EPISODE 878

[INTRODUCTION]

[00:00:00] JM: Envoy is an open source edge and service proxy that was originally developed at Lyft. Envoy is often deployed as a sidecar application that runs alongside a service and helps that service by providing features such as routing, rate limiting, telemetry and security policy. Envoy has gained significant traction in the open source community and has formed the backbone of popular service mesh projects such as Istio. Envoy has been mostly used as a backend technology, but the potential applications of Envoy include frontend client applications as well. The goal of Envoy is to make the network easier to work with and the network includes client applications such as mobile apps running on a phone.

Envoy Mobile is a network proxy for mobile applications. Envoy Mobile brings many of the benefits of envoy to the mobile client ecosystem. It provides mobile developers with a library that can simplify or abstract away many of the modern advances that have been made in networking in recent years, such as HTTP/2, GRPC and Quick.

Matt Kline is the creator of Envoy and he returns to the show to discuss Envoy Mobile. Matt describes how the networking challenges of mobile applications are similar to those of backend systems and cloud infrastructure. We discussed in advances in networking technology that Envoy Mobile helps bring to the mobile ecosystem and also touch on the scalability challenges that Matt is seeing at Lyft.

Matt has been on the show a couple times previously, and you can find all those episode with Matt and the episodes about Envoy, the episodes about service mesh. You can find all those on the SEDdaily apps for iOS and Android. We've refactored these apps recently and they're in great shape. They are great listening experience for Software Engineering Daily, and they include the ability to comment and search through episodes. I will be commenting on each episode going forward for the foreseeable future. So, if you're interested in discussing Software Engineering Daily, check out the apps or go to softwaredaily.com, which is the web version of our platform and share your thoughts. You can also use these apps to become a paid subscriber and get ad-free episodes.

Also, FindCollabs is the company I'm building. FindCollabs is a place to find collaborators and build projects. If you've got an open source project or a music project or really any kind of project and you're looking for collaborators. You want people to collaborate with you on your project or even start a company with. You can check out findcollabs.com.

With that, let's get on to today's show.

[SPONSOR MESSAGE]

[00:02:57] JM: There are so many good podcasts to listen to these days and it can be hard to make time to sit down and read a full-length book, and there are more good books than ever. I like business books, and self-help books, and history books, but I don't have time to get through all the books that I want to.

Blinkist gives you the best takeaways, the need to know information and the important points of thousands of nonfiction books condensed into 15 minutes that you can read or listen to. I like Blinkist, because I like audio, and Blinkist is an innovative, useful audio format. You can get a free 7-day trial and support Software Engineering Daily by going to blinkist.com/sedaily and signing up.

On Blinkist, I've listened to a few very long books about China in their 15-minute form, and I also use Blinkist to review great books that I've read in the past, such as *Principles* by Ray Dalio, or *Being Mortal* by Atul Gawande.

Try out your free 7-day trial by going to blinkist.com/sedaily and signing up. That's blinkist.com/sedaily and get those books condensed into 15 minutes and get more throughput in your book reading activities.

Thanks to Blinkist for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[00:04:37] JM: Matt Klein, welcome back to Software Engineering Daily.

[00:04:39] MK: Thanks for having me.

[00:04:40] JM: The goal of the original Envoy project was to make the network easier to work with. Give a reminder of the networking problems that Envoy addresses.

[00:04:53] MK: Yeah. So, what we've seen as most folks have moved to microservice architectures is that they typically run into a bunch of comment scaling problems, and these are typically around networking and observability, and these are things like not being able to understand where failures are occurring or not having consistent networking features around things like service discovery, load balancing, timeouts, retries, things like that.

At the same time, what we've seen in the last 5 or 10 years is that folks have moved away from situations in which they typically use a single language. So, at the times of enterprise JAVA deployment are typically over, or a single JAVA enterprise deployment. What we see these days is we typically see what I call a polyglot language environment, and that means that either via acquisition, or companies that are allowing developers to develop in multiple languages. We see quite a few different languages that companies are using in production.

What that ends up leading to is that in order to solve some of these common concerns, again, around service discovery, load balancing, observability, etc., companies essentially have two different choices. They can implement libraries in each language and those libraries can actually go through and they can implement all those common concerns. But then, of course, they have to do that in each language, or we've seen a pattern that's become popular over the last few years, which is the sidecar proxy pattern. That sidecar proxy can implement a bunch of that functionality, and then that can be done in a single language. Then if a company is using 5 or 7 or 10 different languages, those frameworks can rely on the sidecar proxy to actually implement that, that functionality.

So, Envoy was built originally as a sidecar proxy to do these types of things. Then over the past few years as it become a lot more popular, we've seen Envoy being used in a variety of different situations. We see it being used for edge load balancing. We see it being used in a sidecar or a

client-side load balancing scenario. Then we also see it being used as an internal or a middle proxy scenario.

The reason that we've seen it being used in all of these cases is that having a single piece of technology that an organization knows how to operate, it's very nice to be able to use it in all of these cases. So the organization only has to learn how to use it once. So, we are solving a bunch of common concerns and we're solving them both for the sidecar proxy scenario, but also for the edge scenario or the middle proxy scenario.

[00:07:40] JM: These different scenarios, the edge proxy, the backend sidecar proxy. These are not mobile application proxies.

[00:07:52] MK: Correct.

[00:07:52] JM: The original envoy was mostly used for server-side application, and mobile applications are more of a client-side situation. How do the networking challenges of mobile differ from the networking challenges that are experienced on the server-side?

[00:08:08] MK: What's super interesting is that they don't actually differ that much. I mean, of course, there are lossy networks, and much more frequently we wind up in environments in which the mobile clients have to deal with either spotty networks, or slow networks. But in terms of what the mobile clients have to do, it ends up being pretty similar. They have to do some type of service discovery to find backends, whether that typically uses something like DNS. They have to make RPCs. They have to have retry policies and timeous and those types of things.

I think the other thing that's happened over the past couple of years in particular as many applications have moved to mobile first applications, is that we have realized from an industry perspective that all of the effort that we're putting into making the server-side system more reliable is obviously fantastic. But if at the edge proxy we have a, let's say, 99.99% success rate or four 9s, that's obviously great. But if the user of the mobile application is only seeing a 75% success rate, it doesn't matter how amazing our infrastructure is on that server-side.

So, we end up needing a lot of the same observability, a lot of the same things that we need on that server-side. We need it on the client side so that we can make sure that we are delivering that fantastic experience to our users.

[00:09:44] JM: There are many things that can go wrong on a mobile client device. You could have a poor cellular network. You could have a problem on the operating system. How do we differentiate between problems in the network versus problems that are just happening as a result of – Well, I guess, when I say network, I should say the programmatic network, rather than the cellular network, or just something that's going wrong in the operating system, or something that's beyond your control.

[00:10:16] MK: Sure. When you actually sit down and think about it, it's not really any different than what we do on the server-side in the sense that if I have a service that talks to another service and I'm running within some type of virtual infrastructure, how do I know if it's my application. How do I know if it's the virtual machine? How do I know if it's the physical network?

So, it's really the same set of problems, and I guess the quick answer, which is not very satisfying is that sometimes it can be very difficult to tell. All we can do from a network operation standpoint is we can try to build in the right metrics, the right analytics, the right logging so that we can view the network at different layers and we can try to correlate failure. That might mean trying to look for patterns.

So, are we seeing, for example, all of our users who are on the T-mobile network? Are they seeing a particular problem that might be indicative of a network failure? Are we only seeing issues with users who are on that particular version of iOS that would indicate that maybe we have some bug that's related to that version of that operating system.

So, what it comes down to is that there's very rarely a smoking gun, and I think what's much more important is to have the right infrastructure in place so that, again, we can gather the right stats, the right logging, the right tracing, the right analytics and then we can start debugging and we can start trying to understand what are the patterns that we're seeing.

So, when we start to talk about why we would want to run something like Envoy on the mobile client, a lot of the reason that we want to do that is that it gives us a base platform with which we can actually emit a lot of these metrics, a lot of these analytics and a lot of these logging so that we can do the right debugging to actually start to understand where the problems are happening.

[00:12:21] JM: So, there are tools for observability that are on the mobile device. There are things like crash reporting. Why is it advantageous to create a module like Envoy mobile where all of these things would be centralized into one system?

[00:12:41] MK: Sure. The main rationale is really for the same reason that we've seen people start to really utilize the sidecar proxy paradigm, which is that if we have a polyglot environment where we're using different frameworks, different programming languages, potentially different operating systems, what we can do as system developers and system operators is what I call reducing cognitive load. By that, I mean that we are developing these super complicated distributed systems. The only way to really understand this effective chaos is to try to limit the number of variables. So, we would like to avoid re-implementing code in multiple places. We would like to have single implementations of things like retry policies.

So, for the same reason that it's potentially advantageous to use something like Envoy as a sidecar or a system proxy, because we get a bunch of these functionality in one place. Instead of having to go and implement a bunch of these logic – Again, that could be for networking primitives, like retries and timeouts or it could be for stats logging and tracing. Instead of having to implement that on both iOS and Android and support every operating system version, we can implement this in one place and we can get the same output in every place and that reduces cognitive load and makes it a lot easier to understand what's actually happening.

What's interesting about this is that there is actually quite a lot of prior president for this. Google has a library called Pronet, which they have shipped for quite some time. It's a section of the code that shifts with the Chrome browser. They use that code in all of their mobile applications. So that's common C++ code that is used both on Android as well as iOS. Facebook also has a long history of doing something similar. So, this is something that's been done for quite some time, but we think that with Envoy Mobile, we can take this paradigm and we can do even more.

[00:14:51] JM: The backend model for deploying Envoy is in this sidecar container. In mobile environments, I guess we don't use containers, or they're at least not as much of a present deployment system as they are in the backend. So, you're just bundling it as part of the binary. Is that right? If an application wants to use Envoy mobile, they just include it as a library?

[00:15:20] MK: Yeah, right. So, it's going back on some of the things that I've typically spoken about, which is that the library-based solution is not very productive and you have to deal with upgrades, etc., etc., etc.

But the current realities I would say of how people ship mobile application are that the code has to be bundled. So, our initial target for Envoy Mobile is to actually bundle the code as a library component and then actually ship that out.

I do think that in the medium long-term, if we are successful with this project, I could see the sidecar pattern moving over to mobile. There's people that I've talked to over at Google that have expressed interest in this. I would imagine that Apple might be interested also. But I think that we'll have to see how things evolve. But in the short and medium term, I think we are looking at a bundled library.

[00:16:16] JM: Compare the deployment model for using Envoy on the backend with Envoy Mobile. I understand it's a binary, and so that's a little bit different. Can you just describe if let's say I'm an application developer, maybe I've got like a food delivery app, and the food delivery app has existed for six years. It's on the app store. Now I want to add Envoy Mobile to it. What's that experience like?

[00:16:45] MK: So, to be super clear, it is very, very, very early days for the project. We made an explicit decision to open source early. So, we prove that we can get it running on both platforms. We have open sourced our roadmap, but it's still I would say very early days.

So, in terms of what will happen for the user or what the API will actually look like, I think that's still being sorted out. But I think the idea is that we are going to offer a networking library, and the goal of this library is to not expose any of the C++ code up to the end user. Our goal is to

build a future-looking network API. By future-looking, I mean using the more modern mobile programming languages. So that will be Swift on iOS and Kotlin on Android.

Our plans are to surface a mobile networking library that offers various pieces of functionality. So that's going to be making API calls. Obviously, dealing with different types of APIs, whether that'd be REST space or proto-based. Then, eventually, we expect to add a bunch of higher layer functionality.

So, one of the things that I'm most excited about is that as we move towards proto or structured APIs, being able to offer higher layer of functionality. So things like streaming API caching, or offline or differed APIs, things like that. I think that will really move the state-of-the-art of mobile networking forward.

So in terms of how people will consume this in the future, is that we expect to offer an API. We may make it look like existing APIs. For example, we would look at, okay, HTTP as something that we might want to emulate. But I think that we are looking at this from a blank slate perspective and just trying to understand what would a modern mobile networking API look like.

Then as we add more functionality, again, we expect to add ergonomic APIs that will allow people to admit stats, logging and tracing, enable them to opt-in to some of these more advanced features, and that would apply both to JSON REST space APIs. Then for those users that are moving towards the proto or the structured APIs, we would allow them to opt-in to even further features.

[SPONSOR MESSAGE]

[00:19:18] JM: Today's show is sponsored by Datadog, a modern, full-stack monitoring platform for cloud infrastructure, applications, logs and metrics all in one place. Use Datadog's rich, customizable dashboards to monitor, correlate, visualize and alert on data from disparate devices and cloud backends to have full visibility into performance.

Datadog breaks down the silos within an organization's teams and removes blind spots that

could cause potential downtime. With more than 250 integrations, Datadog makes it easy to collaborate together and monitor every layer of their stack within a single platform.

Try monitoring with Datadog today using a free 14-day trial at softwareengineeringdaily.com/datadog, and they'll send you a free t-shirt. That's softwareengineeringdaily.com/datadog. You sign up and you get a free t-shirt. Check it out at softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[00:20:34] JM: As I was reading your article, I did have to look up a couple of the networking technologies that you touched on in the article, because one of the things you were saying is if you get this proxy in your application, you can defer some of the work of low-level networking technology to this proxy. So there have been this set of networking technologies that have improved over the last 5 to 7 years and they're not necessarily available out of the box to anyone building a mobile application. Describe some of these lower level networking technologies.

[00:21:20] MK: Yeah. So, these are things like modern TLS. So, TLS 1.3, which has both performance as well as security enhancements. Things like Quick or HTTP/3. This is the next gen networking protocol, which is aimed for mobile or lossy environments. What we typically see on mobile is that just because of the longer development times, the fact that handsets tend to be in the world for longer, we typically see a slower adaption of these types of, I would say, newer or more advanced networking protocols.

So, if a mobile application wants to depend on something like TLS 1.3 or Quick, it may be years before essentially all of the platforms across both iOS and Android and all of the OS versions that they want to support with their application, it may be years before they can rely on those technologies being available.

That's not only years of potentially slightly or security or slightly OS performance. It's years of, again, having that cognitive load. If you're having to support your application across two different major vendor ecosystems, a bunch of different operating systems. That debugging cognitive load just becomes so much larger because you have to start understanding where is

this bug report from, or how do I slice and dice based on the version of the operating system or the platform that the user is actually using?

So, being able to relegate or offload a bunch of these functionality to common code, whether that'd be a sidecar proxy or even a library, it allows for a consistency of experience, and that improves both cognitive load. It potentially improves security. It improves performance. So that's ultimately what we're going to for.

Then like I was saying before, we think that once we have this base functionality in place, just like we can do a bunch with the sidecar proxy pattern of offloading more complicated functionality or more advanced functionality. We think that we can do that on the mobile side as well.

So, if we're implementing, hashing for streaming APIs or doing a deferred or offline APIs, instead of having to implement that and test that and understand that across both Android and iOS, if we can do that in a single codebase, it allows us to focus on that codebase. Make sure that it's well-tested. Make sure it has high-performance, and that just seems like a better use of time.

[00:24:01] JM: Can you shed more light on why this stuff doesn't get bundled into the operating system? It seems like it's such a free lunch, right? If it's just networking technology that improves your performance and your reliability and your security, why is it not an OS level improvement?

[00:24:19] MK: To be fair, it is. So, Google and Apple, they obviously will work on technologies like TLS 1.3 or they will work on Quick. But, again, because of the longer cycles of how these handsets are built. If Google releases a new version of Android, then it has to go to Samsung or it has to go the different makers of phones and they have to certify the operating system and then they have to get it on to the phone.

So, there can be multiyear cycles before a particular piece of technology even gets to a shipping phone for the first time. So, those OS cycles tend to be longer. Because of the risk of breaking existing handsets, Google and Apple, they typically don't backfill major features for a long period

of time. So what you see within the industry is a handset, it might get shipped on a particular version of Android, or one of Apple's handsets might get shipped on a particular version of iOS.

Those handsets may see one or two operating system upgrades in their lifetime, right? So it's not like Apple or Android continues – Or Apple, or Google continues to support all handsets in perpetuity for OS upgrades. That's just not the way that it works.

So, it's not that they're not working on it. These things are OS concerns. It's just that it takes a longtime to get all of these technologies to every user, and in that timeframe, there are some newer technology. So, if you or if one wants their mobile application, they're always be at the forefront of that technology at the networking layer. Again, whether that's for security or performance, or features, it's nearly impossible to ensure that every user of one's application has access to that. That's why see even Google who obviously runs Android, they ship this library to all of their mobile applications, because they and Facebook both view it as a competitive advantage to make sure that they have a consistent networking experience across all of their applications regardless of the operating system version.

[00:26:40] JM: So, if you bundle the networking stack with the application, then as your application gets updated, then the networking stack gets updated and people definitely update their apps more aggressively than their operating system.

[00:26:58] MK: Exactly, right. Again, it's not to say that in a future world something like Envoy might not be bundled in a way where it's out of band from the operating system, but it sees updates, but it's a shared library or a shared proxy, if you will. I could see the industry moving in that direction, but that's not where we are today.

[00:27:23] JM: When I make a request from my mobile application these days, is that request usually in JSON? Is it in a protocol buffer? What format is it in?

[00:27:35] MK: Hard for me to answer that. It's hard for me to speak across the industry. If I had to guess, I would say that the majority of API requests that go from a mobile application to a backend service are probably JSON, YAML, REST type APIs. I do think that there is increasing

uptake of structured APIs, whether that'd be using a technology like Thrift or protocol buffers. I think protocol buffers is becoming the default structured API.

So, I think that overtime, as it becomes easier to consume protocol buffered APIs, I think that we're going to see more and more APIs be based on them. As I've written about before, I think protocol buffers versions 3, or as people say proto 3, I think it's a really fantastic technology. I cannot recommend it enough. I just think Google has put a tremendous amount of thought into it.

One of the nicest features that I find about proto 3 is that it allows APIs to be defined in a structured manner. So, everything is typed. But also allow seamless conversion between that structured representation to JSON or YAML. So, to me, it really gives you the best of both worlds, which is that you can define APIs in a structured way and you can seamlessly transcode them between JSON, YAML and structured.

I think that once more folks realize that inherent power, which is that you really get the best of all situations. You can support backwards compatible clients that only want to deal with JSON, but your internal systems can be structured. For me, there's almost no downside to it and the tooling is getting better and better and better. So, though today I would say that a majority of developers probably still use raw JSON, YAML, REST type APIs. I do think that overtime, that we will see more and more APIs move to being structured.

[00:29:38] JM: What is required of a developer that wants to use protocol buffers in their application?

[00:29:45] MK: So, what is essentially required today is that the APIs obviously have to be defined. So that happens in a structured file. Then those APIs have to be essentially compiled. So, what the protocol buffered compiler does is it compiles language-specific representations of those APIs. That gets admitted out into all of the different languages that protocol buffer supports. Then those APIs can either be used raw or a Google supports technology called GRPC, which essentially is just a wrapper on top of protocol buffer that allows for the sending and receiving of protocol buffer based APIs.

So, from an end user perspective, typically the easiest way to consume protocol buffer based APIs is probably going to be the GRPC. So that will be defining the APIs. Using the generated GRPC clients and servers and then just running wild. But, again, what we've seen historically, and this is getting better every day, is that the GRPC generated code is of differing quality depending on the language. Again, we come back to this inherent tradeoff of the library-based solution versus the sidecar-based solution.

If we are making APIs and we want to co-generate those APIs and a bunch of different languages, there's obviously no way around doing that in each and every language. But as you might imagine, some languages currently have better support than others. So, if you're writing in JAVA, obviously the support will be amazing. If you're writing in a less frequently used language, the support might not be quite as good.

So, that is evolving over time. What we've seen at Lyft, for example, is that Lyft is a very forward-thinking user of Swift. So, we actually were one of the first, I think, companies to rewrite our iOS application using Swift. Historically, for example, the protocol buffer based support for Swift has not been quite as good. So, Lyft has actually worked up on open sourcing Swift code generators.

But overtime, I think as more people want to consume protocol buffer APIs, I would think that the code generators will get better and better and better in every language. Then I think it will increasingly become easier to use.

[00:32:12] JM: Does Envoy Mobile help with the usage of protocol buffers at all? What's the connection between protocol buffers and Envoy Mobile?

[00:32:23] MK: Right. So, from the Envoy Mobile perspective, I would say that there're two different angles that we're actually coming at this from. One of the angles is from the lowest layers. So that's what we already talked about in terms of providing consistent networking across both platforms. So these are based networking technologies, like making sure that we support Quick, or we support TLS 1.3, or we support modern extensions for propagating network quality or things like that. That's obviously one angle that we are approaching it from.

The other angle that we're approaching it from is from the top down angle, and that's from the protocol buffer based angle. One of the nice things about protocol buffers is that it supports something called annotations. An annotation is a way of given a particular API, which is messages and fields, and those fields can have types.

On a field, you can provide custom tags or custom annotations, and those annotations are where we plan on supporting some of this advanced feature. So, for example, again, cashing, offline deferred API's, we imagine a scenario where if Envoy Mobile can determine the current network quality. So, for example, if Envoy Mobile can consistently determine, "We are on a fast Wi-Fi network, or we are on a fast LTE network, we might use the enhanced version of the API," or let's say that the library determines that we are on is spotty network, maybe we can start – We can dynamically shift to a degraded API that uses less bites back and forth and will probably load faster for the user.

So, once we have this ability to provide these custom annotations, we would allow the Envoy Mobile layer to actually operate on those annotations. So, there will be nothing company specific about those annotations. So company organization A can to find their API, but they can use a common set of annotations, and Company B can do the same, and so on.

Then Envoy Mobile, given the definitions of APIs, can actually operate on those annotations in an API agnostic way. So, even though the API might be talking about ridesharing for Lyft or it might be talking about music for someone else, all of these applications can get a common set of functionality.

What we think – Again, as I was saying before, this project is in very early days. I would encourage everyone to check out our repo and obviously follow along. We think that we will probably support the GRPC wire protocol so that Envoy Mobile will always be compatible with GRPC servers. But what I would imagine it may end up happening is we may have our own envoy immobile code generators. So that if you are using Swift or Kotlin, we can generate you library stubs or API stubs that will give you all these advanced functionality and you won't need to worry about the underlying mechanics.

So, I would suspect that that's how this will play out, is that, again, there'll be two angles here. We'll have the lower-layer networking angle, and then we'll have the top down API based angle. We'll be focusing on both.

[00:35:44] JM: You bring up an interesting point, the idea of API degradation, where if I'm using Lyft, for example, and if I want a great network connection, I'd be fine with my Lyft application heart beating with the backend with tons of information about where I am, and my velocity, and so on, just tons of information. But if I get to a spotty cellular connection, I definitely want the Lyft backend to know where I am, and that's maybe the core of what you would want to send. It's like, "Here's my geo every 10 seconds or something." If you're on a really bad connection, maybe you just send it every 10 seconds, like as low as that.

But the idea that you would want different volumes of information sent to the backend under different cellular connections, I would guess that most mobile applications don't factor in that option. For no other reason, then it's pretty hard to just like – What do you have? Like a switch statement? It's like if the cellular connectivity is three bars, do this. If it's four bars, do that. So putting that into a usable API sounds pretty handy.

[00:37:09] MK: Well, what we see, and we've seen this across like almost every type of application. I have personally seen this at multiple companies now. It's been widely reported at tons of other companies, and actually it's a very simple thing, which is that we almost universally see that when applications are faster, when they are snappier, when they load – When the user perceives them to load quickly, this conversion is higher. Again, we just see this across almost every application, and we can see pretty substantial differences. These are not small conversion differences. They can be many, many, many percentage points.

So, from a business perspective, and that's why I think you've seen investment in protocols like Quick and for other types of technology that are really focused on ultimately making the user's application snappier. So, when we see this, again, almost universally across every application, it's widely accepted within the industry now that focusing on making the end user's application snappier and degrade well is a very, very worthwhile thing.

So then the question becomes how does one do that? Like you were saying, it's certainly possible to have a bunch of switch statements and if statement and like specific logic in every application that's trying to determine like network degradation and those types of things. But we believe that there is a tremendous opportunity to do this within the library, and that if we do this well, we can actually be a lot more sophisticated about it and a lot more general about it than any one application could probably justify doing within only their application. We can provide the right hooks and the right tags and the right annotations so that – Again, just to go back to Lyft, you can imagine – Again, I'm not on my speaking about any particular plans here. I'm just trying to give an example. You can imagine with Lyft that, yes, as you pointed out, we have core functionality that must happen. We must provide location data occasionally. We must obviously, when we're requesting rides, we must show car locations. But maybe there're other things that are important, but they're not critical.

For example, let's say that we want to show coupons or something like that, or we want to give updates on the app, or like there might be all types of things that are important. But if we are in really poor networking environment, maybe we just don't want to show those right now, because the critical data has to come through faster.

We believe that by making the API structured and by annotating and tagging the API in certain ways and then having the network later be smart about what can be dropped both by the server and the client, we think that in these poor networking environments, we can greatly reduce the bytes sent back and forth. Like I was saying before, ultimately, at the end of the day, what it comes down to is you just need to send less bytes back and forth. The fewer bytes, in general, the faster things are going to load, and the faster things are going to load with the right functionality, the better business conversion is going to be.

If your users encounter a bug that they want to report themselves, they can just shake their phone and send feedback to you and your development team. This level of communication is great for beta testers, or if you have a live app where you're in close contact with your users, you will also receive automated crash reports, which means that every crash on a user device is going to be reported to you and your development team.

Go to instabug.com/sed and try Instabug for free for 14 days. If you like it, you can use code SED19 for 20% off of any of the Instabug plans. Instabug lets you get feedback from your users inside your app. When your app has an issue or a bug or a UI glitch, your users should be able to report it, and they should be able to have a dialogue with you.

Go to instabug.com/sed to try Instabug for free for 14 days. Just thinking about it makes me want to get Instabug for the SEDaily apps. We should probably check it out in more detail, because companies like Lyft, and PayPal, and Samsung all use Instabug and they want to improve the quality of their apps. Well, I want to improve the quality of my apps too. If you want to improve the quality of your apps, you can check out instabug.com/sed.

[INTERVIEW CONTINUED]

[00:42:41] JM: We've done some shows about the service mesh idea, which is that as an expansion on the model that we've been talking about where you have a proxy and you have a proxy on every service and the proxies can talk to each other, and because you have a standard proxy for communication between different services, you have a more reliable network and you have some homogeneity. Then the next phase on top of that is the idea that you can do control across all of these different proxies, and so you have a unified control plane that interacts with all these different proxies. Did Envoy Mobile change your vision for what the ideal service mesh would look like?

[00:43:30] MK: Yeah. So this is certainly not our short-term goal. It's probably not even our medium-term goal. But when I look out long-term, and there's obviously a big if here, if the project find success, if we can get wide deployment. I think there is a huge amount of long-term potential here in terms of how people actually manage how they configure networking on their mobile applications.

Some of your listeners will probably chuckle at this, but I think there is the potential to “expand” the service mesh out to the mobile client. I'm sure that will scare people greatly, right? But by that, really, all I mean is that one of the reasons that Envoy –I think Envoy’s become very popular for a couple different reasons, but one of the reasons that Envoy has become very popular is because of its configuration API.

These are APIs that collectively we called XDS. These are our discovery service APIs. At least on the server-side, we’ve seen now in entire ecosystem of companies, whether they’d be cloud providers, or startups, or individual companies who are building these management systems that manage fleets of Envoys.

What you'll see is that if we assume for a second that Envoy runs on a bunch of mobile applications or even just the mobile applications for a company that’s already using Envoy. If we can use these real-time configuration APIs to do things like traffic shifting, or potentially authentication, or on-the-fly actually change retry policies, or maybe we want to shift just a single route to a particular data center, or we’d like to test something, or something along those lines. There is the ability to really change how mobile networking is configured, how the policy is actually sent out. These are not really science-fiction things.

So, for example, to give a concrete example, the state of the art today in terms of how major mobile applications tend to do traffic shifting, load-balancing, load shedding, is they typically use DNS and they typically use BGP. People have done some incredible things, right? They use DNS to find the closest location, and then the use BGP to make sure that we’re routing to the right place, and pretty amazing sets of technology.

But in some ways, just – Again, we could do a whole show on the complexities of how people do edge routing. But to make a very long story short, there're a lot of complexity just in how people end up using DNS and how you can have intermediary clients that do caching in weird ways, or with BGP, how occasionally you can get routes that are just black hold and then there's basically no more route, and etc., etc., etc.

The thing about potentially offering something like XDS all the way to the mobile application is that although of course the actual XDS management server would have to be bootstrapped over something like DNS, there's probably no way around that. Once the mobile application is able to get configuration data, even if it's heavily cached, or we're you using different types of mechanisms to actually scale that, we can do much more real-time traffic shifting. We can do much more real-time load shedding. We can do that in a way that actually works similarly to how server-side is doing.

So, if you look at some of the manage control planes that are sprouting up. So Google has a product called Traffic Director. Amazon has a product called AppMesh. Of course, these are targeted towards server-side load-balancing and server-side policy. But you could pretty easily imagine some of these control planes extending all the way out to mobile, because if I have multiple data centers or I have multiple regions and I'm trying to load-balance and I'm trying to actually shed traffic between these regions either for performance or fault tolerance, why would I not potentially want that to apply all the way to my mobile client as well? So, that's just a small taste, right? I think that if we're successful, I think that we may be able to rethink how some of this policy actually happens, which is pretty exciting.

[00:48:10] JM: Whenever I talk to people about mobile application development, I get a sense that there is significantly less dynamism in how we can rollout these applications largely because of the App Store gating process. I think that has been eaten away a little bit with things like React Native where you have some dynamism that just gets bundled into the application. This might be another example where if you bundle in this dynamic networking technology, then maybe you could issue more dynamic networking updates, which should be cool.

But just because you're kind of like a backend developer who has waded into the mobile application space because of Envoy Mobile, what's your perspective on that App Store gating process? Is that useful, and how problematic is it for – Or to you?

[00:49:10] MK: I'll be perfectly honest in saying that I am far from an expert here. So, I don't think I can provide a truly educated perspective on that. I fully understand that both Apple and Google, they're there trying to have some quality control, and I think that's fine.

I think, as you've said, there's been a back-and-forth between application developers and Google and Apple in terms of what is allowed. What is not allowed? When you talk about dynamic code generation, one of the things that we're working on right now on server-side Envoy is actually web assembly support, and we'll have APIs on server-side where a management server can actually ship out precompiled web assembly bundles to Envoys and actually run them.

We believe that there is tremendous potential here for actually, today, within Envoy, we obviously have Lou. Yes, it is possible today to actually update Lua code from the management server. But Lua has its own issues. But from a web assembly approach, you can imagine a world in which people want to write extensions. Maybe those extensions are written in Go, or Rust, or C++, but they're inherently safe because their web assembly and they're written to a particular sandbox or a runtime. You can imagine a system in which in real-time, like almost like kernel level TCP dump style rules. Imagine that I can compile a small web assembly program and I can send that out to my server-side Envoys to actually do you extremely efficient matchings so that I can do traffic tapping or maybe blocking rules or things like that.

Now, you could think in the far future, if I can do that server-side Envoy, why couldn't I do that out to mobile? Why couldn't I potentially have a part of my network logic that's may be implemented in something like web assembly and it's shipped over XDS to the app and it's obviously cached.

Now, would that run afoul the Apple or Google App Store rules, it probably would today. But as you pointed out, there're things that are happening in the React world where there is dynamic programming happening. So, I think it's a very interesting topic. My suspicion is that Apple and Google overtime will probably adapt to the modern realities, right? But I think that it's going to be a give-and-take overtime. I don't see the App Store process going away. So, I think that from the Envoy Mobile perspective, we obviously are going to do everything within the rules, and then if we are successful and we want to do things in the future that might break the rules, we'll obviously have those conversations.

[00:52:01] JM: Yeah. I mean, I do have some respect for the App Store review process, just because our mobile devices are these things that we have come to rely on so much. If you have

an application that if you could skip the review process, you could get applications in the App Store that would just instantly brick your phone, and that's like bricking your lifeline in some conditions. I don't know. Again, I'm not an expert either, but I didn't want to give the impression that I'm just a total cowboy and want to remove the App Store review process.

[00:52:34] MK: No, of course. What happens overtime also is that I think if you look back 8, 10 years, I'm guessing that the App Store review process was even more important, because the operating system, the mobile operating system probably had less control and less safeguard over an application using too much battery or using too much network or stuff like that.

I think overtime it's been pretty clear that the mobile operating system vendors, they're obviously continuing to invest in those types of sandboxing or those types of safeguards, because the App Store review process is only so good, right? You obviously want the right safeguards in place so that even if something slips through, like you said, an application can't brick that device. So, it might even be the case that as the sandboxing gets better and better and better, maybe the review process can get a little less stringent. Again, this is not my area of expertise.

[00:53:36] JM: I know we're running out of time. It's been pretty interesting talking to you a few times over the last couple of years about how the challenges at Lyft have evolved overtime. The scalability challenges. Are you encountering anything new? How many years have you been there at this point?

[00:53:54] MK: Over four years. So, the time just flies.

[00:53:57] JM: Four years. So, what are the newer things that you're seeing these days that are presenting challenges that didn't exist before?

[00:54:06] MK: I think what you're seeing now is – Whereas I think before, at least on the Envoy or the networking side of things, we were developing features and doing all types of things that were aimed at – Again, whether that'd be retries or timeouts or feature after a feature after a feature. I would say that the major scaling problems at Lyft now, at least the ones I'm seeing from an infrastructure perspective, they're less technology problems and more people scaling problems.

By that, I mean, it's trying to figure out – We've gone from – I think when I joined, the whole company was probably less than 300 people for sure. So maybe 250, 275, and there's probably 75 developers or something like that. Now, I actually have no idea. Probably over 5000 people as a company, and there's over thousand developers, maybe even 1,500. I mean, we've probably at least 10X in size in the four years that I've been here.

I think what we've seen from the scaling perspective is that even if the technology maybe works, we've reached a scaling threshold where there's never enough documentation. When you're dealing with 1,500 people, even if one person only has a question every two weeks, if you start doing the math, that's a constant stream of questions.

So, where I'm actually leaving with this is that I think what we begin to realize at this human scale is that we just need less knobs, right? A concrete example of something that's killed us is manually configure circuit breakers. So trying to make sure that every application has the right safety guards in place or the right timeouts. It's just there's too much entropy. It's too difficult to manually ensure that all of these things are correct.

So now we're starting to invest in tooling to either kind of determine these settings from historical context or eventually just do away with the settings entirely and make them adaptive. But I think that's really where the scaling problems are. It's not in needing a specific piece of technology or a specific feature. It's more in how the system is configured.

So, I think at least for Lyft's perspective, development has moved really away from core Envoy. I think Lyft does very few Envoy features anymore and the effort has shift much more in terms of how this system is managed or how we expose all of these knobs to our users.

[00:56:44] JM: All right. Well, we did a show about that. That was our last conversation. Just as a final question, have there been any like personality adjustments that you've made to yourself or philosophical adjustments to accommodate that? Because I imagine, it just feels like if you let it, the environment is just pulling on you all the time to, "Hey, Matt. Do this thing. Do that thing." I don't know. Do you have any words of wisdom?

[00:57:13] MK: Wow! We should probably get together and do probably an entire show on this topic.

[00:57:18] JM: But we already did.

[00:57:19] MK: Well, yeah. Well, I don't know. I just feel like there's even more. I guess, for me, and this is coming at it from a personal perspective, is I have had to learn over the past couple of years both how to scale myself, but also just how to let the organization evolve in its own way or its own path, right? I think that just comes back to human scalability, is that when you move from a small organization to one that's ten times the size, I think things take on a life of their own, right?

I think, for me, it's just – I would say accepting some of the realities of how modern companies either do support, how they do dev ops and trying to tow the line of trying to figure out what is the right amount of documentation. When is documentation not going to work and we just have to automate something, right?

I think there's no easy answer there. But, for me, it's mostly just been trying to find the right balance of helping where I can, trying to steer open source where I can, try to steer the architecture where I can and try just to be more reasonable about how many hours are in the day. That's ultimately I think what it's come down to for me.

[00:58:44] JM: All right. Well, we'll have to continue alternating between Envoy related shows and human scalability related shows.

[00:58:52] MK: Yeah.

[00:58:52] JM: Matt, thanks for coming back on. It's always a pleasure.

[00:58:55] MK: Thanks so much.

[END OF INTERVIEW]

[00:59:00] JM: Podsheets is open source podcast hosting platform. We are building Podsheets with the learnings from Software Engineering Daily, and our goal is to be the best place to host and monetize your podcast.

If you've been thinking about starting a podcast, check out podsheets.com. We believe the best solution to podcasting will be open source, and we had a previous episode of Software Engineering Daily where we discussed the open source vision for Podsheets.

We're in the early days of podcasting, and there's never been a better time to start a podcast. We will help you through the hurdles of starting a podcast on Podsheets. We're already working on tools to help you with the complex process of finding advertisers for your podcast and working with the ads in your podcast. These are problems that we have encountered in Software Engineering Daily. We know them intimately, and we would love to help you get started with your podcast.

You can check out podsheets.com to get started as a podcaster today. Podcasting is as easy as blogging. If you've written a blog post, you can start a podcast. We'll help you through the process, and you can reach us at any time by emailing help at podsheets.com. We also have multiple other ways of getting in touch on Podsheets.

Podsheets is an open source podcast hosting platform, and I hope you start a podcast, because I am still running out of content to listen to. Start a podcast on podsheets.com.

[END]

