time to build big, fast wifi meshes, it's now!
- Use more MIMO streams! If we cut congestion, everything works better.
- Always use latency/jitter controls on your queues
  ...and turn on transmit power control if you can.

apenwarr@gmail.com

MIMO MIMO MIMO. Start installing mesh routers with as many MIMO streams as possible. That could totally change the equations around mesh networking.

And turn on those newfangled latency control features! They really help more than you think!

The end.