EPISODE 478

[INTRODUCTION]

[0:00:00.3] JM: Do you remember when the best game that you could play on your phone was Snake? In 1998, Snake was preloaded on Nokia phones and it was massively popular. If you're on the younger side of the audience, you may not remember what Snake is. You can take a look at the show notes to see a picture from Snake. It's a very primitive game.

But that same year in 1998, Half Life won game of the year on PC. This was a first-person shooter with excellent 3D graphics. Metal Gear Solid came out for Playstation. The first version of Star Craft also came out in 1998, and this was a real-time strategy game with excellent graphics and lots of interaction and animations.

In 1998, a few people would have anticipated that games with as much interactivity as Star Craft would be played on mobile phones 20 years later. Today, mobile phones have the graphics and the processing power of a desktop gaming PC from two decades ago. One thing still separates desktop gaming from mobile gaming, the network.

With desktop gaming, users have a reliable wired connection that keeps their packets moving over the network with speeds that let them compete reliably with other users. With mobile gaming, the network can be flaky. How do we architect real-time strategy games that can be played over an intermittent network connection?

Yan Cui is an engineer at Space Ape Games, a company that makes interactive multiplier games for mobile devices. In a previous episode, Yan described his work re-architecting a social networking startup called Yubl, where the costs had gotten out of control. That's a great episode and I recommend checking it out if you have not heard it. It's called Serverless Startup.

Yan has a skill for describing software architecture and explaining the tradeoffs. When architecting a multiplayer mobile game, there are so many tradeoffs to consider. What do you build and what do you buy? Do you centralize your geographical deployment to make it easier to reconcile the conflicts that are inevitably going to occur or do you spread your server deployment out globally? What is the interaction between the mobile clients and the server?

The question of interaction between client and server for a mobile game has lessons that are actually important for anyone building a highly interactive mobile application. This is a pattern that you sometimes see in computer science where the cutting edge is often done at the gaming application type where games are the cutting edge of something like computer graphics which leads to developments in processors, like GPUs, which lead to a new tool that machine learning people can use. Games are actually quite important in the evolution of computer science.

One example I think about in highly interactive mobile applications is Uber. When I make a request for a car, I can look at my phone and I can see the car on the map slowly approaching me. The driver can look at his phone and he can see the passenger. He can see if the passenger crosses the street and if he needs to go one side of the street or the other in order to pick up that passenger. This real-time communication is accomplished by synchronizing the data from the driver's phone and the passenger's phone in a centralized server and then sending the synchronized state of the world back out to me and the driver.

How much data does the centralized server need to get from the mobile phones in order to create that synchronized state? How often does the centralized server need to make the requests to each mobile phone? The answers to these questions are going to vary based on the bandwidth and the device type and the phone batter life and all these other factors.

Just like in Uber, when you're trying to figure how often do I need to ping the mobile phones in order to keep an updated state of what is going on in the different mobile phones, you got similar problems in mobile game engineering, because the users are in different places on a virtual map. The users are fighting each other. They're trying to avoid enemies. They're trying to steal power ups from each other.

A mobile game can be even more interactive than a ridesharing app like Uber. So the questions of data synchronization can be even harder to answer, and we go into great detail in this episode with Yen Cui.

On Software Engineering Daily we have explored this topic of real-time data synchronization in our past episodes about the infrastructure of Uber and Lyft, and you can find these old episodes by downloading the Software Engineering Daily app for iOS and for android.

In other podcast players, you can only access the most recent 100 episodes, but with these apps you can find all of our episodes and they're indexed with a nice UI that's custom-built for Software Engineering Daily listeners. With these apps we're building a new way to consume content about software engineering, and it's all open-source. You can find it at github.com/softwareengineeringdaily. If you're looking for an open-source project to hack on, we would love to get your help.

With that, let's get on to this episode with Yen Cui.

[SPONSOR MESSAGE]

[0:05:42.6] JM: Consensys is the largest blockchain company focused on building software on the Ethereum platform. They've developed Truffle, the most popular Ethereum development framework. Truffle is your Ethereum Swiss Army knife and it's available for free by going to softwareengineeringdaily.com/consensys.

Nearly 200,000 developers are working with Truffle and you can download it today and start building your own software on Ethereum. Find blogs and tutorials there as well to get started. Truffle is written in JavaScript in a completely modular fashion allowing you to pick and choose the functionality you'd like to use. For example, you could use Truffle as a library in our own tool using only the modules that you need. This lets you take advantage of powerful features, like Truffle migrations in your own command line tools.

Consensys has built several of the leading dapps, decentralized applications, in the Ethereum ecosystem and offers some of the most popular free Ethereum developer tools such as Metamask, Infura and Truffle. These tools are essential if you're thinking about building an Ethereum dapp.

Learn about Truffle and download it directly from softwareengineeringdaily.com/consensys to get going on Ethereum development. If you want to hear a show about one of the topics that Consensys knows a lot about, send me a tweet @software_daily and tag @consensys, that's consensys with a Y instead of a U, Consensys, with the topic that you would like to hear about. Tag both of us and let us know the topics that you're interested in hearing about.

Thank you.

[INTERVIEW]

[0:07:33.2] JM: Yen Cui, you are an engineer at Space Ape Games. You have been on previously to discuss your time at Yubl, which was a social networking startup and in that episode you discussed your refactoring of Yubl, and it was a really popular episode where you were talking about just all these refactoring and this efficiency gains that you've got that were ultimately undermined by the problems of the business, the fundamental problems of the business, but it's great to have you back, Yen.

[0:08:09.5] YC: It's great to be back, Jeff. Thanks for getting me back on here.

[0:08:13.3] JM: Absolutely. Today we're talking about a more optimistic scenario. You have started this job at Space Ape Games. You've been working there for a while, and I watched a presentation that you gave about scalable multiplayer games and I'm excited to dive in to that. I thought we should just start off by talking a little bit about the growth of mobile games, because I've not covered this at all. These mobile games, I think we've basically — The amount of time people have spent playing console games has probably dropped or perhaps mobile games are complementary to that.

Why don't you just talk about your perspective on the growth of mobile gaming?

[0:08:56.4] YC: Yes. I guess going back a couple of years ago I was at a company called GamesIS where I was building, again, casual games, but mostly on Facebook, and straight away, even doing that couple of years, maybe two or three years when Facebook games was the biggest thing around. You have all these games that's got 200 million users, daily active users. Pretty quickly that landscape shifted to mobile and that mobile space has just continued to grow.

Compared to, I guess, console games, it's definitely on decline. At the same time, we are seeing mobile games that are getting more sophisticated and are getting more core as well. For a very long time, mobile games was very focused on the casual audience, but now we're seeing games that are more and more targeting towards the core games, and this perhaps even more so I guess in the east, in the far east than maybe in the west.

One of the things we see often as well is that the mobile games in the west, they are still kind of soft kind of mid-core whereas the games on far east is now going much more focused on real-time gameplay, but we are slowly seeing that trend happening in the west as well. If you look at some of the top grossing games in 2017, I think in the west maybe 13 of the top 100 grossing games has got a real-time multiplier element to it. If you look at the absolute biggest game in 2017 is game by a Chinese company, Tencent, who you may or may not have heard of. They're quickly becoming the — If not already, the kind of mobile games in the world and they owned something like 90% of Supercell who in turn owns about 70% of Space Ape Games where I'm working. You see that family tree there.

Supercell, as you probably know, they are the creator for top grossing games in the west, games like Clash of Clans and Clash Royale. In Tencent, they have a number of massive hits that you may have heard about in the west, but they are doing fantastically well, and the top grossing game in 2017 is a 5v5, real-time multiplier game that Tencent has created. You can think of it as a League of Legends clone and they're doing some crazy numbers, something like over $400 million monthly revenue as recorded by, I think, Bloomberg, and something around 80 million daily active users. Those are pretty scary numbers, and they have only just moved to the west recently and have rebranded, they've done the UI, some of the styling has changed for the western audience. Whereas in the east, it's much more focused around fantasy theme and they've adapted for the western audience as well. Most of those numbers are coming mostly from China alone, which goes to show some of the potential that's available in this space.

[0:12:02.5] JM: 5v5 real-time competition is quite an engineering problem too. When you think about 10 people running around in virtual space fighting each other over an intermittent network connection, that's not an easy problem to solve, and the closest to analog in terms of the shows that I've done that comes to mind is Uber, where you have the car that you summon and you look on your phone on the map and the car is slowly climbing across the street and then you see it turn and then it like spins and like — Just trying to synchronize the real-time actions of a car and a person who has summoned the car, that is still not something that we could really do. When I think about trying to synchronize the actions of 10 independent people, it sounds pretty hard.

[0:13:09.8] YC: It's definitely a really interesting technical challenge for sure. As you said, I have that same problem the other day as I was leaving a conference. I was waiting for my Uber and I have no idea where the car actually is, because the update comes every — I don't know, once every minute it seems. Whereas for these kind of games, you're talking about you have to send10 to 20 inputs per second per player and you're super sensitive to any kind of LAG. From what we can see from our own test, anything beyond 300 millisecond round trip, so from you sending an input from your device to hitting a server and coming back to your device, anything over 300 millisecond, you can start to feel it.

In terms of engineering, it does have some really unique challenges in this particular space and it doesn't mean that you have to make decisions that you probably wouldn't have to in many other spaces. For example, it's quite common in this particular space to have to deploy your game servers all over the world so that you can have your infrastructure as close to your customers as your players as possible, but also making sure that when you match players together, they're in the same geographical location so that everyone gets the best experience possible and you don't have matches where you are winning just because your opponents is having the worst network latency than you are.

Also, we're trying to make our games so that they can perform reasonably well in countries with poor bandwidth and we have run alpha in both Netherlands for the game I'm working on in both Netherlands as well as Philippines. The different in the networking, both the speed as well as the bandwidth, is quite drastic, is quite different.

A lot of that, you have to factors those things into account when designing your protocols and making sure that you're using — You're being very efficient in terms of bandwidth but also your server have to very performant so that you spend as little time as possible on a server. Then there's still a whole bunch of tuning around the CCP layer and we're also implementing reliable UDP right now as well, and we can even more about different approaches available. None of the things we do right now is really, I guess, groundbreaking. It's all common sense. It's the things that people have done already in this space, but when you're trying to make them work on mobile, there are some additional challenges that you have to take to account.

[0:15:40.3] JM: Let's just start with just the idea of 10 players, or even just 2 players that are on different network connection speeds. If I'm on a very slow and intermittent network connection and I'm fighting against you and you're on an extremely good network connection, what kinds of conflicts can those create in a real-time game?

[0:16:07.3] YC: That would depend on the approach you're going with. For a lot of the traditional real-time strategy games, they use an approach what they call lockstep. Essentially, you have a peer-to-peer network between all the players in the match and the problem in that space is that if one person experience a LAG, then everybody experience a lag.

A common approach that you find with mobile games that are doing real-time multiplayer or strategy or just battle arena type of games, is that you may still be doing lockstep, but the lockstep happens via the server. So if one player experience a LAG, he's the only that experiences LAG in that match rather than impacting everybody in the match. Also, there's a significant tradeoff whether or not you want to have the server be the authoritative, so all the state changes happen on a server and a server broadcast state changes in deltas in the state to all the players, or you just have the players send inputs to each other and have the synchronization happen on the client's side so they minimize the amount of data they have get transferred.

Also, you do certain tricks with the client as well making it so that as you send the input to the server, you start to render some of the changes based on the assumption that what you send, receive form the server within a reasonable amount of time, so you apply a predictive model to the UI rendering so that the feedback is so much more immediate compared to what actually happens under the hood when data gets transferred over the network to the server and back and then you update the internal model on the client. Which also means that you need to have a determinism on the client side as well, which funny enough, with Unity 3D, the game engine, there's a well-known determinism problem when it comes to floating point calculations. So we also have to implement the fixed point maps ourselves as part of our game as well.

[0:18:07.7] JM: If you have people that are moving around on a map and then they're sending their movements to the server, the server is going to be authoritative about whether what circumstances those movements will resolve to, but the clients can make predictions about those movements. If I move from one point in the map to another point in the map that is open space, I'm just walking across grassy field and there's no enemies on the grassy field, my client side device, my phone, can make a prediction that the server is going to be completely fine with that movement. I'm moving across field, that's totally fine.

Even if the network connection drops off and my packet is received, but it takes a while for the server to send the next packet that defines the next set of states across this big map, everything is going to be okay, because my client side device can just say, "Okay. Well, we're just going to make a prediction about where you're going to move to," and you're going to move along grassy field and it's going to be totally fine. Where that gets problematic is let's say my client side device makes prediction that I'm moving along the grassy field and then it turns out that an enemy has been moving towards that area of the grassy field as well and the server — After my intermittent network connection reconnects, the server tells me, "Hey, you just encountered an enemy," but on my client side device, my client side device has just been predicting that, "Oh, you're just walking along a grassy field."

You can encounter these problems with the client side prediction conflicting with what the server calculates is the actual reality. So what are some approaches to avoiding those types of conflicts or resolving them as they occur?

[0:20:00.5] YC: The simplest approach will be to just limit how far the prediction model can take you. So the client side prediction model can deviate from the last known good state that's been confirmed in the server. So you allow the client side prediction model to maybe deviate by certain amount of actions before you get confirmation from the server.

One of the things you could do is — And this is something that we see a lot of similar mobile games do is once you get to the point where you are [inaudible 0:20:32.0] a bit too much to either slow down the action, so it looks your character is moving as fast or in some cases just grind everything to a stop and just put your head down and say, “Okay. I can’t predict anymore. I need to know from the server what the next action is,” and that can happen if you have intermittent connectivity issues from the server, so you have a spike in latency for example.

[0:20:58.2] JM: I see. To put a finer point on this, you define two different ways of modeling the client server relationship in a game. You have this continuous centralized model, the server authoritative model, and then you have this lockstep model. Could you walk us through these two models?

[0:21:20.5] YC: Sure. With the server authoritative model, the games that is only on a server and as the players move around, they move from one part of the world to another part of the world and maybe they will shoot at somebody. Those inputs are then sent to the server and there are consequences depending on how sophisticated the game is. It may just be taken at that moment in time when a server receives the input, but for some games such as Overwatch and a few other games, they would do something a bit more sophisticated. They will walk backwards work out when the — Based on latency, when the event actually happened on a device and then reapply some of the changes to the game state and then broadcast out on an interval all the status in the game states to all the connected players so that they can update things on their side.

This approach is easier in many sense, because you don’t have to worry about player cheating. It’s quite hard to cheat in this setting because a server controls all the state changes, and the server — At the same time, you end up sending a lot of more data to all of the players, which means on the client side you have — A, you have a better network connectivity and using a lot more bandwidth which in countries like Philippines and China is not as readily available or as compared to the west.

The lockstep approach will be, again, the client is sending input to the server and the server is buffering the map into frames of, say, 15 frames or 20 frames a second. I don’t know — Every frame, you send out all the buffered inputs from all the players to everybody so that as I move around on my device, my input is sent to the server, but maybe I don't get it back for another 60, 80 or 100 milliseconds, because it needs to do that round trip, but also you didn't take into account the time it takes to buffer everything up for one frame on a server. I only get my input as well as everybody else's input back. But because all the clients now receive all the inputs from everybody in a same sequence so they can apply those updates to their state, so you move frames onwards on a client side and all the clients would do the same thing to their internal model. So that's how they achieve consistency.

The problem with this approach is that now it's possible for people to cheat if they were able to hack the client or maybe reverse engineer the APK, for example, because at the end of the match, all the clients, the owner of the state, so they can say, "You know what? I won the match here," and then you have somehow report the result of the match to the server and the server would then have to validate those results that it collects from everybody.

[SPONSOR MESSAGE]

[0:24:29.8] JM: You are building a cloud-native application and you need to pick a cloud service provider. Maybe you're just starting out with a new app, but you have dreams of scaling into the next giant unicorn. Maybe your business had been using on-premise servers and you want to start moving some of your infrastructure to a secure cloud provider that you can trust. Maybe you're already in the cloud, but you want to go multi-cloud for added resilience.

IBM Cloud gives you all the tools you need to build cloud-native applications. Use IBM Cloud Container Service to easily manage the deployment of your Docker containers. For serverless applications, use IBM Cloud Functions for low-cost, event-driven scalability. If you like to work with a fully-managed platform as a service, IBM Cloud Foundry gives you a cloud operating system to control your distributed application.

IBM Cloud is built on top of open-source tools and it integrates with all the third-party services that you need to build, deploy and manage your application. To start building with AI, IoT, data and mobile services today, go to softwareengineeringdaily.com/ibm and get started with countless tutorials and SDKs. You can start building apps for free and try numerous cloud services with no time restrictions. Try it out at softwareengineeringdaily.com/ibm.

Thanks again to IBM for being a new sponsor. We really appreciate it.

[INTERVIEW]

[0:26:06.2] JM: Okay. I didn't quite understand how you could do cheating in the lockstep model. Maybe you can explain that a little bit later, but I guess I just want to make sure I understand this correctly. So the server authoritative model is more of a streaming model where at any given time, if I've got a client, I can send my state and my changes to the server and the server is going to respond to me with other updates that it's received from people, but in the lockstep model —

[0:26:42.1] YC: For the server authoritative model, you are sending your input but you're receiving from the server updates to the state for the entire game world. For example, I'd say, "Okay, for my character, I've moved to north, but when I get back would be my character's new position in the world as well as the position for every other character in the same match as well as maybe if the world is a dynamic world where volcanoes will fire or something else will happen, those state changes to the game world would also sent back to you from the server as well.

[0:27:18.8] JM: In the lockstep model, I can send updates to the server however frequently I want, but my received global state changes that involve everybody else's moves, those are going to be batched. Everybody's inputs get buffered and then they get calculated in some batch and then every N-milliseconds or whatever, there is a lockstep and the lockstep sends a multicast message to all the clients that are involved all of the updated state. Am I contrasting this correctly?

[0:27:58.1] YC: Both lockstep and the server authoritative evolves some batch and they evolve multicasting at a certain frame rate. The difference is whether or not a server is sending you game state changes or just inputs from everybody. The lockstep approach would be the server sent inputs for everybody rather than the game state changes, because a server doesn't have the game states. The clients are authoritative in this case of the game state.

[0:28:31.8] JM: Wow! Okay. In the lockstep model the client just gets deltas from the other clients essentially that are —

[0:28:40.4] YC: I move forward, I move backward, and those are inputs that I'm sending to the server and those are inputs I get back in the lockstep approach, but also get back from the server all inputs for everybody else. In this case, it'd be Jeff, his character moved forward, he's not aiming at a 40 degree angle.

[0:29:01.2] JM: The server authoritative model is — Now I understand the hacking problem and the lockstep model, because the clients are essentially — They're each individually responsible for deciding on the game state, which should be totally fine. If all of the clients send the moves that they have made throughout the game and then those moves get multicast by the server to the rest of the clients, there should be no issue there at all. They should be able to derive the same game state, because they each have the same set of moves that they have aggregated from each of the other clients, but the problem is that ultimately the calculating is done by the clients.

If I'm one client or even all five of the clients, if we're talking about this 5 by 5 game, and all of five of the clients somehow reverse engineer the APK, the software package, they could say, "Oh, you know what? Actually, we calculated that we won this game." The other team would be saying, "You know? We calculated. We won this game." You could have some sort of problem there, because the clients are deciding on the ultimate calculated end game state.

[0:30:16.5] YC: That's right, which is why you still then have to have a mechanism for doing server side validation based on inputs. Run the simulation on a server to have them to decide what is the authoritative answer for what's the end game state.

[0:30:33.2] JM: Okay. At Space Ape, the game that you're working on or the games that you're working on, you typically use the lockstep model?

[0:30:42.1] YC: We have two games that we've been working on that's a real-time multiplayer. The one that I'm working on right now is using the lockstep approach. Another game is using a slightly different variant where the simulation for the AI components. Imagine, if you play League of Legends, you have those creeps, right? You have those towers. They also have AI as well

In the lockstep scenario that I've described to you, every client will be running the simulation, the AI for those things, but there's also a slight variant whereby you have a master client that are doing the calculation for the AI, and he then would broadcast any changes to the AI state to everybody else. One of other game is using that variant which in terms of bandwidth and bandwidth use is slightly more expensive than the lockstep, but it's a lot better compared to server authoritative.

[0:31:46.9] JM: The tradeoff that we're talking about here is primarily in the bandwidth, because in the server authoritative model it sounds great because the server is resolving all of the moves of everybody and the server decides on a game state and then the server can just broadcast the game state to all the users, but the problem is that if you need to broadcast the entire game state to all the users, that is a lot of bandwidth. Whereas in lockstep, you can just broadcast essentially the diffs of each move and have the clients do the calculation, because have the clients do the calculation, you're going to be much less bandwidth constraint when you are just sending the change set of the moves.

[0:32:32.9] YC: That's right. We know for other companies that had built these kind of games, that your biggest cost when it comes to offering these kinds of games in scale is going to be bandwidth, but also impacts a user's experience especially in those countries where bandwidth is much more scarce compared if you're in the west. Another factor to consider is when a server side simulation can be really expensive, so you also have to run a lot more servers as well.

[0:33:06.8] JM: When we're talking about these mobile games in contrast to the classic multiplayer games, like Counter-Strike or Star Craft, these games are typically played over high bandwidth wired connections. Do they have a totally different model for the multiplayer different client and server resolutions because they have those lower bandwidth constraints?

[0:33:37.8] YC: Yes. I think so. I haven't looked at — We've done a lot of researched, looked into what other games do. League of Legends, Dota, Dota 2, Call of Duty, a number of other games that primarily play on a desktop, on Wi-Fi connections, they're all using server authoritative or some variants of that whether or not you're sending the whole state on every frame can optimize, so you just send deltas as well, but even those deltas, it's still going to be a lot more than the pure user inputs. Whereas where you look at mobile games, and I keep going to the same game by Tencent, because those guys, they have to work in the constraint being a game that's played on the [inaudible 0:34:20.1] in a country like China where bandwidth is something like number 70th in the world and have to make their work really well. Those guys that use the lockstep approach, which is one of the things that we looked at and see, "Okay. Do we want our games to have a global reach? Do you want to go to markets like China? Like South East Asia where we're going to have to think about this kind of problems." We made the decision quite early on that as much we think server authoritative is going to be a lot easier to implement in many ways given the sort of conditions that we want our app, our game to be played in [inaudible 0:34:58.1] with a lockstep approach.

[0:35:01.2] JM: What you said there with a game like League of Legends where it's primarily played maybe a T1 connection or some other kind of wired connection, even then, there's a gradient between server authoritative and lockstep, because you can take the different actions of all the users and you can find some overall change set to the game environment that takes into account all the individual changes from the users. Even then, you don't necessarily need to send the entire calculated overall game state over the wire. You're just sending some conglomeration of the different user's changes multi-casted to people and then you can still have the clients derive the new game state from that. I'm just saying that to emphasize the fact that this is a gradient between server authoritative and lockstep.

[0:36:05.2] YC: That's right. Another thing to consider is therefore games like League of Legends, you have more than just the player control characters. You have those NPCs, you have those towers. Those are also games states that need to get broadcasted in the server authoritative model. Even if you only send in the delta for individual states, those can still really increase your bandwidth use as your game becomes more complex.

Right now the game that we're working on is played in the static environment, but we also have the option to in the future make a game more dynamic, have the world be more destructible perhaps, maybe have other NPC that you can interact with. For those work and be able to scale those but still have them playable in a country like China, then those questions we need to think about ahead of time and that's also one of the things that [inaudible 0:37:01.5] into our mind when we decided to with the lockstep approach.

[0:37:06.0] JM: How does that affect game design itself? Because I can imagine if you know that you're doing this particular type of networking in the game interactions, if you're smart about it, you can design a better game that's going to be less — The interactions that take place in the game are going to be more — I hesitate to use the word resilient, because that's kind of overloaded in this conversation, but the conflicts are not going to be as painful for the people on this low bandwidth connections. How does that affect game design?

[0:37:42.7] YC: It hasn't affected game design too much apart from whether or not we will have the option to make the world dynamic, more expensive and introduce NPCs, because in —

[0:37:54.3] JM: Sorry. NPC is what? [inaudible 0:37:56.8].

[0:37:57.7] YC: Non-playable characters.

[0:38:00.0] JM: Got it. Bots.

[0:38:01.9] YC: Yes. If you play League of Legends, you have those parts that get spawned and those are characters that you can also fight and also fight with you as well.

[0:38:11.5] JM: These are expensive, because the server is essentially running these bots, and the more dynamism you have that is shared state between people, the more you're going to have to send over the network.

[0:38:25.2] YC: That's right.

[0:38:26.9] JM: Fascinating.

[0:38:27.8] YC: Even using the server authoritative approach and trying to make it work on mobile, then that would put a constraint in terms of how dynamic you can make the world.

Whereas with the lockstep approach where you only send the input, then you can make the world as interesting, as dynamic as you want so long that the simulation doesn't cause your phone to [inaudible 0:38:47.7] because it's doing too much rendering.

[0:38:51.0] JM: Okay. Let's talk about some other implementation details. One thing you emphasized in a talk that I saw you give was that you want to deploy these kinds of games to multiple geographic locations, because if you have a game like 5 by — It's a 5 by 5, 5 on 5 war game, all you need is 10 people. You don't actually need the entire network.

Something like Facebook, you need a global synchronized deployment because if somebody in China comments on a post that I made, we want that update to happen aggressively. But if you're talking about a game where you just have 10 people that are engaged in a real-time interaction, ideally you would just want all 10 of those people in a geographically local environment. Like you want to have all 10 people in — If you've got 10 people in the United States that log on to play and you've got 10 people in China that log on to play, the ideal world is you've got a server in China that has the server that's hosting the battle between those 10 people in China and then another one in the U.S. that hosts those battles between the 10 people in the U.S.

Why don't you contrast the deployment of a globally synchronized application like Facebook with a locally synchronized application, like a multiplayer game?

[0:40:20.2] YC: Yeah. You're absolutely right. That's the design goal that we have as well that you have a server, the real-time multiplayer server deployed geographically close to your user. Like you said, if 10 people are playing in China, if you'll all be playing real-time against the server that's hosted in China or very close to China and the 10 people that play in the U.S. would be playing on a server that's closest to them. All of that should happen automatically without the players having to do anything.

When going to the game you say, "I want to start a match." Automatically you should be match-made against players that are in the same region as you so that everyone is playing against the server closest to them. That means when it comes to deployment, we have infrastructure that's closest to the players all of the world, but we only need to that with the real-time servers [inaudible 0:41:14.6] deal a lot of other infrastructure around managing your game, your player profile, managing purchases and all these other things. Those can still be centralized in one region. You can still save some of the operational overhead in terms of having a globally deployed infrastructure.

There's a couple of things in terms of doing has impact on game design. For example, if you have leader boards and you have alliances, guilds and you have chat rooms. It brings about the question of should those things be global or regional. If people are chatting the game, should they all going to be playing in different parts of the world using different languages. Does it make sense for chat rooms to be global or more regional? At the same time, if you're going to have alliances and guilds and you want to encourage people to start matches with their friends in the guild and you still want them to be playing in a same geographical location, it doesn't mean that those game features should be global or should they be region [inaudible 0:42:20.4] some of those locality that you have in terms of the multiplayer rules as well.

The whole bunch of different things that comes out once you have the split of — Some parts infrastructure being global and some parts of it being regional to optimize what player experience.

[0:42:41.2] JM: The last show that I did with you is really fun, because you were so good at describing the infrastructure for the social network, Yubl, that you were working on. You talked about a lot of different managed services. The shows that I've been hosting have been increasingly about how do you fit together these managed services often times, because like I did a show recently with Thumbtack, which is a marketplace for home services and they have so many challenges in terms of how do you create a marketplace that works effectively.

The business is pretty good because they have network effects and whatnot, so they would rather not have to think like a software company. They don't want to have to think about infrastructure and uptime and scalability. They would rather purchase managed services that take care of all of that.

With a game — If you can get greedy in a game, you're going to end up paying very expensive infrastructure cost. You talked about the tradeoffs of build versus buy in this talk that you gave about scalable multiplayer games, so I think it's more of a nuanced conversation if we're talking about a game than a marketplace. I say all that to preface. Why don't you just give an overview of the infrastructure of the game that you most recently worked on?

[0:44:09.5] YC: The game I'm working on right now a multiplayer game. I can't really talk too much about the game itself as that is still under development. In terms of infrastructure, as I mentioned already, we have real-time game servers. They are deployed to multiple regions so that we have infrastructure globally for that. The rest of our infrastructure are ran out of the same — One region in [inaudible 0:44:31.8] where the player profiles are stored, where all of our databases, all a bunch of others services are all hosted out of there.

In terms of the build versus buy, you have several companies that are still operating in this space. The biggest one is probably [inaudible 0:44:52.0] games. They have a product called Photon which is essentially what we are building, but they offer it as a service and it gives you many of the things that are building already in terms of having the multi-region support.

They also have a lobby system out of the box as well, so for things like matchmaking. They've got quite a flexible system for you to do that. Also, they have been around for something like 10, 15 years, so they've had a lot of time to automize the networking stack and they can run on their own soft layer infrastructure where it's all real brick and mortar hardware so that you get a much better networking, I guess more consistent networking performance maybe then the cloud-hosted servers that we are using.

At the same time there's also a downsize for using a hosted — I guess a product like Photon. Based on the pricing model they have, they can be quite expensive especially once you scale. The way Photon works is that you pay for a provision, the peak monthly connected — The concurring connected user. So if you have 100,000 people connected at peak in the month, that's what you're paying for. You go back to those — I guess that brings you back to the capex versus opex discussion that people have at the start of the cloud. We don't have flexible model of pay as you use, and you can't just go over those peak provisioned usage level as well, but when you do that, they charge you quite a bit of extra.

I guess for us as a company also, as we're building more and more games that has got a real-time multiplayer element to it as well, this question took us a very long time to decide whether or not we should continue to build something in-house or to just use Photon especially as we already used Photon for some of our prototype games and there's quite a bit of experience in the company and we do really love what the [inaudible 0:46:57.2] games have done with the Photon product.

As a company if we're going to be banking the future of the company on all these real-time multiplayer games, we feel that this element of risk, this element of [inaudible 0:47:11.0] exposure that we have by owning some of the core stack that we have.

[SPONSOR MESSAGE]

[0:47:23.0] JM: Do you have a product that is sold to software engineers? Are you looking to hire software engineers? Become a sponsor of Software Engineering Daily and support the show while getting your company into the ears of 24,000 developers around the world. Developers listen to Software Engineering Daily to find out about the latest strategies and tools for building software. Send me an email to find out more, jeff@softwareengineeringdaily.com.

The sponsors of Software Engineering Daily make this show possible, and I have enjoyed advertising for some of the brands that I personally love using in my software projects. If you're curious about becoming a sponsor, send me an email, or email your marketing director and tell them that they should send me an email, jeff@softwareengineeringdaily.com.

Thanks as always for listening and supporting the show, and let's get on with the show.

[INTERVIEW]

[0:48:24.8] JM: Photon is some platform as a service that's kind of expensive.

[0:48:31.6] YC: But it's really good.

[0:48:32.7] JM: But it's really good.

[0:48:33.2] YC: it's really good. Yeah.

[0:48:34.9] JM: That's pretty interesting. All these stuff that Photon takes care of for you, if you wanted to do that yourself, you would have to rewrite like a lobby system and it sounds like a lot of networking stuff that's difficult, and then also their networking stack is on soft layer. I think with soft layer, the advantage of soft layer is like you get directly deployed to bare metal rather than on AWS where you're deployed to virtual machines that are on bare metal. Is that right?

[0:49:10.0] YC: That's right. The downside with using bare metal machines is that — That's why Photon has that pricing model of peak [inaudible 0:49:19.2] level is how long it takes for them to provision additional hardware.

Also on the other hand, the networking on Amazon web services has been gradually improving. Just recently, they announced support for the new C5 class of instances and [inaudible 0:49:38.3] additional networking conversation that's done to the instances type. I think in this case it allows you to [inaudible 0:49:47.6] networking to get the real networking rather than a virtualized network. So we have tested it on other instance types that has got a same support, the same optimization and it does make a big difference in terms of both the latency as well as how much variants you see as well.

Networking on Amazon is getting better, because for the multiplayer servers, we're just going to run it on VM. We're not dependent on other managed services, so it means that we can also use Google cloud as well, which has got more expensive networking, but also from what we can gather, it's also a better networking as well.

It's interesting, Google has literally gone the other way where now they're offering cheaper, but worst networking capability to the VMs, whereas Amazon is trying to improve their networking offering with the EC2.

[0:50:44.7] JM: Just to clarify, you have some of these game workflow on — What was it called? Not electron. What was the name of the service?

[0:50:56.1] YC: Photon.

[0:50:56.3] JM: Photon. Right. Okay, so you've got some of your system on Photon, which is a platform as a service for game stuff, and then —

[0:51:04.2] YC: We have some games using Photon and during the early prototype development, we were using Photon but we made sure that — For the game that I'm working on, the networking layer is separate from the game layer so that we're able to have the same game run on both our own stack as well as on Photon's. That's what allows us to run A/B test when we went to Netherlands in the Philippines so that we can compare our implementation with Photon side-by-side to gauge how good our implementation is.

I guess one of the big reasons for us to decide to build our own is because we feel we have enough expertise in the company to take on this type of engineering task, which if you don't have, and many of the companies that we know that work in this space, they may not have the same expertise, then Photon absolutely makes sense.

[0:51:59.2] JM: This networking stack that we're talking about, this is the conversation that we were having at the beginning of the show. If you're using Photon, you get some API into a lockstep networking model?

[0:52:13.0] YC: You summon those protocol yourself. With Photon [inaudible 0:52:16.9] networking layer in terms of TCP and the reliable UDP, but also allows you to write some code that runs on their server. So you can [inaudible 0:52:27.9] both the server authoritative as well as lockstep between your custom server code on Photon as well as your client code. Photon is [inaudible 0:52:38.2] is a great product and is very, very flexible.

[0:52:42.2] JM: Sure. No, I mean this is like — I talk about — I'm always talking Heroku, because Heroku is a sponsor of the show, but also I use Heroku all the time and I just love it. But there are certain products for which Heroku would just make zero-sense. If you're like a stock trading company, it probably would not make any sense to be deployed to Heroku, because we have such high bandwidth requirements and just — Also, it just is going to eat into your margin eventually and you just want to be — I don't know. I don't even know what I'm talking about.

Basically, this is just a discussion that the build versus buy of platform as a service versus infrastructure as a service is a conversation that many companies are having with themselves. When you're talking about this networking logic that Photon takes care of for you, that you are now implementing. I know in your talk you discussed the actor model. Maybe we can go into that. But what exactly are they taking care of for you? Because you said you have to deploy yourself written code that manages like what is going on in the server. How the lockstep stuff is going, but what exactly are they doing in terms of networking?

[0:54:02.8] YC: They [inaudible 0:54:04.6] of having deployments in multi-regions so that they can have infrastructure that's deployed to different datacenters all around the world. You get to alter the bugs and they also have a lobby system that get after the bugs. You do still have to write a lot of custom code in terms of the communication between the client and the server, but the client library [inaudible 0:54:24.7] you can alter the bugs, you can choose to say use a TCP, use our UDP [inaudible 0:54:30.0] UDP and all of those things that you have to implement yourself otherwise at a protocol level. So they give you a lot of the out the box.

[0:54:39.9] JM: When you say implement something at the protocol level, what does that mean?

[0:54:43.6] YC: Take our TCP implementation as example. We're not just making HTTP request with JSON. All that protocol, all that is happening with a custom protocol so that we can package all the [inaudible 0:54:56.5] as close as possible so that we minimize that amount of bandwidth you're using, but also managing the state so that you have the concept of a match. How many players and what the current state for those players? Photon makes a lot of those very simple for you, and you can plug in your custom code that you can do additional things.

I'm not familiar with Photon, but from my understanding, all of that becomes a lot simpler with Photon and you don't have to write a lot of the custom networking code that I've had to write myself. Things that our managing connections reconnect and matching connection to player to a match and all of that.

[0:55:39.2] JM: Is this why — I saw another presentation you gave recently about protocol buffers. Did you have to write your own proto buff interface to describe the communication between the client and the server for the sake of not being on Photon? Does Photon take care of the over-the-wire object representation for you?

[0:56:01.5] YC: They can do, but in our case because we only that layer that you can [inaudible 0:56:06.9] the Photon client with our own client, talk to our networking stack and having to write some of those ourselves in [inaudible 0:56:15.0] anyway just so that we have that portability. In our case, we are not just sending a proto buffer because proto buff can also have those tags as well which means that you're sending a bit more data [inaudible 0:56:30.3]. So we are using a custom — A protocol to communicate between the client and server.

[0:56:40.1] JM: Interesting. It sounds like Photon is really quite flexible and you can basically go as low level as you want to, but if you want them to take care of things like over-the-wire object representation, they will figure that out for you.

[0:56:56.3] YC: Yeah. That's my understanding as well.

[0:56:59.0] JM: Okay. All right. Well, we won't speculate any further on Photon. Tell me more about your infrastructure? What do you use for the database, for example? If you've got this real-time state management that needs to be reconciled in — I'm sorry. Well, you have some servers, some games that are doing the server authoritative model and at least these servers have to do the real-time state management. What's your durable store?

[0:57:29.9] YC: For the match itself, [inaudible 0:57:32.0] are stored inside, in this case, in our case, an actor, in the actor. We don't save the state in the database doing the match. We do stream it out via Kinesis so that we can then, in the background, have a Lambda function that will consume those data, consume those stream of inputs from different matches per system to S3 for the purpose of allowing you to spectate with a small delay or watch a match after the fact. Because once data is in S3, it means that you can expose them by [inaudible 0:58:10.5] so that you don't have this problem of, "Okay. What if you've made the game — The game becomes popular. You want to make into an e-sport and you want people to be able to watch other famous player right there in the app. How do you facilitate that in a way that doesn't affect the match experience itself? That's how we are building all of that.

In terms of databases, just a few different piece of stack to keep track of in terms of this data [inaudible 0:58:38.3] match, how many, who's in them, all of those which are just right now stored in DynamoDB. Then we have a bunch of background stream processing. I'm a big fan of the whole [inaudible 0:58:49.0] model.

[0:58:51.0] JM: I know you are.

[0:58:53.2] YC: It certainly makes life a lot easier. From the point of view of the multiplayer server, it just broadcasts a number of different events [inaudible 0:59:03.2] that a match has started, this player has joined, here's the seconds' worth of input I've batched up and that I can other systems, Lambda functions often, to consume those events.

For example, we have a feature whereby when I start a match, if I'm inside a guild, other people in my guild would get a notification via our inside app that says, "Oh, Yan started a match. Do you want to join him?" They can see button to join me right next to my name or they click another button to watch the match that I'm currently playing in. All of those are coming out of one USEs 1 region.

[0:59:47.7] JM: Okay. That's awesome. You've got all these. Individual clients are doing stuff. Their state gets aggregated on the central server in some form or fashion and you're going to either send it out — You're going to either stream it out to them or you're going to send out updates in lockstep. We already discussed to depth, so maybe that's not so interesting at this point. But what is interesting is what you just mentioned where you've also got Kinesis that is buffering the state of the centralized server and being written out to S3, and the S3 is like a hosting service for spectators.

Now, you've got these different game frames and at the most granular level, I could imagine you could write out every single frame to S3 and then the spectators would be just consuming frames over S3. I imagine it is much more efficient to get those frames and either batch them at the actor level and put them on Kinesis in a certain batch or send every frame to Kinesis and then batch them in chunks off of Kinesis on to S3. You've got some different tuning decisions you could make there. How do you evaluate that batching of the rights to S3?

[1:01:12.5] YC: We actually do both batching on the actor level. So the actor would send inputs. You batch them, you send them out into the actual players in the match at 15 frames a second or whatever, but it only sends those inputs to Kinesis for, I think, every three seconds or so so that you get batching at that layer. Then when the Lambda function receive the input, and there's a secondary batching there that happens before you write to S3. Then on the clients, when you want to spectate and watch a live match, there's also buffering happening on that side as well so that the client will get the first block, which contains a bunch of frames, and then you'd get the next block, next block and so on. The client also gets all the frames available and then catches up to the [inaudible 1:02:00.4] it has and then you start to play the match with a few seconds behind so that as a watching player, even watching a live, try to watch a live match, I'll be watching actions maybe a few seconds behind the real action happening. At the same time you can imagine once you have all the frames for the same match, then people can just go back in time and watch other matches that I have played or you have played.

[1:02:27.5] JM: Let's zoom out. There are so much that we didn't cover technically-wise in this show, but people can check out the talk that you gave which I'll put in the show notes. Just to talk a little bit about what is like to work at a game company. I know nothing about that. I have done zero coverage of it. What it's like to work at a games company that makes these kinds of multiplayer games?

My understanding is it's kind of a hits business, where if you have a hit, it's really exciting and it's really difficult and it's really challenging and it's really fun. If you have a non-hit, it falls flat and it's kind of depressing and many people in the games industry goes through both of these things. What's it like?

[1:03:13.6] YC: You're obviously spot on when it comes to it being a hit-driven industry. It's very much that, and if you'd look at companies like Zynga, companies like King, Machine Zone, they've all become massive, massive company off the back of one hit, but they've all had many, many games in the past that hasn't been that number one hit, and all you need is that one game to go from where we are as Space Ape to become a billion dollar company.

Working for games company, one of the things that I guess you find as quite different for many other companies is that, A; typically you find a very flat hierarchy and you find the teams that are — Very often you find a company that are building one product, so that whole company is geared towards optimizing for their one product. For games companies, you have many games. One of the things that you comes up often for us is how do we have that drive for new ideas, but at the same time trying to focus our attention on the great idea that we think can become a top grossing game.

Over the last 8 to 12 months we have created these, I guess, now more that's formal funnel where we have monthly game jams or hackathons that we run that allow us that people would go to a game jams, they make new ideas, and that feeds into the top end of the funnel. Usually based on some hack prophecies, that maybe there's a category, there's a [inaudible 1:04:49.1], there's underserved, but there's a massive market out there. Then we create — Going from there, maybe over the course of two or three game jams you take their initial idea, you polish it up to the point where, "Okay. We have something that is playable that you can showcase to other people in the company." As you build that more and more confidence in the company, that, "Hey, maybe this is something we can pursue. Then slowly we start to form teams around those ideas and we let them go wild and prototype and try out different idea, different game mechanics and whatnot. Then you try to start to size the market, the test against the market, do various tricks to see whether how much interest thereof with maybe the theme or maybe the genre, and then we keep going from there to making company [inaudible 1:05:40.0] or that you have a one day or two days and everyone just take part in the massive tournament inside the company to play your game, give you feedbacks and at the same time you can collect the metric to see how well, how engaged are people. Then you go to alpha and beta to get those external validation.

Right now, of this whole funnel, we have one game that's in beta and the game I'm working on right now is heading towards this second alpha built earlier next year. At the top end we probably had at this point 80 to 90 games, different ideas. One of the great things, one of the things that really stand both Supercell and Space Ape games out form the competitor is that when you have ideas, it's not the CEO and you have people that form those prototype teams to work on those ideas, and it's the team that decides what ideas to cure, it's not the CEO, it's not the game designer, but it's the team itself. The team has been responsible to say, "Okay, we've —"

Actually, this happened recently for one of the teams that's been working on this idea for maybe 7 months and they eventually they just sat down and say , "Well, we tried so many things, but all these ideas we've tried, we can't see doing better than what's on the market already. We're going to disband the team and go back into the company to work on various different things.

That's something that the Space Ape does really as well is in terms of recognizing that you need to align personal interest with the company's vision, company's best interest. One of the things that I often ask employees when I was doing [inaudible 1:07:26.4] interviews after [inaudible 1:07:27.7] went under is, "How do you do that? How do you make sure that people's best interest is aligned with the company?"

I think this is a problem that often the management don't recognize that everyone will just tell me, "Oh, it's not a problem when we make those personal objectives 12, 6 months ahead of time. We'll make sure that everyone is aligned with the company's overall goal." That's a very long time.

If everyone has got their own — At the end of the six months or 12 months cycle your performance review and you suddenly reveal a space on how you measure to those personal objective that is set by your manager 6 months to 12 months earlier, how do then deal with the fact what you're working on may not be in the best interest of the company or if you have to collaborate with another team and what the other team is working on is more important for the company's future, but if you help those guys out, now when it comes the end of year review, you'll be penalized for your performance scores as well [inaudible 1:08:35.5]. Depending on a company, you may even be [inaudible 1:08:38.9] if you're working for companies that are directly coded to the bottom 25% of the employees based on their performance score.

[1:08:50.7] JM: You bring up such a great point, and like how do you build a company where creative destruction can happen is a really important question to answer, because developers are — They want to be artistic. They want to have — I think everybody, everybody who is a — The type of people that you want to hire at a company are creative go getters and they want to take ownership over a product. They want to do creative stuff with it. Unfortunately, that can sometimes conflict with, “Let’s get the business done.” “Let’s debug the thing that is really not fun, it’s really annoying, but it is debugging our cash cow which we need to do.” There are some tradeoffs that can occur there.

If you’re an employee who only tries to launch new stuff and only tries to do creative stuff, and especially if none of those things take off, if you do that for a year and then during your performance review they look into like, “Hey, did you actually do anything? Did you actually debug anything?” Then it’s problematic. Of course, if you just have your employees debugging stuff, then it’s equally problematic. At a company you need to figure out this mix of creative destruction and getting the business done.

[1:10:14.4] YC: Yeah.

[1:10:15.5] JM: Anyway, Yan, it’s a been another fantastic episode. I love talking to you. The time flew by, so it’s always a good sign. I’m sure we’ll do it again in the future, and I’m really happy that you’ve landed at a place where you seem very gratified.

[1:10:31.1] YC: Yes. It wasn’t easy. The whole interview process was pretty — Well, let’s just say I’m not looking forward to doing all these interviews again soon.

[1:10:42.6] JM: Okay. All right. Hopefully you look forward to the Software Engineering Daily interviews a little more.

[1:10:48.5] YC: Oh, yeah. These are good.

[1:10:49.3] JM: Okay. Great, Yan. Thanks a lot.

[END OF INTERVIEW]

[1:10:53.7] JM: Who do you use for log management? I want to tell you about Scalyr, the first purpose-built log management tool on the market. Most tools on the market utilize text indexing search, and this is great for indexing a book, for example, but if you want to search logs at scale, fast, it breaks down.

Scyler built their own database from scratch and the system is fast. Most of the searches take less than a second. In fact, 99% of the queries execute in less than a second. That's why companies like OkCupid and Giffy and Career Builder use Scaler to build their log management systems. You can try it today free for 90 days if you go to the promo URL, which is softwareengineeringdaily.com/scalyr, S-C-A-L-Y-R. That's softwareengineeringdaily.com/scalyr.

Scalyr was built by one of the founders of Writely, which is the company that become Google Docs, and if you know anything about Google Docs' history, it was quite transformational when the product came out. This was a consumer-grade UI product that solved many distributed systems problems and had great scalability, which is why it turned into Google Docs, and so the founder of Writely is now turning his focus to log management and it has the consumer-grade UI. It has the scalability that you would expect from somebody who built Google Docs, and you can use Scalyr to monitor key metrics. You can use it to trigger alerts. It's got integration with PagerDuty and it's really easy to use. It's really lightning fast, and you can get a free 90-day trial by signing up at softwareengineeringdaily.com/S-C-A-L-Y-R, softwareengineeringdaily.com/scalyr.

I really recommend it trying it out. I've heard from multiple companies on the show that they use Scalyr and it's been a real differentiator for them. Check out Scalyr, and thanks to Scalyr for being a new sponsor of Software Engineering Daily.

[END]