EPISODE 1199

[INTRODUCTION]

[00:00:00] JM: Video calling over the internet has experienced explosive growth in the last decade. In 2010, surveys estimated that around 1 in 5 Americans had tried online video calling for any reason. By May of 2020, that number had nearly tripped. A significant factor in the growth of video calling has been an open source project called WebRTC, or web real-time communication. WebRTC makes it possible to capture and stream audio or video data between browsers without the use of plug-ins or third-party software.

Daily is a developer platform that builds on WebRTC to provide real-time video APIs for developers. Developers can easily add video call widgets to their code which come with a set of default configurations for functions such as bandwidth management and cross-browser support. Daily also offers a set of front-end libraries and rest APIs for developers who want to build a customized experience.

Kwindla Hultman Kramer is a co-founder at Daily, and he's joined today by Wesley Faulkner who handles developer relations. They join the show to talk about the growth in demand for video calling services, building a developer friendly video calling API and what's next for video calling applications.

[INTERVIEW]

[00:01:15] JM: Guys, welcome to the show.

[00:01:17] WF: Thanks for having us.

[00:01:18] KHK: We're big fans.

[00:01:20] JM: Great to hear it. So you both work on Daily. Daily is a video streaming API, what is the purpose of a video streaming API?

[00:01:27] KHK: We try to make it really easy to add video to any website or app. You can embed a video call into a web UX using our APIs, for example, with just a few lines of code.

[00:01:37] JM: And why is that useful? What are some applications?

[00:01:41] KHK: We have a pretty broad range of customers, telehealth, online learning. There's a lot of growth in virtual classroom experiences of all kinds. Some social and gaming applications where people have audio and video side channels associated with gameplay. We also have a lot of customers who are doing really interesting work around the future of remote and distributed team collaboration. So for example, folks on this podcast might be interested in GitDuck, which embeds video into your programming IDE. Or a tool like Tandem, another of our customers that has an always-on virtual desktop for remote teams.

[00:02:19] WF: Pitch is also a really good use case, in which it's collaboration for video slides or presentations. So being able to collaborate in real-time with specific slides on a deck. So there's less confusions, less back and forth to make just work generally more efficient.

[00:02:38] JM: So we can start with just the simple case of two people being connected over a video streaming call. Tell me about what's going on under the hood when a call is initialized between two people.

[00:02:50] KHK: We're built on top of an open standard called WebRTC, which is embedded in all the major web browsers now and which are there are several open source implementations of. So initially your client hits our HTTP servers to do a basic account and config check. And then if the client is allowed to join the video call, it opens up a WebSocket connection to one of our media server clusters, which are deployed all over the world to be as close to users as possible. Then over that WebSocket connection we negotiate how the client is going to send audio and video packets. There's a lot of complexity to that negotiation. We can definitely dig

into it. But the short version is that if all goes well, the client starts sending and receiving audio, video and data packets over UDP as encrypted RTP streams.

[00:03:36] JM: So at a higher level, the main thing to know is that it's a WebRTC connection initialized between these two users, right?

[00:03:45] KHK: That's right. It's always WebRTC under the hood. The WebRTC primitives are fairly low-level. So we try to abstract as much as possible of the complexity of WebRTC. All the corner cases that are involved with making a call work everywhere in the world on any device and also all the complexities that are involved in scaling, scaling the size of calls and scaling across geography and scaling in terms of volume.

[00:04:12] JM: What are some of those corner cases?

[00:04:15] KHK: There's lots of different ways it can be hard to route packets, for example. We're all behind some kind of firewall these days. Maybe firewall is overstating it a little bit sometimes. But if you're on a home internet connection behind typical a cable modem, you don't actually have a public IP address. So the WebRTC machinery has to figure out how to get those packets through that cable router from you and back to you. If you're at work, you're probably behind a different kind of infrastructure frontend. If you're on a mobile device, you have a pretty different network configuration for cellular data. WebRTC is really good at handling those but there are lots of little details and we can handle lots of those details. For example, we can route through our servers if a peer-to-peer connection isn't possible to set up. We can also switch the whole call over to route through our servers if there's something like recording going on we're having a centralized server involved is useful. Or as a call scales, we can do a lot of server-side work to improve the quality of the call for everybody on the call. So we have a lot of flexibility with how we actually set up the call and we try to optimize the calls in real time for lots of different configurations.

[00:05:28] WF: There's also bandwidth management to determine the scale if it's high-resolution, low-resolutions, if it's higher quality, lower quality. A lot of those knobs need to

be tweaked depending on the use case. And so if you really want to over index on audio, you might want to do some bandwidth management to make sure that that portion of the stream is more intact. If there's going to be degradation, it's more on the video side than the audio side. So depending on the use case, it can vary if the person's on a slow connection, fast connection or even a mobile device.

[00:06:01] KHK: That's a really great point. I mean from a zoomed-out perspective, our two big engineering challenges or the unique work we do at Daily always comes back to two things. The fact that video in particular is a very high-bandwidth use case, and real-time video calls and other real-time video experiences can't ever buffer. So we have to deliver every packet within a couple hundred milliseconds at worst for the call experience to be good. So we're always thinking about volume and throughput of these UDP traffic streams.

[00:06:37] JM: So we start with the example of just the two-person video stream initialization. What about a call that scales up to hundreds of people? What is different about that?

[00:06:49] KHK: WebRTC architecturally is always peer-to-peer. You're always negotiating two endpoints of a connection. And so for a one-on-one call ideally, you actually do try to send the packets, the audio and video packets directly from participant A to participant B. As a call scales, you can't do that anymore because you would end up having a mesh network configuration where you're trying to push and pull way too much bandwidth. So you have to route through a central server. And that's the big thing that changes in terms of call logic. But it's also a really nice thing because we have a lot of things, as Wesley was saying, that we can do on the server. We can decrease the bandwidth selectively for each leg of the call as it goes through the server.

[00:07:36] WF: One thing to note is that if it was truly peer-to-peer, that means that every client would be sending both the video and audio stream to every single other client, which would dramatically increase the amount of processing power each one would have to do to both support all those streams and also to decode all of those streams. But by using a server in the

middle to handle those, you're able to combine them into one stream to send to each individual client, which makes things a lot more bandwidth friendly and usable.

[00:08:09] KHK: The growth of larger call experiences of all kinds was the biggest single thing that happened for our users in 2020. We went into 2020 with the vast majority of the session minutes we hosted being calls of fewer than 10 people. And by the end of 2020 the majority of our session minutes were calls with 30, 40, 50, 100 people in them. So the whole market scrambled to try to build great tools for things like virtual conferences and large online classes for obvious reasons because 2020 forced a whole lot of changes in where people were working from and how they were working.

So the Holy Grail now for a lot of us who are working on WebRTC-based platforms is to scale really well for these larger and hybrid experiences. We have a number of customers who are building things like a fitness application where the instructor, say, a yoga class where the instructor wants to be able to teach 500 people at once, but you want to join the yoga class and feel like you're joining that class with your friends. So 500 people in that session get the instructor's video, but each person in that session also just gets video from three or four or five of their friends. So it's a really interesting kind of hybrid scaling problem.

[00:09:24] JM: So you've described these two modes, the peer-to-peer network mode and the media server mode. Can you describe these two modes in more detail?

[00:09:34] KHK: Sure. For peer-to-peer, we're taking each client's network information, their local IP address, the IP address of their public-facing firewall and some port numbers that are available and we're trying to send packets through all of the options that we enumerate from the pairwise combination of all of the network information on each side. And we, in theory, should always be able to get packets sort of through the firewall on both sides and send packets directly. That works really well in about 85% of cases. In the 15% of cases where it fails, we end up routing the packets through our servers in a really simple way. So it still seems like a peer-to-peer connection, but we're bouncing the packets off of a server where we host in the cloud somewhere.

As the calls get bigger, we can't connect every peer to every other peer. So dynamically during the call we switch modes and we set up a new single peer, sort of super peer connection to one of our media servers, and everybody in the call starts sending audio and video through that server and getting all of their audio and video from that server. And so it goes from a mesh topology to a star topology during the call. And from a very low-level perspective, it's still all the same building blocks. It's still all the same core WebRTC primitives. If you've looked at the WebRTC kind of JavaScript objects, it's an RTC peer connection an RTP sender at an RTP receiver, but the server is doing a ton of invisible work behind the scenes to try to manage the bandwidth to and from each person who's connected to it.

[00:11:11] JM: Can we zoom-in on that media server a little bit more and tell me more about what the server is doing to handle all these concurrent video streams that have to be sent out in this star architecture?

[00:11:25] KHK: We built on top of a really lovely open source toolkit for WebRTC servers called mediasoup. And mediasoup is a set of primitives for processing and forwarding media and data packets of the kind that WebRTC implementation send from a client. So that server has a stateful WebSocket connection that's doing call state management and distributing client state so that everybody in the call knows what else everybody in the call is trying to do. And then it has a lot of machinery for statelessly processing UDP packets that are just flowing into the server and need to go somewhere. And these packets come in encrypted. So the server has to decrypt each stream. Figure out how many different places need to get which of the packets and then send the packets along.

One of the big building blocks for bandwidth management is called Simulcast. So by default in a call, we send three layers, three different qualities of encoding up from each client. We send a really low bandwidth, low resolution layer, a kind of middle bandwidth, middle resolution layer, and then a really high quality layer. So one of the things the media server does is it peels off Simulcast layers when it forwards a video stream. So you might be on a cellphone with not much bandwidth available. The media server is going to send you the lowest quality video layer

that I'm sending. On the other hand, somebody on a really great network connection with enough CPU to decode lots of videos coming in at once at high quality is going to get the highest quality bandwidth layer for me.

So a lot of what the media server is doing is trying to figure out what is the available bandwidth to each client and from each client sending control signals and managing the state to peel off the right simulcast layers is the way we talk about it internally. Then there's a bunch of other stuff that server can do that are kind of additional features that lots of people use in video calls. So the server could record some or all of the tracks. The server can transcode in real-time and send out to another service for live broadcast. We partner with the folks at Mux that you've had on the podcast for live broadcast to very large audiences of a call. We try to use those servers to add features that help developers build new things into their apps as well as to make sure the calls always work really well.

[00:13:49] WF: If you're looking this up on your own, it'd be helpful to know that this is called an SFU, which stands for selective forwarding unit, and that's one of the terminologies instead of P2P, which was the previous case that we talked to when it's more of a peer-to-peer connection.

[00:14:08] JM: Got you. So the media server, it's trying to be aware of available bandwidth. So is there a notion of a bit rate ladder like where you can have the server send lower bit rate connections to people who have poor connection speeds?

[00:14:29] WF: That's right. So in I think the HLS world, you would call it a bitrate ladder. In our world we call it, depending on exactly how we're doing it, Simulcast or scalable video codecs. That's definitely a big part of what the media server, the selective forwarding unit server does. It figures out how much bandwidth is available downstream to each client and it sends a lower bit rate if it possibly can to the clients that need a lower bit rate.

Today the way to do that across kind of all the major browsers is called Simulcast, and we send through the media server multiple separate encoding layers. So they're all encoded

separately, although we can reuse some of the CPU on the encoding side in that encoding pipeline. So it's not too bad. The future is the next generation of video codex called SVC or scalable video codecs. So as we move from h.264 to h.265 and from VP8 to VP9, we'll have more ways to save bandwidth and CPU, because these new generation of codecs were designed from the ground up to be really efficient in terms of how you can do a bitrate ladder kind of operation on them in real-time.

[00:15:43] JM: You also mentioned a partnership with Mux, which is a video API. I'm interested in that, because Mux is a company I would think of almost as a competitor to Daily. How do you utilize Mux? Talk a little bit more about that.

[00:15:58] KHK: It turns out that Daily and Mux are almost perfectly complementary, because even though we both focus on video and we both try really hard to build great APIs that make scalable video easy from a developer perspective, Mux focuses on broadcast distribution to large audiences using a technology stack built on HLS. We focus on real-time sub-200 millisecond interactive video experiences built on the WebRTC standards. Those are different enough use cases that there's a pretty bright line between them and the easiest way to think about it is latency. If you need 200 millisecond latency, the kind of latency that's important if you're talking to someone, you really need WebRTC.

On the other hand, if you can tolerate a few seconds of latency, then HLS is a really nice technology stack because it scales incredibly well both from a volume and a cost perspective. So the canonical customer for us has 30-person in a video call. The canonical customer for Mux has 10,000 people watching a live stream.

[00:17:11] JM: Gotcha. So there's a certain scalability point where you need to switch over to that kind of infrastructure.

[00:17:18] KHK: That's right. Both from kind of a reliability an engineering perspective and from a cost perspective. HLS, the standard MUX, the MUX folks are the best in the world at is built on top of HTTP’s standard infrastructure primitives. So for example, you can pull HLS

video from a CDN and that gives you lots of great engineering work that's been done over 40 years to make HTTP work great everywhere in the world at very low cost. We can't use CDNs. It's a slight oversimplification. But it's fair to say we're 100% UDP, not TCP. So HTTP and TCP are not available to us because they're too high latency. So we have to build this pretty complicated custom engineering tech stack to route UDP packets in real-time everywhere in the world at very low latency.

[00:18:16] JM: Let's talk a little bit more about infrastructure. What cloud providers do you guys use?

[00:18:21] KHK: We're mostly AWS, and we sort of have two parts of our world. One is a pretty standard HTTP and REST API serving infrastructure. We use things like Elastic Beanstalk and Aurora MySQL. It's straightforward in the sense that lots of people have solved those problems before us and so we try to live on top of known best practices. We do a few billion requests a month. So it's not small, but it is really, really standard. Then we have the completely custom architecture for our WebSocket and UDP media stuff that's completely unique to doing WebRTC at scale. And there are no kind of out of the box solutions for that. There are no services for that. In fact, that's what we are. We're a service for that. So we do things like least path routing to figure out which of our global clusters a client should connect to. We have lots and lots of custom code built on top of the open source mediasoup framework to manage lots of different kinds of calls and lots of different kinds of user experiences and do the bandwidth management we talked about earlier. And all of that is it's always a work in progress. We're always rolling out new data centers. We're always improving the logic inside our media servers. Those are mostly on AWS as well. And we mostly use ECS to kind of manage our containers. But for us, even something like ECS is not super helpful. In fact we sort of talk regularly about whether we should just be on top of kind of just bare EC2 instead of ECS with our container architecture. Because there are no – For example, there are no good out-of-the-box auto-scaling algorithms that we can just plug in. We have to write all the auto scaling stuff ourselves because our workloads are so unique. We have to write all the deployment and connection draining stuff ourselves because, for example, we might have a video call hosted on one of our instances that's going to last another 10 or 12 hours. We can't take that instance

down after connection draining for a couple of minutes, which is the assumption built into the most of the container architectures. So there's a lot of super interesting DevOps work, I think, that we spend a lot of time on in addition to the sort of algorithm and C++ level packet routing kind of stuff.

[00:20:39] JM: Could you use Fargate containers?

[00:20:41] KHK: It's a really good question. So we built a little Fargate prototype at one point. It's worth understanding the evolution of all of the platform tools because they really are great and they evolve so in such interesting ways. Fargate doesn't help us that much because, again, the assumptions that the Fargate folks had in mind when they built Fargate out were serving HTTP requests, and totally understandably because the world revolves around HTTP. Our HTTP is a solved problem for us. Like we could migrate some of our HTTP infrastructure to Fargate, but there aren't really advantages over Elastic Beanstalk for us, vis-a-vis Fargate. And Fargate doesn't really have any of the kind of routing and scaling stuff that we do custom for UDP and websocket traffic built in. So I think Fargate is super interesting as a direction things are going. It's not yet anywhere close to being able to help us.

[00:21:36] JM: So you mentioned there's a lot of lower level C++ work you have to do. Tell me more about the C++ stuff, then we can talk about other programming languages.

[00:21:45] KHK: Oh yeah, sure. So some stuff has to be fast. We decrypt the incoming UDP packets, the RTP media streams, and we need to look at those packets and figure out what they are and where they should go and then we switch them to a different routing plane, virtually speaking, inside our server code, and then we re-encrypt them and send them out. So that code is tricky from a logic perspective and it has to run very, very fast with no blocking and no hiccups no garbage collection. So mediasoup is written in C++. We had a bunch of C++ code before we started using mediasoup three years ago now. So we've just sort of combined our work with the really great open source stuff that the mediasoup maintainers are doing, and that's all C++.

We are starting to do a bunch of stuff in Rust as well. And Rust is interesting to us from a few perspectives. It's a language that we think we can make things as fast as C++ in, but it's newer. So it's cleaner from a lot of perspectives and it has some really nice memory safety design stuff baked in. And then the other reason we like Rust is WebAssembly is a first class part of the Rust ecosystem. Compiling to WebAssembly is something people who are using Rust are doing a lot. The Rust core team cares a lot about compiling to WebAssembly. And WebAssembly is a way we can write code that runs across devices. It can run in the browser, which is a super important part of the WebRTC world. It can run on android. It can run on iOS. It can run in our servers. And maybe someday, some of the WebAssembly work that people like Cloudflare and Fastly are doing would let us deploy some of our code on top of those platforms as well.

[00:23:37] JM: And so what other languages do you use?

[00:23:39] KHK: We use C++, JavaScript, and relatively recently, but increasingly, Rust. And then internally just for various kind of big data analytics types of things we use Python.

[00:23:53] JM: What do you use JavaScript for?

[00:23:55] KHK: Our client-side libraries that our customers use to embed video and audio in web browsers are all JavaScript. And then we have a whole lot of core library code that gets loaded up in the web browser that's written in JavaScript as well. Some of that's getting migrated to Rust and WebAssembly, but we have a really large amount of JavaScript code. So that migrating that code is going to take some time.

[00:24:21] JM: And so you start to do some analytics? What are you looking at in terms of the data set that you accumulate?

[00:24:28] KHK: Yeah. We say internally that we have two things that everything else we do flows up into or that guide our priorities, and one is developer time to value and the other is called quality and reliability. For developer time to value, we mean can you get up and running

with our APIs really quickly with very few surprises and do really sophisticated things? And that implicates every part of our stack from API design of our Rest APIs to API design and library architecture for our client-side libraries that you load up in the web browser through to our docs and our sample code. And then for call quality and reliability, because every call has to just work great or we're not doing our job, we collect data, completely anonymized metrics data from every single client in every single call. So we have a way to look. We can slice and dice it in all sorts of different cohorts, but we have a way to look at when calls didn't work as well as we hoped they did and to try to figure out why. And that turns out to be useful from two really big angles. One is we can support our customers really well.

So if a customer pings us and says, “Hey, somebody who uses my app had a bad call.” We can actually dig in with them to that call. And because they know who was on that call even though the date is anonymized, they can figure out what happened with each client. The other thing we can do is analyze across all of that data to try to understand patterns. So maybe a bunch of people in Germany on a particular mobile provider had a tough month. Error rates on calls went from .5% to 3% or something like that. Then we can dig in more and try to understand why. Was it a bug in our code? Was it something we can figure out and then work around in that geography? Was it an android system update that changed something that wasn't expected to break calls but did break some calls?

So we do a combination of really hands-on deep dives into individual call metrics records and we also do the big data analytics across kind of all of the calls we do. And I think we'll do that forever and ever because it's just a never-ending and really interesting set of kind of knobs we can turn to make call quality and call reliability asymptotically approaching 100% awesome.

[00:26:50] WF: And of course we also have a portion of this data like a reduced set that's available for every customer through the dashboard and control panel. So make sure that if they want to get immediate feedback on a call that just happened that that data is available for them, we have different retention policies depending on the level. But all tiers have access to this data for every call. So they don't nearly need to rely on us for every single call or for the

analytics. But if they need a deep dive or some help with understanding the data, we're always available to help them too.

[00:27:24] KHK: That's a really great point. I mean one of the things we try to do is make it possible for developers to build great stuff without knowing any of the details about video. But we also want to make it possible for people to get better and better at supporting video-centric use cases. So in a sense, we're a kind of application monitoring or observability service for our customers as well as a set of APIs that they build features with if we're doing our job right.

As Wesley said, the dashboard is a big focus for us. We put a lot of work into rolling out the kind of call quality and analytics data in a gooey fashion accessible in the dashboard last year. That was sort of a big push for us and we're really happy with how it came out. And now we have another 100 features we want to add to it.

[00:28:10] WF: And the fidelity is the same no matter if you're on the free tier or the highest cost tier. It's just the retention is different depending on what level you're on.

[00:28:19] KHK: Yeah. We want all our customers to have access to great data.

[00:28:24] JM: The monitoring challenge here is really interesting because you've got all these different users who are using the API and you've got high-bandwidth connections on each of these users. What kinds of telemetry data do you collect and how do you make sure the service is running at high quality?

[00:28:44] KHK: Yeah, it's a really great question. So we collect a bunch of stuff from our servers. It's pretty traditional observability stuff from our servers, but focused on UDP traffic as much as on HTTP traffic. So kind of real-time traffic flows through our servers and all of the kind of indicators of any kind of CPU pressure or slowdown in packet rate that we can kind of hook into. And then from the clients, we send up – We log events and we log metrics like frame rate, overall bandwidth used, packet loss on the UDP streams, a bunch of timing and jitter stuff and we bundle those metrics up and we send a little blob of them to our infrastructure every 15

seconds. And that's the basic building block for how we can analyze not just what happened on our servers but what happened on the clients. Our goal, always a kind of receding goal as we get better and better at everything, but our goal is to have complete end-to-end visibility into what happened from each client through our servers through to the other clients that were receiving all of the audio and video from that first client and correlating all of that data. Correlating between clients, correlating between the clients and our servers with timestamps and session IDs. And, again, we have to anonymize all this because we take privacy really, really, really, really seriously.

So there's a logging volume challenge, a privacy requirement. And then how do we use that data effectively? Because it's a huge huge volume of data. And the way we use it effectively is, A, we put it in the dashboards for customers like Wesley said. And B, we work with our customers anytime they ping us about a call they have concerns about and do a really hands-on deep dive with them if they want to. And then C, we do the sort of big data analytics machine learning type stuff to look for patterns, patterns that we should know about.

[00:30:38] WF: And because we are an API, there's more data that can be collected on the implementation side. So if you're a big company using ingesting our API to help with your own implementation, that client side could take more data if needed to help them with their troubleshooting on their side that they can retain and they can control. So we don't prevent that if that's needed for more of a diagnostic side. So you can build on top of our tools to get more insights, but we do the base level just to make sure that we don't invade privacy or make sure we capture any data that shouldn't be captured.

[00:31:14] KHK: I think one of the things that happens when you build good telemetry systems is you learn a bunch more about the questions you should be asking kind of iteratively and sometimes even accidentally. But big data's super fun. I mean I think anybody who's even got a personal website and does like curiosity queries like, "How long did it take for iPhone versions to update across my user base?" knows that it's just really addictive to dig into high-quality big datasets. So we can do things like, say, "Did the frame rate drop when customers upgraded from Chrome 86 to 87? If it did, we want to understand why and we want

to tweak the defaults that we try to set perfectly and make our developers never have to worry about," because every browser release changes something. And worst case, browser releases break something fundamental at the WebRTC layer and we work around that for our customers. But best case, there's like a small quality change that we can leverage made possible by a new browser update. Then we can put a little flag in our code. We can change some small thing about how we handle something that becomes possible with an improved API in a new browser. Again, invisible to most of our customers, we can improve the call quality experience. And we could spend all day just looking at the data we have. So we try to balance, figuring stuff out based on the data and talking to customers to learn what our customers' pain points are.

[00:32:47] JM: What has been a particularly challenging moment as the company has grown?

[00:32:53] KHK: I think everything about real-time video is challenging and sometimes stressful. Daily's co-founder, Nina, says sometimes late at night that if she'd thought in advance about how hard it is to run a system that has several nines of uptime, she might have done something different with her life because our system can just never go down. We try really hard to never have maintenance windows. Even 30 seconds of downtime means some of our customers get disrupted with a feature that's critical for most of our customers, video call. And that's really, really a challenge we always try to try to live up to, like uptime and reliability of our core services.

The other thing that's challenging is real-time video at scale broadly accessible built into lots of different kinds of applications kind of by default available in your browser is new. We started Daily in 2016 and we started Daily because we love the engineering challenges around video. We've been working on video a long time and we thought that the newly available open standard of WebRTC was going to be an inflection point in the growth of video and video being kind of embedded everywhere and just super, super common, and we're really excited about contributing to that, trying to build great tools for other engineers on top of that. Like just a really fun and really interesting moment to start a company around.

We were probably too early. I mean when we started, WebRTC was in Chrome and kind of in Firefox and not in Safari at all, and every release of Chrome broke something. Firefox was a little more stable, but also made some different choices than Chrome about which parts of the spec to support win. So cross-browser interop was like I probably – When we were three or four people and I was writing a big chunk of our code personally, I probably spent half my time working on browser interop, which it's a really good thing to solve for customers. It also just kind of feels like pushing a rock up a hill when every browser update breaks something. I probably have a bunch of tweets in my – Tweet stream from 2016, 2017, 2018 saying some version of building for video feels like building for the web in like 1998 when you had to special case all the different web browsers with your HTLM and your CSS. That's stabilized a lot, but it's not stabilized completely. Safari 14 release broke some stuff. Most of which, but not quite, all of which we can work around at a level that's lower and invisible to our customers. So just being on the cutting edge of a new wave of standards and kind of ecosystem adoption is super challenging. It's fulfilling when we get it right. It's really stressful when we're struggling.

[00:35:46] WF: I have to say from the inside, I can see the growth of the company from even a year ago to now has also been something that has done a lot of headroom in terms of the company has grown fairly large not just because of the employee base, but because of the use cases. Before, it just used to be video was a square and a grid of squares, but now it could be round. We have spatial audio. We have like snap filters where people are trying to add augmented reality into the stream. We have – Of course COVID happened when the one-to-many and to like whole companies trying to do video chat and video conferencing. All of those new use cases that continue to evolve every single day, the new applications that are trying to use video and Daily's commitment to try to enable those new use cases using our API. I've seen a flurry of development to try to get these concerns around video safety, reliability, encryption, all of that, all those features around video that people took for granted or didn't worry about. I see that being implemented on top of the WebRTC spec. And so it's making it so complicated under the hood and us trying to make it so much easier. So a lot of people who see these cool new uses are able to bring those as actual products to their customers.

[00:37:11] KHK: That's such a great point. I mean one of the things that's both really fun and really challenging about the goals we've set for ourselves are we want to support every kind of developer. So you can get started with our pre-built UI with literally two lines of JavaScript. And a lot of people are using our API in that way on top of low-code and no-code platforms, embedding video calls into pages on Webflow or WordPress. And then at the other end of the spectrum we have customers who are what we sometimes say internally are deconstructing the whole idea of a video call and putting together brand new experiences. Those customers have really different concerns at least in part. They're a lot more worried about the developer ergonomics of our low-level APIs, client-side APIs, and they also tend to exercise code paths that our pre-built UI doesn't just because there are such a variety of different things people are trying to use. So those customers find bugs or performance shortcomings in our code that are completely different from what our kind of low-code developers do. And that's code complexity. It's product feature complexity. It's definitely a challenge. It's really fun to work with both kinds of developers though. The low-code or the pre-built UI developers kind of teach us a lot about how video is getting embedded into kind of everything we all do online every day. And then the custom UX developers show us the future. Like what are these UIs going to look like when they're not, as Wesley said, boxes within boxes anymore? When they're little floating heads and there are AI GPU filters changing them on the fly and they're getting kind of recorded and archived and reused and rebroadcast in all sorts of different ways?

So we're sort of violating one of the good rules of thumb of a startup, which is that you should focus really well. I mean we are focused 100% on video and real-time video, but we aren't specializing in what kind of developer we think is our customer. We're trying to serve every kind of developer in every use case for real-time video.

[00:39:20] JM: So those two use cases, the very experienced JavaScript developer who builds really cool front-end experiences that are totally unprecedented versus the low-code developer that's building maybe an app in Webflow or something like that. Can you go deeper into the dichotomy between these two kinds of developers?

[00:39:45] KHK: Sure. It varies a lot. So I'm going to try to oversimplify a little bit. But the low-code developer often has some really, really compelling core use case that they know a lot about. Like we have a customer who does live cattle auctions, and they're not first and foremost a kind of hard tech developer, but they know how to build the thing that their customer needs and they just need super reliable, easy to use video as a core component of what they're doing. So in the live auction use case, if you want to extend the people who can participate in an auction to people who can't actually physically show up in one room, you need a really good, really reliable way of having 500 people join a video broadcast that's low latency enough that the auctioneer can actually kind of manage the bids. And we've got lots and lots of different versions of those kinds of use cases where the core workflow around solving a particular problem is the customer's expertise. And the video is critical, but it just has to work. And they don't want to worry about the video and they don't want to become video experts because they're experts in a bunch of other stuff that's already hard and specific.

On the other end of the spectrum, people who are building really new video experiences often do care a lot about the details of how video works, because they bump up against limitations. And they need to understand the kind of interface between what you can do with a system and what they want to kind of experiment with for their users. So our customer tandem who's always on virtual office has a bunch of different ways people use audio and video and they care about a bunch of different things like almost instantaneous push to talk with anybody on your team.

We work with them to understand like what drives the value of this new experience? And then we try to improve our code or build new features to do something like reduce the latency of the first time to audio byte being received on any connected client. And so the experience is a little different in the sense that we learn from both ends. Like what just has to work versus where the frontiers are that kind of don't work yet, but we can probably make them work with some engineering effort. And we probably should because the world is moving in that direction.

One other thing that's interesting about the more custom UX developers, they often come to us after having done a prototype on top of WebRTC themselves. So if you're a really experienced

engineer and you look at the MDN docs for for WebRTC in a browser, you realize quite rightly that you can probably build a prototype of the UX you want that you can test with customers in a couple of days. And as an experienced engineer, that's a known workflow. What you realize after you've built that prototype is that all kinds of cross-browser real-world client quirk scaling the size of the calls, scaling the volume of your usage are super non-trivial. And so those folks often then start to look at various open source projects and various services like us. And so part of what we do is try to tell people, “Hey, here are the problems you're going to hit as you move from prototype to something you're rolling out at scale.”

We don't want you to use daily, unless we're the perfect solution for you. But we definitely want to help you kind of understand what's scaling up a product that includes video looks like. And our kind of early sales funnel with the experience developer part of our customer base is all let us just tell you a little bit about what we've seen and then you can decide if we're a good fit for you. Because there's so much growth and interest in video that we would rather have a kind of general soft sell. We'll just tell you what we know. You're an experienced developer, you can decide what the right platform for you is.

[00:43:45] JM: All right. Well, let's start to close off. I'd like to get both of your perspective on the future and where Daily is going. Where video streaming is going? Just give me your thoughts on where things are headed.

[00:43:58] WF: From the previous question, we're talking about experienced developers and how they use our platform. And what I'm seeing also from that perspective and I'll see more I think this coming year, is that video is really good enhancement. It's a good seasoning for your application. So even if you're not a video-first type of application that you want to make sure that video is always on with every interaction. It's a good backup and a good – If text doesn't work, move to video. For instance, Slack has video and has had a video integrated for a very long time, but people don't think of the video portion when they think of Slack. But when text breaks down and you're like, “Let's just hop on a call.” Being able to add that to your application or be a backup to the primary form of either collaboration or for presentation, I see

video being a really good enhancement for a lot of applications that you wouldn't necessarily think of video first.

And so in terms of like taking a traditional program, let's say, like Microsoft Word and then having that collaboration, but then turning in video, adding video into that collaboration can make it that much better. So I see a lot of momentum and video for sure and video-first applications, but I am now starting to see video also type of applications and I think we're going to also see a lot more of that. That and stories. People are adding stories to applications. But specific to video, I see that being a good enhancement in everyday applications that people use.

[00:45:35] KHK: I completely agree with Wesley. I think there are a couple of interesting things that are happening in parallel, and one is that video is getting added as a value add to lots and lots of different things we all do online. Some of that's real-time video like we support. Some of it is non-real-time stuff like stories getting added to every app. And then the second category is new things you couldn't do before where video is just completely core. So if you're adding video to a web collaboration app, it's really valuable. It probably doesn't fundamentally change how you're collaborating with your teammates. It just makes it better. But new things like an always-on virtual office for your team, that's meaningfully different than collaborating over email in the same way, or collaborating over just Slack in the same way that Slack was meaningfully different from collaborating over just email.

We see both of those trends happening and kind of reinforcing each other. I mean we started the company because we thought video was going to be just ubiquitous. And it was a hard story to tell and get people to believe it in 2016. I mean all the investors we pitch did their work over phone calls and in-person. And now every investor takes pictures over video. So just a kind of tiny microcosm in our Silicon Valley startup world of how things have changed. It's not hard for us anymore to convince people that you know video is not niche and video is going to grow.

I think we still believe in video being ubiquitous in a way that other people who aren't quite as close to it as us do. I see all kinds of trends towards video just being assumed that feel like the early days of a massive change. I mean one little one is a few years ago – It's hard to remember now, but a few years ago, no websites had live chat for customer support. You filled out a form. And then tools like Intercom came along and built really great low-friction UX for live chat. And now if I go to a website and I have something I need to communicate with that company about, I'm actually kind of frustrated if there's not live chat.

We're starting to see our customers add video to live chat, and that's as big an improvement for certain kinds of customer support or sales conversations as adding live chat is over adding an email form. And we just see that across the board. We see it in things like telehealth where there's an increasing assumption that you shouldn't have to go to the doctor's office to get really good care. In online learning where the quality of curriculum and now live instructor experience that's available online over video is just so, so high, and it opens up and democratizes all kinds of online learning. And even things like IoT and robotics are starting to change because video is really, really, really capable and available everywhere.

We tend to think that kind of there are three ingredients for a big, big change in how we all use technology. One is the kind of core tech limitations. And we've gotten a long way up the scaling curve with CPU and network available everywhere that you can pretty much assume video is going to work for you even on a cellphone. And then the second ingredient is the kind of technical ecosystem and infrastructure. So that's things like WebRTC. And we hope the platform we're building. And then the third ingredient is people's expectations and habits. And back when we all picked up the phone to talk to each other, our expectation was that audio was better than audio plus video. I think you see with people who are teenagers today, their assumption is video is better than audio and video. I mean video plus audio is better than audio only. And that's a huge generational change. I think the rest of us who are a little older than teenagers are coming along behind and starting to adjust our expectations in the same way. So as we all start to assume that video should be everywhere because there's value there, video will become everywhere.

[00:49:23] WF: I just wanted to chime in and just say that so much communication is non-verbal, that you read off cues, you read off if someone's in a good, mood bad mood, if they seem disheveled or if they seem full of energy. A lot of that you can take through audio, but some of that you really need video to understand people's personal cues and how they symbolize, how they feel. And that connection is really hard to replicate unless you have video.

[00:49:51] JM: Nice. Well, it seems like a good place to wind down. Do you guys have anything else you want to add?

[00:49:55] KHK: No. Thanks for the conversation. This was really fun.

[00:49:58] JM: Okay. Likewise. Great product, very interesting. Looking forward to see where it goes.

[00:50:03] KHK: Thank you for your time. Appreciate it.

[END]