The current market Alpha enjoys with rideshare missions will not continue into the future because they are underpriced to account for the risk of early launches. These previous launches are underpriced in order to find customers for initial launches. With increased prices in the future, the primary reason to conduct this type of rideshare mission is an accelerated launch date compared to Falcon 9 rideshare missions. Since 2022, Electron has launched a single rideshare mission of this type.

Electron's other rideshare missions have either had unique requirements or were payloads tagging along with a larger primary payload. The recent "Beginning Of The Swarm" mission launched the NeonSAT-1 and ACS3 satellites to different orbits on a single mission by utilizing the Photon kick stage. About a year before this, Rocket Lab launched the "Baby Come Back" mission with the 30kg Teleset LEO 3 satellite and various smaller payloads for Spire Global and NASA. Firefly will be able to launch these types of missions with their Elytra kick stage and by ridesharing small satellites with larger primary payloads.

The market most well suited to Alpha is ~1-ton satellites. Judging from the earlier Falcon 9 missions, there may be one satellite per year that Alpha will be able to launch in this market segment.

The US Military has not made public how many responsive launches they will conduct in the future. This makes it difficult to predict the size of this market in the future. Given the competition betwene Firefly and Rocket Lab for these launches along with the developmental nature of these contracts, Firefly could see a responsive space mission every 1-2 years. Please argue with me if you have a differing view.

By far the market with the largest potential for Alpha is constellations. Currently, there are no constellations that have satellites massive enough and launched in enough numbers to take advantage of the payload advantage of Alpha, so we can only speculate on Lockheed Martin and L3 Harris' contracts and extrapolate from Electron. During the first 7 months of 2024, Rocket Lab conducted 7 constellation launches. According to Wikipedia, they have 23 planned constellation launches between 2024-2027. Given the fact that Electron will likely sign more contracts before 2027, the number of constellation launches over this timeframe should be 10-15 per year.

Earth Observation constellations don't appear to require Alpha's 1-ton payload capacity, but military constellations may be well suited for it. Lockheed Martin and L3 Harris both contracted Alpha for about 5 launches per year during a 4 year period. These periods have slight overlap, 2025-2029 and 2027-2031 respectively. We don't know what payloads are planned for these launches and during such a preliminary stage it is unlikely large sums of money have changed hands. For these reasons, the launch contracts should be discounted for the fact that they may not materialize. Around 3 constellation launches per year (Either military, large earth observation satellites, or other) is a reasonable estimate.

Conclusion

Summing the potential launches listed above, we get demand for about 6 launches per year. The breakdown is 3 Constellation, 1 Dedicated Satellite, 1 Responsive Space, and 1 Rideshare. Assuming an average price of $15M, this is a $90M market. Comparing this to ~20 $8M Electron launches per year, Firefly's TAM is about 50% of Electron's upcoming launch rate. This assumes that Firefly realizes the constellation market and that Electron launches don't substantially increase. Specifically looking at constellations - as they are the largest and fastest growing market - Firefly could launch 3 missions per year versus Electron's ~10-15. The higher price of an Alpha flight makes the gap smaller than the launch counts imply, 60% vs. 80%.

All the payload categories I've delved into above should make it clear that there is little overlap for competition between Electron and Alpha. Similar to Electron vs. Rideshare, Alpha is not cheap enough to take Electron's core market. Rather, it will create its own niche just as Electron has against Falcon 9 Rideshare.

How to Land an Orbital Rocket Booster with kOS

Jul 24, 2024 • Christopher Kalitin

Tell me where I'm wrong or just give compliments here.

Read the code here.

One of the reasons Casey Handmer cites when telling people to write blogs is that they are proof of work. Many of my early projects will not be impressive at all, but it's worth documenting it for a few reasons. (1) To share my thought process throughout the project. (2) To force myself into documenting it. (3) To be able to reflect in the future. Forcing yourself to document something means it's much harder to be satisfied with the shitty way of solving a problem. This is why everyone should write a blog.

Can't think of a good way to write this, so I'll just detail all the mistakes I made and stupid things I did. You have to start somewhere!

I could write many thousands more words about this about the suicide burn, flight phases, printing, pitch mulitplier, and more. But all those things are less interesting and not what I want to cover. I'm not going to explain the simple stuff, this isn't a tutorial.

Initial Aero Control Approach

```
// Get distance between two positions without considering the altitude
// Eg. LatLngDist(V(SHIP:GEOPOSITION:LAT, SHIP:GEOPOSITION:LNG, 0), V(-0.09729775,-74.55
function LatLngDist {
    // Only x and y are used for lat/long. z is to be ignored
    Parameter pos1.
    Parameter pos2.

    // 10471.975 is the length of one degree lat/long on Kerbin. 3769911/360
    return (pos1 - pos2):MAG * 10471.975.
}

// Return direction to position in degrees starting from 0 at north
function DirToPos {
    // Only x and y are used for lat/long. z is to be ignored
    Parameter pos1.
    Parameter pos2.

    SET diff to pos2 - pos1.

    // atan2 resolves arctan ambiguity (ASTC quadrants)
    // Reversing x and y to rotate by 90 degrees so we start at 0 degrees at north, usua
    SET result to arcTan2(diff:X, diff:Y).

    // Keep degress between 0 and 360
    if result < 0 { SET result to result + 360. }

    return result.
}

// Return east/west and north/south components of velocity
function GetVelocityInCompassDirections {
    // https://www.reddit.com/r/Kos/comments/bwy79n/clarifications_on_shipvelocitysurfac
    SET vEast to vDot(ship:velocity:surface, ship:north:starvector).
    SET vNorth to vDot(ship:velocity:surface, ship:north:forevector).
    return v(vEast, vNorth, 0).
}
```

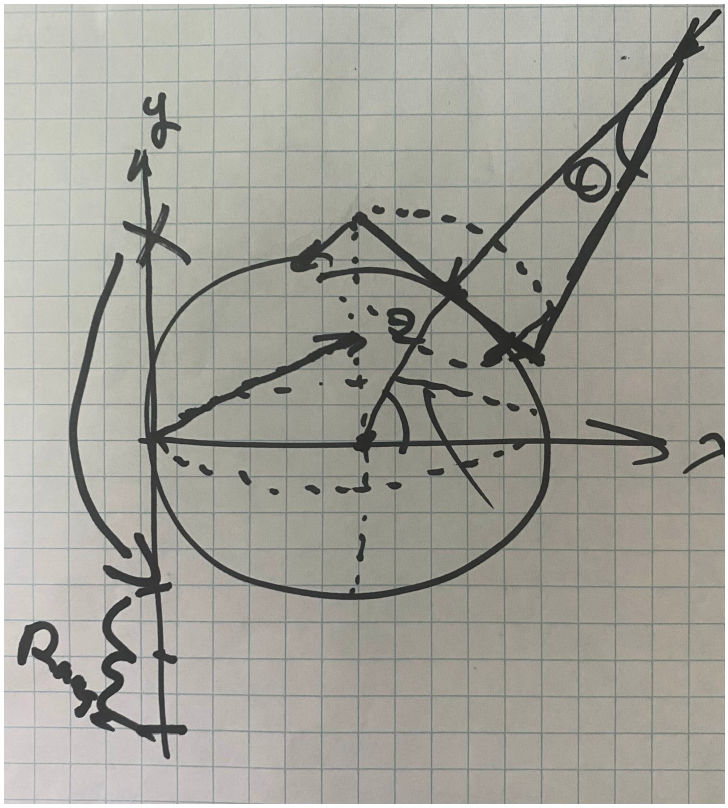
When I first attempted to land a booster with Kerbal Operating System about 2-3 years ago I got stuck trying to implement aerodynamic control. So, this is where I started.

In the intervening years I became a much better programmer and quickly implemented the helper functions above, kids stuff. The principle is to get the direction from the target point to the impact point and pitch in that direction depending on what the required change in distance per second is.

The horrible problem emerges when you realize that the DirToPos function returns the direction from one point to another on the surface of Kerbin. This value is then used as the bearing (degrees relative to north) for the booster.

Anyone who has attempted to land a booster in KSP knows that to adjust your landing site you have to adjust your pitch relative to the retrograde vector. The method above pitches relative to north (bearing relative to north, pitch is used as “amplitude”). At the scale I was testing at (~5000m above the Kerbal Space Center), this problem was not obvious. However, when attempting to reenter after a boostback burn the error became clear.

Failing to Clamp Angle Relative to Retrograde



In kOS (Kerbal Operating System, the scripting mod/language), you control the direction of the craft by inputting a heading which consists of a bearing and pitch value. This is easy to conceptualize for beginners as it’s similar to the Nav Ball in KSP. However, what I didn’t realize fast enough is that you don’t want to do any operations on the heading. There’s a reason why you use heading as a pilot and not when learning trig.

Imagine you're trying to clamp your booster's pitch to within 10 degrees of the retrograde vector. You only have the heading (bearing, pitch) value to work with. Pitch = 0 when straight up, bearing = 0 when pointing north. Stop and think how you would do it. Don't be lazy, do it.

Well if you're rocket is horizontal, clamping the raw bearing and pitch values will work great. A 10 degree offset in either bearing or pitch will result in a displacement of equal magnitude. However, if you're not horizontal (on the equator), a 10 degree change in bearing will result in a smaller displacement than a 10 degree change in pitch. This is the same reason why Vancouver is rotating around the Earth slower than Equador. Hopefully the diagram above makes this as clear as the diagram should be. If you increase pitch (closer to a pole), a single degree of bearing becomes shorter. When you're point straight up, a single degree of bearing is 0.

This problem took a few days to solve (because I don't know that much math, university solves this) and it all stemmed from the initial approach I used for heading control.

tl;dr this is very stupid:

```
function GlideToTarget {
    local aproxTimeRemaining to (SHIP:altitude - TargetPosAltituide) / (SHIP:velocity:su
    local targetChangeInDistanceToTargetPerSecond to impactToTargetDistance/aproxTimeRem

    // If impact dist < 50, do fine control that asymptotically approaches the target (b
    local pitchMultiplier to targetChangeInDistanceToTargetPerSecond * 2.
    if impactToTargetDistance < 50 { SET pitchMultiplier to (impactToTargetDistance^1.6)

    if RetrogradePitch > 70 AND ship:velocity:surface:mag < 450 { SET bearingLimit to 36
    else SET bearingLimit to pitchLimit.

    local shipDirToTarget to impactToTargetDir - 180 - RetrogradeBearing.
    local bearingAndPitch to GetBearingAndPitchFromDir(shipDirToTarget, pitchMultiplier)
    LOCK STEERING TO HEADING(bearingAndPitch:x, bearingAndPitch:y).

    // If retrograde pitch is nearing straight up, behaviour is not correct, so, lock to
    if RetrogradePitch > 80 {
        LOCK STEERING to HEADING(shipDirToTarget + RetrogradeBearing, pitchMultiplier).
    }
}
```

How to Properly Control Heading

Hopefully it's clear that bearing and pitch are horrible values that only pilots should ever use. In

math class you use Eulers, Cartesians, etc. for a reason. The solution is to convert the bearing and pitch to a more suitable rotation system, clamp it, then convert back to bearing and pitch.

A very important insight I heard about learning to code is that half your time should be spent coding and the other half should be spent reading code. There are tons of people who've solved the problem you're working on, it is far more efficient to learn from them. This doesn't mean copy and pasting code, but truly understanding the problem and the solution.

I went back to the video that prompted me to try kOS again and found the solution. Turns out Donies did things the shitty way instead of truly learning (well everyone starts somewhere) and copied the code Edwin Robert wrote here (Video).

```
// I overengineered for 5 wasted days, this is the solution from: https://github.com/Dor
// This functions steers the ship relative to retrograde towards the target position, it
function GetSteeringRelativeToRetrograde {
    local Parameter pitchMultiplierLocalSteerRetrograde. // Local variable naming like t

    // Retrograde vector is in the SHIP-RAW Reference Frame https://ksp-kos.github.io/KC
    local retrogradeVector to -ship:velocity:surface.

    // :position converts from latlng to SHIP-RAW reference frame
    // Refactoring needed to minimize transforming values like LatLng
    local targetVector to ImpactPos:position - LATLNG(TargetPos:x, TargetPos:y):positior
    local targetDirection to retrogradeVector + targetVector * pitchMultiplierLocalSteer

    // If relative angle is too high, limit it.
    // Normalize the vectors, then multiply the target direction by the tan of pitch lim
    local angleDifference to vAng(targetDirection, retrogradeVector). // Angle of two ca
    if angleDifference > PitchLimit { SET targetDirection to retrogradeVector:normalizec

    return lookDirUp(targetDirection, facing:topvector).
}
```

Converting to the SHIP-RAW reference frame is the key. This allows for standard operations to be done on the rotation vectors and to use functions included in kOS like vAng.

The vAng function abstracts away some concepts I don't yet understand. Without it I would've had to study trig for a few weeks to properly implement it. Projects like this are great because they clearly show the extent of your knowledge. "retrogradeVector:normalized + targetDirection:normalized * tan(PitchLimit)." makes perfect sense to me and I can draw the diagram for you, but what goes on inside vAng is a mystery for now.

Failed Refactor

From the use of periods instead of semi-colons and other quirks like “local Parameter”, you might be able to tell that kOS is not a language meant for programmers, but rather for KSP players. This means a new mental framework is required to use kOS efficiently.

When implementing the intial helper functions and first attempt at landing I decided to use the techniques I was aware of and refactor in the future. This is immensely stupid and leads to a lot of wasted time that you will have to deal with in the future. Just write the code properly the first time.

The problem with the existing code is it used SET instead of LOCK on variables. In kOS you can declare a variable the way you’re used to by using SET. LOCK is used to update the value of a variable every physics tick. kOS is obviously supposed to be used with LOCK and my attempt at a refactor changed the code to use this different paradigm. The fundamental solution here is - of course - to rewrite kOS to be a proper language, but I ain’t doing that.

This refactor took more time than expected because I had to port the code to an entirely different execution paradigm.

Previous SET paradigm:

1. Create an infinite “UNTIL false” loop and keep a variable to track flightPhase.
2. Depending on the current flight phase, execute the appropriate function.
3. Break the loop when we’ve landed.

```
// directionError is SET in OrientForBoostback()

UNTIL false {
    if NOT ADDONS:TR:HASIMPACT { LOCK THROTTLE TO 0. CLEARSCREEN. BREAK. }

    UpdateFlightVariables().

    if flightPhase = 0 {
        PRINT "Flight Phase: Orient For Boostback (1/6)" at (0, 0).

        OrientForBoostback().

        PRINT "Flight Variables: " + numberHere at (0, 2).

        if directionError < 30 { StartBoostbackBurn(). }
    }
```

```
}

function StartBoostbackBurn {
    LOCK throttle to 1.

    SET flightPhase to 1.
    CLEARSCREEN.
}
```

New LOCK paradigm:

1. Lock global variables that are needed very often.
2. Call the first flight function (OrientForBoostbackBurn()).
3. Inside OrientForBoostbackBurn(), lock the appropriate variables and wait for completion condition.
4. When the completion condition is met, call the next function.

```
LOCK ImpactToTargetDir to DirToPoint(ImpactPos, TargetPos).

OrientForBoostbackBurn().

function OrientForBoostbackBurn {
    LOCK TargetHeading to Heading(ImpactToTargetDir, 0).

    WAIT UNTIL vAng(targetHeading:vector, ship:facing:vector) < 30 { BoostbackBurn(). }
}
```

The code above was my first attempt at the refactor. It failed because when WAIT UNTIL is called, all other execution stops. In the previous SET paradigm, I used WAIT(0.1) to control the tick speed (Which itself is flawed because the code needs time to run, so Hertz is actually <10). In the new LOCK paradigm, "WAIT UNTIL(completion condition)" simply pauses the program until the condition is met, which is never because it is never updated. Apart from this glaring flaw, this approach also doesn't allow printing variables continuously.

The solution is to add a loop (eg. 10Hz) to the end of the flight functions with this line: "WHEN (completion condition) { RunNextFlightFunction(). }". The loop can also be used to print variables continuously or kOS GUI widgets can be used (better and proper). This will also make the code far more readable. I would've done this but I was on week 3 and wanted to finish the project, maybe will in the future when bored.

No Unifed Solution To Cancel Horizontal Velocity and Minimize Landing Error

In the second part of the video at the top of this post, you can see the booster landing with the UI active. Unlike the first cinematic landing, this one barely makes it onto the landing pad.

The approach I implemented to have a soft touchdown has two phases. First, the Suicide Burn is started and it targets a point ~30 meters away from the landing pad in the opposite direction of the rocket. Second, when the rocket is <40 m/s or <25m altitude, the SoftTouchdown() function is called. This cancels out horizontal velocity and slowly decreases vertical velocity until touchdown (lerp between 10 m/s to 2 m/s, t=altitude/50). While the suicide burn is performed, the horizontal velocity changes and by extension the landing location. This is why aiming ~30 meters off is necessary in the beginning (shitty solution).

```
// Extra code not included, this gets the point across
function StartSuicideBurn {
    local magnitude to -(GetHorizationVelocity():mag^1.67) / 45. // Offset by multiple c
    SET TargetPos to AddMetersToGeoPos(targetSite, GetOffsetPosFromTargetSite(magnitude)
}

function SoftTouchdown {
    local t to TrueAltitude / 50.
    SET TargetVerticalVelocity to Lerp(-2, -10, CLAMP(t, 0, 1)).

    local aproxTimeRemaining to (TrueAltitude - TargetPosAltitude) / (SHIP:velocity:surf
    SET aproxTimeRemaining to CLAMP(aproxTimeRemaining, 5, 10). // Clamp to 10 seconds,

    local pitchMultiplier to Lerp(0, pitchLimit, CLAMP(GetHorizationVelocity():MAG/3, 0,
    LOCK STEERING TO HEADING(RetrogradeBearing, 90 - pitchMultiplier, 0).

    local baseThrottle to SHIP:Mass/(SHIP:MAXTHRUST / 9.964016384)-0.02. // Hover, Kn tc

    local vertVelError to TargetVerticalVelocity - GetVerticalVelocity().
    local throttleChange to CLAMP(vertVelError^1.7/50, 0.01, 0.25) * (vertVelError/ABS(v

    LOCK throttle to CLAMP(baseThrottle + throttleChange, 0, 1).
}
```

This approach has a poor success rate. With our two data points in the video, only 50% make it to the inner circle of the landing pad. A unified solution that both cancels horizontal velocity and minimizes landing error is needed.

There is a shitty solution here that many people have used. You can do an entry burn to cancel horizontal velocity far above the landing site, then land. This approach is shitty because it's unrealistic, uses extra fuel, and is avoiding a really fun problem.

I imagine the solution is to track the estimated net displacement in landing position during the Suicide Burn. With the estimated time to touchdown, current pitch, and current horizontal velocity you could approximate the net displacement. Add this to the target landing location and it should be a much more accurate landing. However, even this is a slightly shitty solution, maybe Rafael (Best kOS landing script I've ever seen) knows the right way.

The Fundamental Insight

The fundamental insight I learned in the past month of doing this project and watching Deep Learning lectures is that to build things properly you need sufficient knowledge of the underlying fields. The most important insights are often the most obvious ones, truly understanding and applying them is the important step. This project could have taken a few days if I was good at rotation/vector math and knew more about GNC. Skill acquisition is the most important goal all young people must have. After you've acquired the skills, building becomes exponentially easier.

"I may not be able to do it the good way, but I sure can do it the shitty way." Don't be a lazy fuck, you won't be a good programmer (or actually good at anything) by doing things the shitty way.

If I had good skills, I would've written the code like this (Video). I have a ton of respect for the man that wrote that code. He didn't do it the shitty way.

Subscribe

CKalitin
x.com/CKalitin

Geohot made a blog too. You should be working on hardware

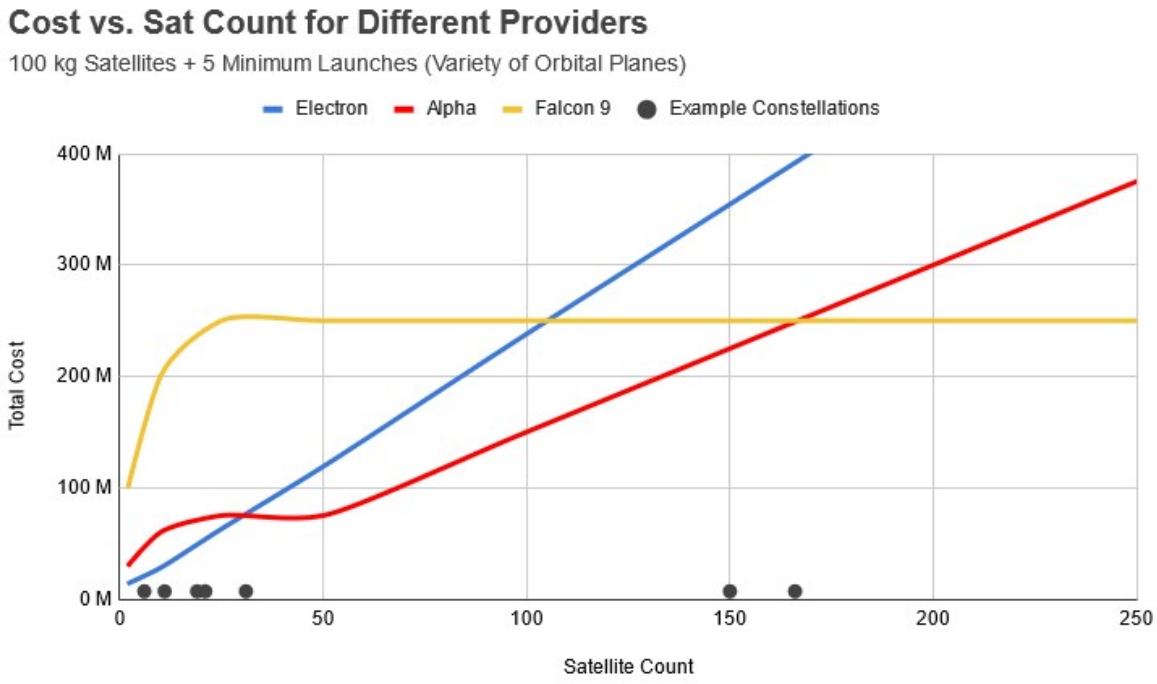
Visualizing Small Sat Constellation Tradeoffs with Charts

Jul 16, 2024 • Christopher Kalitin

Spreadsheet Link

Tell me where I'm wrong or just give compliments here.

What Data Do We Need?



In my previous blog post I gained an intuitive understanding of the tradeoffs for the different methods of launching small sat constellations. This blog post is an exercise in quantifying and visualizing that understanding.

Yesterday I felt the need to create a visual representation of the tradeoff between small sat constellation operators choosing either Electron, Alpha, or Transporter. The result was the chart

you see above. It clearly shows the size of constellation where Electron, Alpha, or Rideshare are the best options. However, this doesn't take into account variables other than the raw cost, such as the suboptimal orbits of rideshare missions, true Transporter mission costs, or the low annual mass to orbit capability of Electron because of its "low" cadence coupled with it's low payload capacity (Isn't it great we get to live in a world in which 20 launches per year is "low" cadence, get on SpaceX's level guys!).

The solution to getting a more accurate picture of the tradeoffs is to abandon the true cost of launch and replace it with Adjusted Cost, a score that takes into account the issues laid out above. This is a simple concept, adjust the cost with a multiplier that uses the number of satellites as the input variable.

Orbit Detriment Multiplier (Rideshare): $y = 85 * 2.71^{(-0.08*\text{sat count})}$
Cadence Detriment Multiplier (Dedicated): $y = 0.0005(\text{sat count} - 20)^{1.5}$

Orbit Detriment starts very high at 0 satellites and approaches zero as we near 100 satellites. This represents the fact that rideshare missions can't be used to launch small constellations that require specific orbits like NASA Tropics, Capella, or BlackSky. It shifts the calculus towards dedicated launches for constellations with a low number of satellites.

Cadence Detriment begins at 20 and scales exponentially with the number of satellites, this takes into account the fact that Electron can't launch 100 times on a whim. Furthermore, there is a minimum orbital plane input which sets the floor for the number of launches, this is required to get accurate data between Electron and Firefly's Alpha at low satellite counts.

The result is several charts that are available here.

Insights

1. Most Constellations Below 100 Satellites Are Either 100kg or 50kg

Data ends Dec 31 2023 (mostly)

Constellation / Company	Category	Sat Count	Launches	Sat Mass	Launch Vehicle	Orbits	Rideshare Addressable	Unique Plane Required	Planned Sats	Notes
Planet Labs	Observation	542	29	5.0 kg	Soyuz, Antares, I	LEO / Rideshare SSO	Yes (Scale)	No (Scale)	543+	
Spire Global	Communications	166	34	4.0 kg	PSLV, Atlas V, A	LEO	Yes (Scale)	No (Scale)	166+	Many uses: Ship tracking, m
Swarm	Communications	150	13	0.3 kg	Vega, Electron, F	Unique Plane / LEO / S	Yes (Scale)	No (Scale)	320*	Acquired by SpaceX
ICEYE	SAR	31	12	85.0 kg	Soyuz, Falcon 9	Rideshare SSO	Yes	No (Intermittent Data)	31+	Entirely F9
HawkEye 360	RF Situational Awar	21	7	30.0 kg	Falcon 9, Electro	Rideshare SSO / Band	Yes	No (Intermittent Data)	~33	Lately only F9
Black Sky	Observation	19	12	56.0 kg	PSLV, Falcon 9, I	LEO	No (Lack of Scale)	Yes (Lack of Scale)	60*	Lately only Electron, old sour
Capella	SAR	11	11	112.0 kg	Electron, Falcon	LEO / Rideshare SSO	No (Lack of Scale)	Yes (Lack of Scale)	30	
Fleet Space Centauri	Communications	6	6	35.0 kg	Falcon 9, PSLV, I	Rideshare SSO	Yes (Scale)	No (Scale)	140	Mainly Tech Demos so far
Kineis	Communications	5	1	30.0 kg	Electron	High SSO	No (Lack of Scale)	Yes (Lack of Scale)	25	
Tropics	Observation	4	2	5.3 kg	Electron	Unique Plane	No	Yes	N/A	
iQPS	SAR	4	4	100.0 kg	Epsilon, Electron	Rideshare SSO	No (Lack of Scale)	Yes (Lack of Scale)	36	Mainly Tech Demos so far
Synspective Strix	SAR	3	3	100.0 kg	Electron	Unique Plane SSO	No (Lack of Scale)	Yes (Lack of Scale)	25	
Prefire	Observation	2	2	15.0 kg	Electron	Unique Plane	No	Yes	N/A	

Expanded Chart

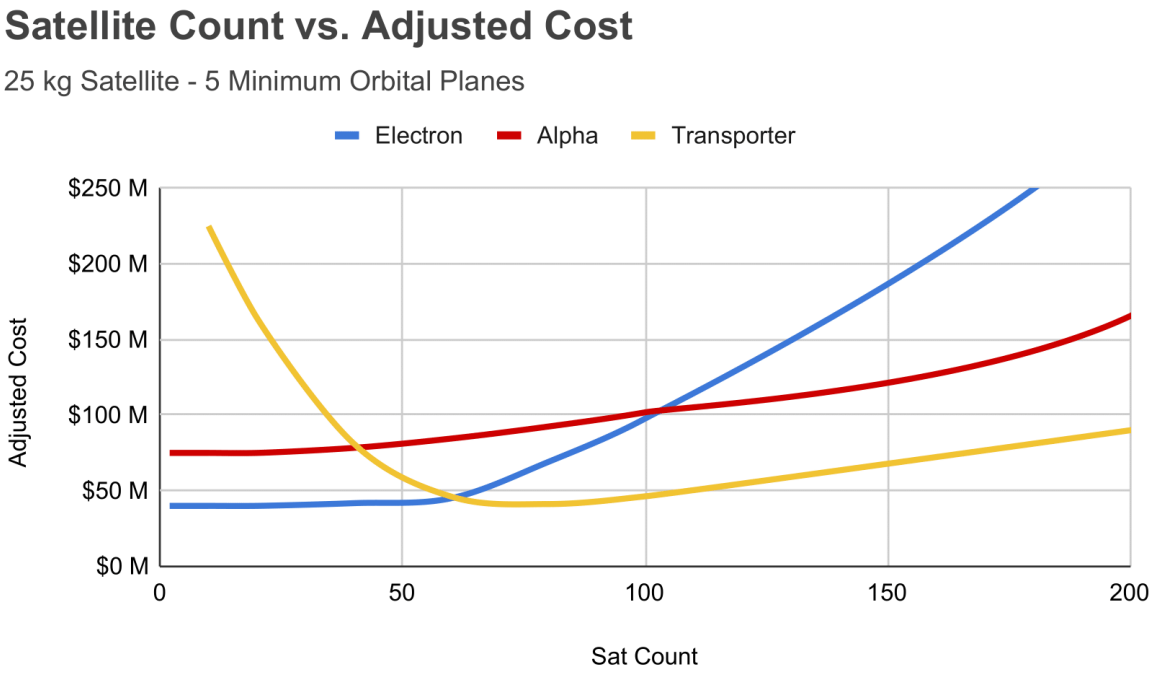
This is essential data to understand the implications of the data below.

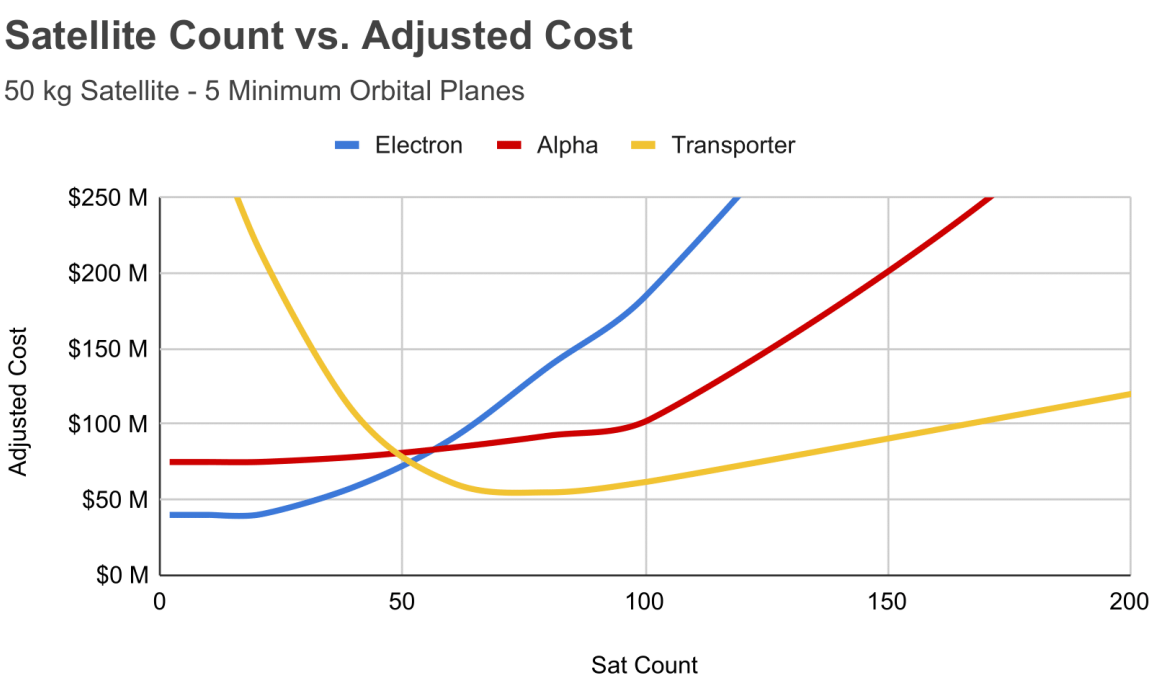
There are three primary categories of small satellite constellations.

1. Constellations with >100 satellites, most satellites under 5kg (eg. PlanetLabs)
2. 10-50 satellite constellations, mass between 30-112kg (eg. Capella)
3. <10 satellite constellation, <15kg mass (eg. Tropics)

The most interesting data I've gathered from this exercise applies to the second category. These are constellations that fit into the category of either 25 or 50 satellites total with masses of either 50kg or 100kg. This is the biggest category of the market available to small sat launch providers.

2. The Tradeoff Between Electron and Rideshare Occurs at ~50 Satellites

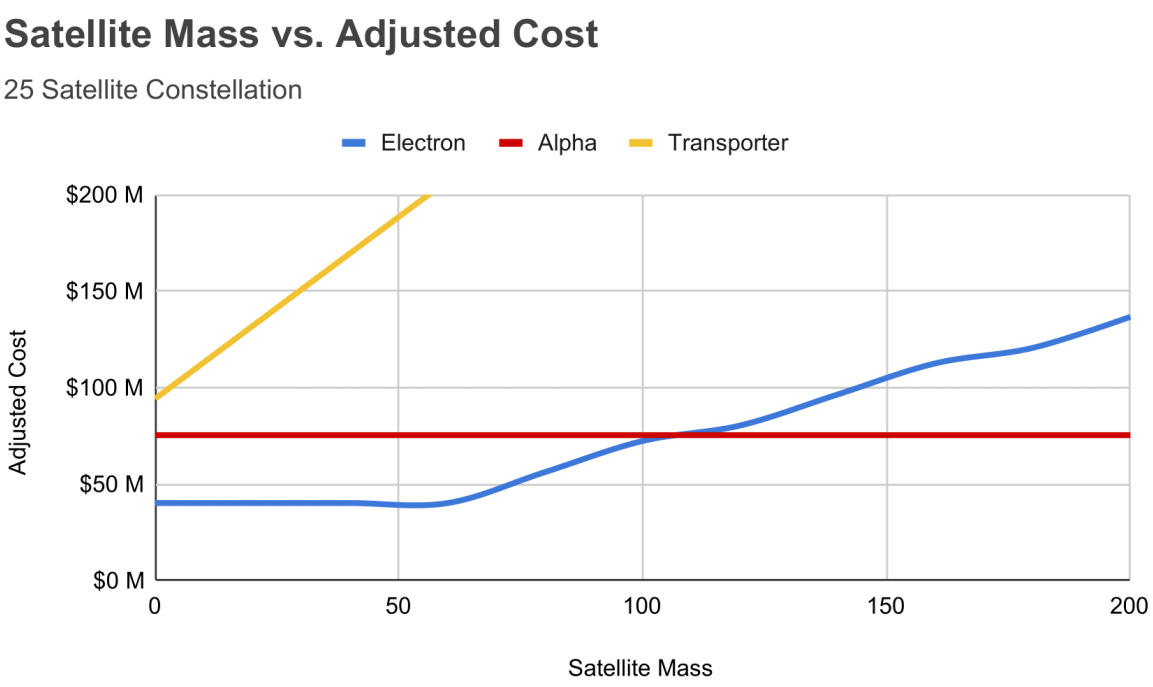




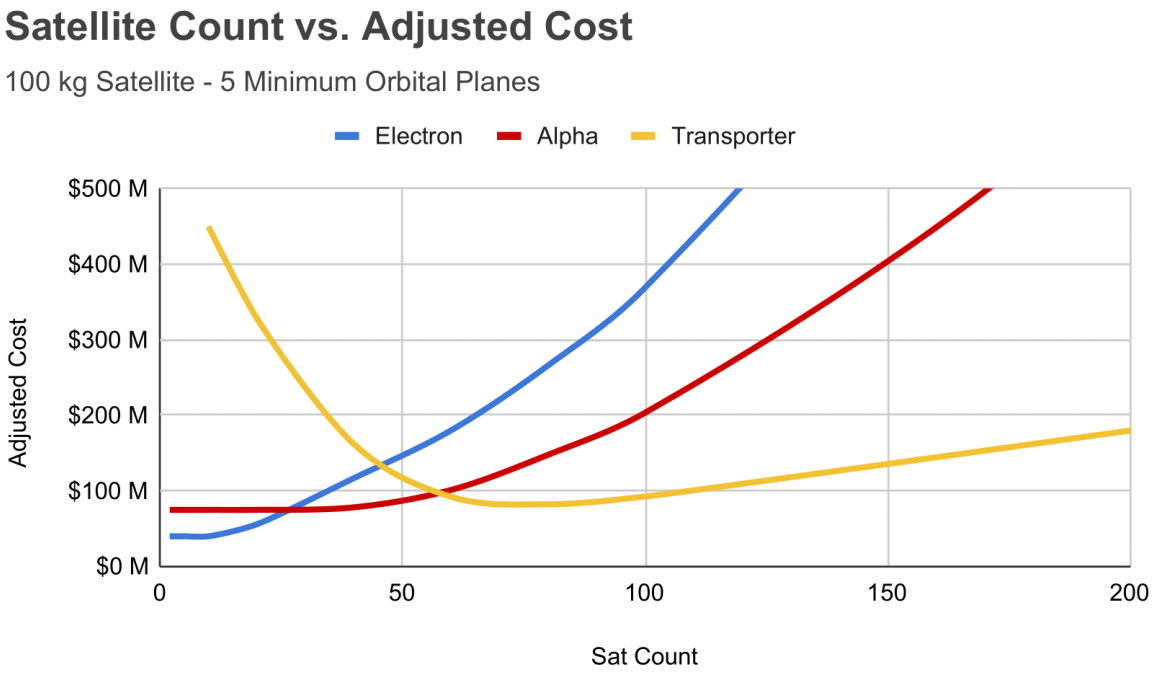
For constellations that don't require the higher payload capacity of a 1-ton class launch vehicle like Firefly's Alpha or ABL's RS1, the options available are either Electron or Falcon 9 Transporter missions. These <50kg satellites are the most common size of constellations and the tradeoff between Electron and Rideshare occurs at ~50 satellites. There are no constellations at this 50 satellite size in my dataset because this is one of the gaps in the market. Go bigger and you end up with Planet Labs, go smaller and you end up with BlackSky at ~20 satellites. This conclusion is present in both the data of real constellations above and in the charts I've created.

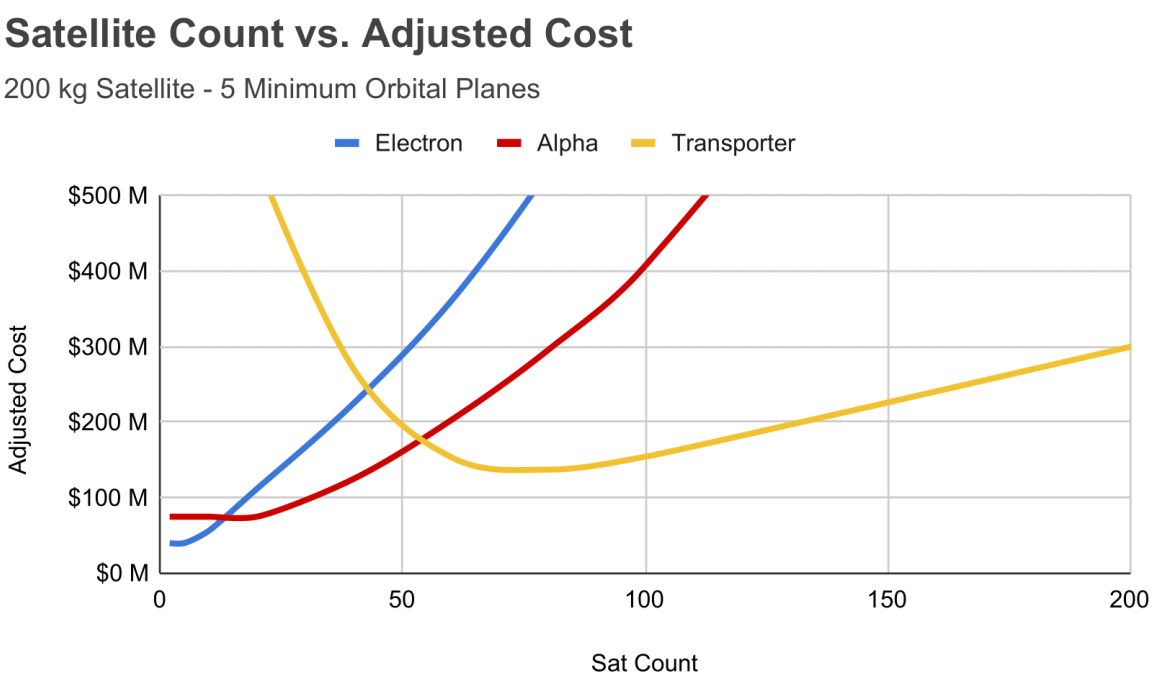
It's remarkable there's enough satellite constellations that I can make statements like the one above. I was born at the perfect time to bask in the glory of the growth of commercial spaceflight.

3. At The Most Common Constellation Size, Alpha Is Optimal for 100kg+ satellites



For most constellations (~25 satellites), Alpha is unable to properly compete with Electron because the satellites are not heavy enough to take advantage of the 1-ton payload capacity of Alpha. The higher payload capacity only starts to kick in with satellites that are over 100kg. For even heavier satellites (eg. 200kg+), Alpha provides a cheaper path to orbit than Electron, we may see some constellations pop up in this category in the future if Firefly or ABL demonstrate reliability. This is only true if the higher satellite mass prodives a significant enough advantage. For example, Earth Observation satellites seem to level out at around 100kg.





For larger constellations - eg. 50 satellites - the tradeoff shifts to lower mass satellites. This benefit extends until rideshare takes over at very large constellations, Eg. Swarm or Spire Global. The reason for this is that with a low number of orbital planes (I used 5 as a default value, we can debate this) a single 1-ton launch is more efficient than several 300kg Electron launches. Firefly's Alpha has ~3x the payload capacity of Electron for ~2x the price. In short, larger constellations benefit Firefly (until they don't) and more orbital planes benefit Rocket Lab.

Subscribe

CKalitin
x.com/CKalitin

Geohot made a blog too. You should be working on hardware

Small Sat Constellations: The line between Electron and Rideshare

Jul 4, 2024 • Christopher Kalitin

Data ends Dec 31 2023 (mostly)

Constellation / Company	Category	Sat Count	Launches	Sat Mass	Launch Vehicle	Orbits	Rideshare Addressable	Unique Plane Required	Planned Sats	Notes
Planet Labs	Observation	542	29	5.0 kg	Soyuz, Antares, I	LEO / Rideshare SSO	Yes (Scale)	No (Scale)	543+	
Spire Global	Communications	166	34	4.0 kg	PSLV, Atlas V, A	LEO	Yes (Scale)	No (Scale)	166+	Many uses: Ship tracking, m
Swarm	Communications	150	13	0.3 kg	Vega, Electron, F	Unique Plane / LEO / S	Yes (Scale)	No (Scale)	320*	Acquired by SpaceX
ICEYE	SAR	31	12	85.0 kg	Soyuz, Falcon 9	Rideshare SSO	Yes	No (Intermittent Data)	31+	Entirely F9
HawkEye 360	RF Situational Awar	21	7	30.0 kg	Falcon 9, Electro	Rideshare SSO / Bands	Yes	No (Intermittent Data)	~33	Lately only F9
Black Sky	Observation	19	12	56.0 kg	PSLV, Falcon 9, I	LEO	No (Lack of Scale)	Yes (Lack of Scale)	60*	Lately only Electron, old sour
Capella	SAR	11	11	112.0 kg	Electron, Falcon	LEO / Rideshare SSO	No (Lack of Scale)	Yes (Lack of Scale)	30	
Fleet Space Centauri	Communications	6	6	35.0 kg	Falcon 9, PSLV, I	Rideshare SSO	Yes (Scale)	No (Scale)	140	Mainly Tech Demos so far
Kineis	Communications	5	1	30.0 kg	Electron	High SSO	No (Lack of Scale)	Yes (Lack of Scale)	25	
Tropics	Observation	4	2	5.3 kg	Electron	Unique Plane	No	Yes	N/A	
iQPS	SAR	4	4	100.0 kg	Epsilon, Electron	Rideshare SSO	No (Lack of Scale)	Yes (Lack of Scale)	36	Mainly Tech Demos so far
Synspective Strix	SAR	3	3	100.0 kg	Electron	Unique Plane SSO	No (Lack of Scale)	Yes (Lack of Scale)	25	
Prefire	Observation	2	2	15.0 kg	Electron	Unique Plane	No	Yes	N/A	

Expanded Chart - Spreadsheet

If you have any feedback or criticism, please reply here.

Which Constellations Require Dedicated Launches?

Constellations that have a relatively small number of satellites and require high & regular revisit rates are well-suited for dedicated launches. Many of the recent Synthetic Aperture Radar constellations fall into this category such as Capella, iQPS, Synspective, and Black Sky (Earth Obv). These constellations benefit from high & regular revisit rates so they can provide near real-time data to their customers and cover specific parts of the Earth regularly (eg. once every half hour).

Recently there's been a great example of the Commercial benefit that dedicated launches provide: Synspective signed a 10-launch deal with Rocket Lab. Given an Electron average sales price of $8M, this is a $80M contract. The cost of a SpaceX rideshare mission is an initial charge of $300k for 50kg to SSO with each additional kilogram at $6k. Assuming Synspective launches 2 100kg Strix satellites per Electron this is 20 satellites in total. On SpaceX rideshare missions this would cost $12M, a $68M cost delta.

Launch schedule is another factor to consider when analysing Synspective's choice to go with Rocket Lab. The Electron launches will occur between 2025 and 2027. SpaceX rideshare missions are rumoured to be booked 2 years out and they launch 3-5 times per year (Up to 5 with

Bandwagon rideshare missions). Assuming a start date in 2026, Completing 10 launches for Synspective could take SpaceX until 2028 or 2029. However, this time can be shortened if the satellites have sufficient on-board propulsion to change their orbital planes (This is covered in the final section). Even assuming the schedule issue is resolved with a higher scale in the future, rideshare missions still mean suboptimal orbits.

The cost advantage of rideshare missions is cancelled out by the schedule (temporary) and orbit/delivery (permanent) disadvantages. One of the primary disadvantages of rideshare missions is having to negotiate with several other companies on which orbit the satellites will be deployed into. This means every customer is not where they want to be and on top of the schedule disadvantage, this means each satellite is initially producing less revenue than it otherwise could be. In the case of Synspective, the advantage of launching on Electron appears to be worth at least $3.4M.

Prefire and Tropics do not fall into the category laid out above because they are made of 2 & 4 satellites respectively and will not be expanded in the future. Unlike commercial constellations, these are one-off NASA constellations that serve a very narrow and specific Earth Observation purpose. So, there will not be an increase in demand for the data these satellites provide. When new data is needed, NASA will develop and launch a new set of 2-10 satellites.

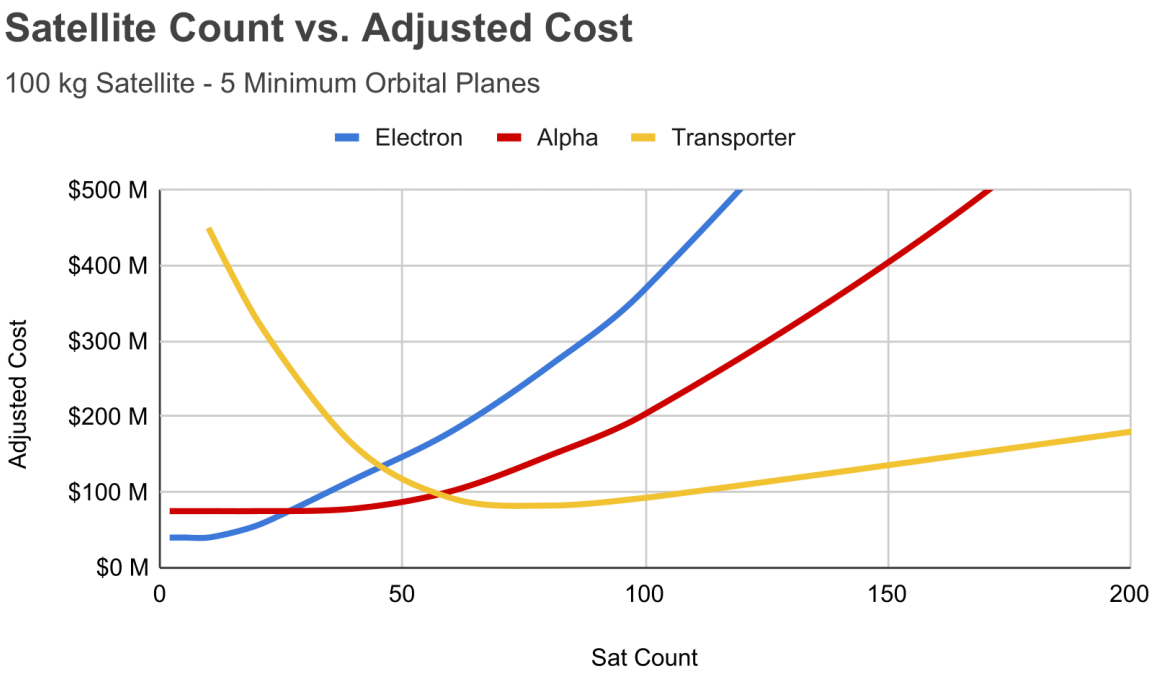
Rideshare Missions Launch Most Constellations Satellites

The first category of Small Satellite Constellations that launch on rideshare missions are those with such high scale that the exact orbits of dedicated launches are not required. These include Planet Labs, Spire Global, and Swarm (before SpaceX acquired them). Planet Labs has 542 satellites, Spire has 166, and Swarm had 150 before acquisition. Given the number of launches required to deploy these constellations, you get a large enough distribution of orbital planes that you can cover all of the Earth with a high revisit rate. Given the falling price of launch, creating satellites cheap enough to produce hundreds so you can utilize rideshare missions could be a winning strategy (Like Planet Labs).

The second category is constellations that don't require the revisit rate benefit of dedicated launches (even spacing of orbital planes). HawkEye 360 is the best example of this as this is one of the few constellations that is not focused on Earth Observation or communication with Earth-based assets. They sell Radio Frequency signal location data. This process involves detecting RF emissions and triangulating the source with multiple satellite passes over the same location. This means the orbit requirements are not as strict as other constellations. In contrast to the Synspective example in the previous section, HawkEye 360 does not get as much direct monetary benefit from dedicated launches. They recently cancelled a few of their upcoming Electron launches.

One of the great benefits of rideshare missions (even to Electron) has been the ability to launch test satellites for a low cost. Many early Capella and BlackSky satellites launched on Falcon 9 and this allowed for cheaper development of their constellations which ended up launching on Electron. This allows for cheaper launches during the early scaling of the constellation and an accelerated completion timeline.

The Market for 1-Ton Class Launch Vehicles



Just like Falcon 9, Electron owns the small sat market because there are no capable competitors. Over the next few years, we'll see Firefly ramp up its Alpha launches and ABL's RS1 come online (hopefully). From a product perspective, these 1-ton class rockets have the potential to take market share in the small-sat constellation launch market. Ozan Bellik pointed this out when I mentioned my blog post Comparing Demand for Firefly's Alpha vs. Electron. My conclusion wasn't completely incorrect, but inaccurate enough to warrant this section.

The niche of the market that these vehicles can solve is constellations with ~100kg satellites that plan to launch more than 20-30 satellites. These include iQPS, Synspective, Capella, and BlackSky. Alpha and RS1 cost around $15M which is ~2x more than Electron while having ~4x the payload capacity. On the surface, this appears to be a 2x improvement in $/kg, but the entire capacity of the rocket may not be used. Using the entire 1000kg of payload capacity would mean launching ~10 satellites at a time which negates the advantage of having satellites spaced out in different orbital planes which lowers revisit rates which in turn makes a constellations offering less competitive with other Earth Observation companies, ie. lower revenue. The breakeven point for a 1-ton class rocket vs. Electron occurs when 4-6 100kg satellites are launched on a single mission,