EPISODE 621

[INTRODUCTION]

[0:00:00.3] JM: Napster, Kazaa and BitTorrent are peer-to-peer file sharing systems. In these P to P systems, nodes need to find each other. Users need to be able to search for files that exist across the system. P to P systems are decentralized, so these routing problems must be solved without a centralized service routing traffic. Without these centralized service that has all the information in one place, how can you solve these problems of node discovery and file lookup?

This is the central question that Petar Maymounkov sought to answer with Kademlia. Kademlia is a peer-to-peer distributed hash table. Kademlia implements the put and get operations of an efficiently scalable has table without using any centralized service. Each node in the system maintains its own routing table. When a user queries the system, which is the get operation, that query is serviced by the nodes coordinating with each other to intelligently route the user to their target location. When a file is stores, this is a put operation, the update to the file system can propagate through the network in a decentralized, uncoordinated way.

Petar joins the show to give a brief history of P to P networks and explain why he created Kademlia. He also explains what he's working on today.

We are hiring for Software Engineering Daily. The jobs include writers, researchers and a videographer and several other jobs. You could find them at softwareengineeringdaily.com/jobs. Some of these are part-time, some are fulltime. If you're interested in working with us, then check it out, softwareengineeringdaily.com/jobs, and you can also post jobs on our job board if you're hiring people and want to get at our Software Engineering Daily audience. It's easy and it's free. You can just go to softwareengineeringdaily.com/jobs and it's quite straightforward on how to post a job.

[SPONSOR MESSAGE]

[0:02:09.8] JM: Citus Data can scale your PostgreS database horizontally. For many of you, your PostgreS database is the heart of your application. You chose PostgreS because you trust it. After all, PostgreS is battle tested, trustworthy database software, but are you spending more and more time dealing with scalability issues? Citus distributes your data and your queries across multiple nodes. Are your queries getting slow? Citus can parallelize your SQL queries across multiple nodes dramatically speeding them up and giving you much lower latency.

Are you worried about hitting the limits of single node PostgreS and not being able to grow your app or having to spend your time on database infrastructure instead of creating new features for you application? Available as open source as a database as a service and as enterprise software, Citus makes it simple to shard PostgreS. Go to citusdata.com/sedaily to learn more about how Citus transforms PostgreS into a distributed database. That's citusdata.com/sedaily, citusdata.com/sedaily.

Get back the time that you're spending on database operations. Companies like Algolia, Prosperworks and Cisco are all using Citus so they no longer have to worry about scaling their database. Try it yourself at citusdata.com/sedaily. That's citusdata.com/sedaily. Thank you to Citus Data for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:03:54.9] JM: Petar Maymounkov is the co-creator of Kademlia. Petar, welcome to Software Engineering Daily.

[0:04:01.0] PM: Thank you for having me, Jeff.

[0:04:02.3] JM: We’re talking about Kademlia today, and Kademlia is a core component of peer-to-peer networking. That’s the application that Kademlia is most commonly associated with. What is a peer-to-peer network?

[0:04:17.1] PM: Okay. So the way I like to look at it is in the form of a distinction between peer-to-peer software in general and what you otherwise known as cloud software. I’m using these familiar terms, but the distinction is sort of more general and it’s the following. Peer-to-peer software, regardless of what it is, is always comprised of multiple computers, so multiple computing entities which all around the exact same algorithm at the start. A good metaphor to remember is to think of a colony of ants where all worker ants have the exact same DNA and, of course, doing different roles, but they all are equal at the start.

Contrast is with cloud software, which is also called distributed software or distributed algorithms where, again, you have like a collection of computing devices, but every single one is is running different software has a different purpose, and importantly there is hierarchical controls structure implied.

Is a good example of this metaphoric thing is a human shepherd is controlling a dog which is controlling the herd of sheep. So you have different species with different DNAs. So they have completely different specializations and there is also, as part of their specialization, there is an implied hierarchical controlled structure with an entity on top.

In the cloud software, the human happens to be the coherent kind of business workflow that the company wants to implement in the data center and the hierarchical relationship is that, usually, this is all encoded at some high level orchestration software, which runs other pieces of software, which do like specific data processing task and so forth. So you have the two worlds of multiple species controlling each other in an hierarchical way versus identical species that participate kind of democratically, everybody on their own, but they follow the same algorithm, so they end up doing something useful as a whole.

[0:06:29.9] JM: Right. So peer-to-peer networks are more of a flat structure, at least from the beginning.

[0:06:37.1] PM: From the start. Yes.

[0:06:38.4] JM: From the start. Right. Then it may develop certain structures where nodes in the network play a bigger – Some nodes play a bigger role in the network than others, but foundationally it's a flat structure.

[0:06:51.9] PM: Right. It's not known who will become an important player, because it depends on how they interact with the environment. Whereas with cloud software, it's set up front, which is the master and which are the workers, basically, machines.

[0:07:08.7] JM: Yeah. In a cloud software, you might have an application where there's four different services and then you've got a database service, and the database service services requests from all four of the other services, and so the database service is probably a priori, a much bigger player in the network.

[0:07:29.9] PM: Actually, no. It's a good thing you gave this example. What I mean here is the layers of control if you look at the company holistically. So the services that you mentioned in your example as well as the database are all applications that are running inside a cluster, and the cluster itself is managed by an orchestrating software which starts them in the first place. You see? This is a higher level of control.

[0:07:59.8] JM: I see.

[0:08:00.4] PM: So the point is that – So maybe from an application point of view, the database might be more important, but if you really look at like all the way the entire software stack all the way to the human operators, it's called a stack for a reason. The stack is the hierarchy of control basically.

[0:08:18.8] JM: It's more about the central planning from that perspective.

[0:08:22.0] PM: Right. No, you're exactly right. The simple point here is that central planning always creates a hierarchy of control because that's just the nature of it. So that's a good way of putting it. Yeah.

[0:08:33.6] JM: Right. The earliest peer-to-peer systems that reach widespread application that I know of or things like Napster, Kazaa, Limewire, these distributed file sharing systems. And I'd like to talk through these as an example of an early application of your distributed hash table technology, Kademlia, and I want to start with the technologies that came before Kademlia, which is that you had peer-to-peer systems that were built with a centralized database, like Napster, and then you had peer-to-peer systems that were built with a technique called flooding.

Explain the peer-to-peer technologies, the peer-to-peer stacks that were used for this role of the distributed file sharing network that came before Kademlia. What were the pros and cons of those systems?

[0:09:31.8] PM: Right. The first one was Napster, and this was simply the model where you put the whole database of songs and the locations of their files. You put it in one computer and all of the clients, all the users would access this one computer as a database. The drawback of this at the time actually was more legal than computational.

I mean, nowadays, someone might say this doesn't scale, but actually back at the time the real problem was that a subpoena was sent to the address of the Napster company and a subpoena suffices to stop the service until further resolution. So actually at the time I was thinking that the subpoena process – So the creation of peer-to-peer systems, at least the reason I created Kademlia initially was both to create scale, but also to essentially make the subpoena process language applicable, because if you had a service that – Or applicable in a very difficult way. So if you had a service that was sprawled across a thousand homes, especially across different legal boundaries, it would be very difficult to subpoena all of those. So at the time, really, it was more about resilience towards kind of legal boundaries to make systems like this available in multiple regions basically.

[0:11:02.8] JM: Yeah. So the next technique was this flooding, which I think is the multicast kind of methodology. Can you explain what flooding is?

[0:11:12.7] PM: Yeah. So I should back up to say that the main computer – Sort of the main sort of computational benefit of talking about this peer-to-peer system is flooding or DHTs, is after all indeed that you get to have much more space as you're using more computational devices. You have a replication, so some pieces might be available in multiple geographic regions. But the big point is that you get all these new space.

You can describe many more songs and sort of save them in this communal memory, which is much bigger than just one computer's memory. Now, the issue that comes with storing information though is how do you retrieve when you're looking for something specific, and this is when the distinction comes between flooding or other methods.

I would say that, first of all, all peer-to-peer technology is generally disperse the information of the database that they are memorizing. They disperse it evenly across its notes, and the difference is have to do with how do they kind of market target so that they can find pieces efficiently on a query. Because they are simply key value stores, in other words they just save some data under a given string, usually the query simply find the data associated with a given string. So how the filing is done differentiates flooding from other algorithms.

Flooding is the simplest algorithm. It's the first one that a peer – In a peer-to-peer system. Flooding simply just means that every node in the peer-to-peer system would ask every other nodes that they're connected to and this would continue until it floods the whole networks and collects all possible answers basically.

[0:13:04.6] JM: So the commonality between Napster, which was the centralized database, and flooding, which is the – I guess there's no centralized database, but you have to ask everybody. So every query is – You have to query all N nodes in the system potentially or something on the order of N. But in each of these systems, we've got the same basic setup, which I didn't really discuss. But you have a bunch of MP3 files, for example, and MP3 files are kind of big. So you can break them up into chunks and allocate them across the network.

If I've got a little bit of extra storage on my computer, maybe I store a chunk of an MP3. If you've got a slightly bigger set of space, maybe you can store a larger chunk or a multiple sets of chunks. The real difficulty is how do you have efficient lookups of those chunks. How do you retrieve those chunks efficiently and have those chunks addressed in, hopefully, a distributed way so that the network is resilient to any particular node going offline and the queries are faster. But underlying at the storage system is kind of the same for the different peer-to-peer network technologies that we're talking about. Is that right? Do I have a correct understanding?

[0:14:22.3] PM: Yeah. The setup is the following. Actually, the actual songs are present only on computers that people actually care to have them. I'm ignoring optimizations just to see the clean module. Songs are where they're being used by somebody. So on some subset of users for any given song.

Now, if a new users wants to find a new song and download it, they make a query that contains the name of the song, and what they're trying to get in return is the list of people who currently have the song. Really, they're looking up in this metastable where under the song name, you want to have a list of every person that currently has the song on their computer.

Now, what you described about, chunks, this is an additional layer of optimization which kind of goes beyond – So this is completely separate. It just is the notion that the files themselves might be separated into smaller pieces and instead of looking up, different algorithms might also hash them on additional nodes that are not necessarily listening to the songs.

Then the only modification happens in your algorithm is that when you look up a song, you're not for the files, for the people who have the files and area listening to them. You're asking a slightly more technical question, "Where are all the people that, for any reason, might have a piece of the file?" But this is an optimization having to do with efficient kind of storage.

[SPONSOR MESSAGE]

[0:16:06.5] JM: In today's fast-paced world, you have to be able to build the skills that you need when you need them. With Pluralsight's learning platform, you can level up your skills in cutting edge technology, like machine learning, cloud infrastructure, mobile development, dev ops and blockchain. Find out where your skills stand with Pluralsight IQ and then jump into expert-led courses organized into curated learning paths.

Pluralsight is a personalized learning experience that helps you keep pace. So get ahead by visiting pluralsight.com/sedaily for a free 10-day trial. If you're a leading team, discover how your organization can move faster with plans for enterprises. Pluralsight has helped thousands of organizations innovate, including Adobe, AT&T, VMWare and Tableau.

Go to pluralsight.com/sedaily to get a free 10-day trial and dive into the platform. When you sign up, you also would get 50% off of your first month, and if you want to commit, you can get $50 off an annual subscription. Get access to all three, the 10-day free trial, 50% off of your first month and $50 off a yearly subscription at pluralsight.com/sedaily.

Thank you to Pluralsight for being a new sponsor of Software Engineering Daily, and to check it out while supporting Software Engineering Daily, go to pluralsight.com/sedaily.

[INTERVIEW]

[0:17:45.5] JM: We've outlined the problem that we're trying to solve. Whether we are trying to address specific MP3 files or we're trying to address chunks of files across a distributed storage network, we're just trying to address something. We're trying to build a system where if I want to find a specific MP3 that is stores somewhere in the system, first of all, I'm going to have to know if that file is on the system. I'm going to need to understand if there's the presence of the file. But if I know that the file is present, if I have some way of searching for it, I also need to know who to ask for that file, because if we've got a peer-to-peer network of millions of nodes, the whole idea of peer-to-peer is that I don't have to be connected to all of those nodes. I can ask a node in the network to find the adjacent nodes and ask their adjacent nodes and then those nodes to ask other nodes, and there should be some way that I can locate where in the network the file that I'm looking for is located.

This is the challenge that you were solving with the Kademlia algorithm, the Kademlia algorithm for a distributed hash table. If I understand it correctly, the role of the distributed hash table in a peer-to-peer network is to be a routing table. It's supposed to help you understand where to find information in the network. It's not like you're storing the information itself in the distributed hash table.

[0:19:17.5] PM: It's actually the other way around. The core algorithm is a routing algorithm which we can describe, and this algorithm is in service to just providing a data obstruction to the programmer, which is exactly the same as a hash table. But they call it a distributed hash table just to indicate that it's implemented in some peer-to-peer way. But really what's happening is that we have a routing algorithm, which is the implementation of what's otherwise known as a hash table.

Because hash table is a data service. It has two functions; put a value under a key and query. Is there a value under a given key? That's just like the semantic meaning of a hash table. It's a question and answer thing. You can view as a data model. Basically, it's a table that has two columns. The first column is the key, which is a string, and the second column is the value. The routing algorithm is am implementation of these hash table database in a peer-to-peer system.

So maybe I can try to tell you how it works to see –

[0:20:32.0] JM: Sure. But just to be clear, am I completely mistaken that the routing table is core data structure for the Kademlia network?

[0:20:41.6] PM: The routing table is a core data structure for the Kademlia algorithm, which is a routing algorithm.

[0:20:48.4] JM: Right.

[0:20:48.8] PM: These writing algorithm can be used directly as – Can be reused basically as a hash table, but this is a second layer. This is the application layer basically of the Kademlia story.

From the point of view of a user, you really – A user is somebody who's sitting on one computer. From their point of view, if they're using the Kademlia library, they just say "join network", and then they have the two functions that they can apply against the network, store a key value or query for a key.

So what they see as a program is just service interface with basically a database, simple database interface with a store and a retrieve function. That's what the user sees. But what's happening inside the – The question is; how is this database implemented? This is where the Kademlia routing algorithm comes into play, because the idea is that the Kedemlia –

So let me try to explain it briefly. The Kademlia algorithm tries to – Basically, imagine a hypothetical space, like a high-dimensional hypercube for instance, but it's some high-dimensional space that is – Or you can think of it like a sphere, a high-dimensional sphere. Then, basically, every node that joins the network gets a random address. It's like on the sphere. So it's like this is a mental model. They just get an address where they live in this imaginary sphere.

We associate keys with any stringed key that the user might try to put in their database. We associate the string key uniquely also with the location on this hypothetical sphere and we simply postulate which ever node is closest on the sphere to the key is the node that I should try to find and ask about the key. This would be the node responsible for these keys.

The reason why we need the hypothetical sphere, is because every node, just like it works in social networks on the globe. Every person knows people that are near them geographically or near them contextually, like for instance job-wise or language-wise. But one way or another, they have these local links, but if they want to send a letter – So this is the famous social experiment. If you want to send a letter to somebody far away, it's enough to say the name of the person, the country, something general about them and just pass it on to one of your contacts that is roughly going closer and have them just keep passing through local contacts until they get to the destination.

So we created the exact same situation but in an imaginary sphere and everybody's contacts are the nodes that they're close to them on the sphere. We're just creating an imaginary world so they know how to route through it.

[0:24:08.5] JM: Okay. So I want to make this less abstract for people. So let's say I login to a peer-to-peer file system that is using a Kademlia distributed hash table. When I join the network, I get assigned a node ID. I find another node somehow. Describe what happens when I onboard to a Kademlia network.

[0:24:32.5] PM: So to onboard, you have to have a connection from outside to somebody who's already in the network. So you onboard through another node, and during the onboarding process – The purpose of the onboarding process is so that you become a member of the network. So to become a member of the network, you need to accomplish two things. First, you need to pick an address for yourself, which is usually some random number.

So these places you kind of – This gives you an address in this space of participants. Then after you have the address, you need to establish contacts to a small number of other participants, and these are contacts that you're going to be maintaining overtime. These are going to be your neighbors so to speak. They're going to do favors for you in terms of routing and you reciprocate.

But now who you're neighbors are, this is essentially the content of your routing tables. Who your neighbors are, initially you get help from the node that introduces you to fill your routing tables. So when you're being introduced, you pick your address. Your address essentially determines who are the people that are near you currently in the network, so the person, the contacts that is introducing to the network will actually query the network for you, find the IP addresses and generally information about the nodes that are closest to you, to your address, deliver it back to you so you can populate your routing tables with information about your neighbors. From them on, you're independent, because are now alive in the network and you have physical, which usually means TCP, contact to your neighbors in these address space. You can just keep maintaining this overtime basically.

[0:26:31.6] JM: Once I've joined the network, I can find other nodes. I also need to be able to find files or I need to be able to locate – Or I need to be able to help service a query. If I am just a node in the peer-to-peer network, I not have my routing table that tells me how to find certain nodes in the network. How do I locate another file, or what role do I play as a node in the peer-to-peer network in finding files?

[0:27:01.6] PM: Every node essentially services requests for finding files, and these requests come from your neighbors. Actually, they can come from anyone. So the way it works is that somebody says, "I'm looking for a file with a given key," and all you do is that you look at all of your neighbors and you see which neighbor is closest to the key of the file and you forward the question to them. So that's the algorithm. That's all everybody does, is they simply forward the question to whoever neighbor is closest. Finally, the person who the node that actually produces the answer is whichever node actually sees that they have the file itself on their storage.

So every node essentially first checks whether they have the file in their database, in their file system, wherever they store it. If they don't have it, they forward the question to a neighbor that is closer, and if they don't have a neighbor that is closer to the file address and they themselves don't have it, it means that the file just doesn't exist in the system.

[0:28:16.7] JM: Yes. That would be the get operation. If we're thinking of Kademlia in terms of the distributed hash table, somebody issues a get operation for a movie or for a song or whatever file they're looking for. They issue the get operation to any node in the network and then the get operation effectively propagates through the network.

[0:28:43.8] PM: Yes, but the important point is that it propagates efficiently. So it doesn't end up flooding, because if you remember, the rule is if somebody doesn't find the file or the value or the key on to their node, then they forward the question but they don't forward it to all of their neighbors, which would be flooding. They forward it to the neighbor that is closer to the key. So they only forward it to one neighbor. So the query goes in a path towards the node that would have it as supposed to flooding every time.

[0:29:18.8] JM: Yes. Okay. So the get operation for a hash table typically has the parameter of a key. So the key might be the name of the movie. Let's say it's Titanic, Titanic.mov or Titanic.mp4 or whatever it is. So the results of the get operation will be the actual file. So if the parameter is a string with the name of the file and the result is going to be the file, we have this key and this value and the –

[0:29:57.0] PM: So let me just correct you. The result is not the file. First, when you have a query, like Titanic, or this might be a file name, whatever it is. It's some textual key. These key first gets converted through just a hash function into a number, and this number is a number in the same space as the keys of the Kademlia nodes.

Because you want to have a connection between the address of a node and these keys. So they both have to be numbers. You reduce the structural query to a number. This number is treated as an address just like the addresses of the Kademlia nodes themselves. Then you simply postulate that you're looking for the nodes whose addresses are closest to the address of your query.

[0:30:56.9] JM: The key space gets broken up among the nodes based off of the different numerical rangers. Titanic is going to hash to some specific number and that number will be in some key space, and then the key space again is broken up across the network. How is that defined? How do you balance the key space among the different nodes in the network?

[0:31:25.1] PM: The key space in general is always something huge. Let's say it's 160 bits. This is an enormous huge key space, and then the question is how do you know which keys go to which nodes in the network? So the answer to this question varies dynamically. So the answer simply is whichever nodes are currently alive and are closest in terms of their key, so the address of a node is their key.

So whichever nodes are currently alive and are closest to the key for a query, they are the ones responsible for this query. We can just say numerically closest. It happens to be the geometric notion of closeness. That's a little bit more elaborate than numerical difference. But for the sake of this discussion, it's enough to say that the nodes whose own addresses are numerically closest to the keys, they are responsible for that particular key. So does this make sense?

[0:32:25.3] JM: Yes, it does make sense. The missing piece that I don't understand yet is if I make a query to the network, there are some deterministic routing that allows me to find the file Titanic based off of how that Titanic query consistently hashes to a node that's going to lead me to wherever the Titanic file is close to. I'm having trouble connecting how the file, the location of the file gets – When I joined the network, my routing table is established. Then I upload Titanic, right? I'm having trouble connecting the fact that if I have Titanic, if I've uploaded Titanic after my routing table has been established, how is there is a connection between the key space, like when somebody enters in a query?

[0:33:19.5] PM: I think I know what your confusion is. Because what you call to upload Titanic is actually a two-step process.

[0:33:27.5] JM: Okay. The put operation.

[0:33:28.9] PM: Yes. I think you're conflating, basically the actual file versus the indexing. So let's start from what happens. When you upload Titanic, when you hit upload on your application, under the hood two things happen. The file is already on your computer. So maybe it just gets declared as it's being shared. But the more important thing that happens is that you – Your computer, using Kademlia, finds the other node, the node closest to the notion Titanic. It computes the key for the word Titanic. This is the file that is being uploaded on your computer, but you have to tell the network that you have it.

So you actually have to store a key. Under the Titanic key, you have to store the information that you are being identified, whoever you want, the IP address or otherwise, that you have the file. The information of storing this meta-pointer towards you, this is what the Kademlia algorithm does, and this information is not going to be stored on your computer. Your computer says, “Initiate the storage operation,” but this operation is going to trickle through the network until it reaches the node in the network which happens to be responsible for the key space around Titanic. That’s how you inject the information.

[0:34:56.2] JM: I understand. Yes. To try to rephrase what you just said. If I’m going to upload Titanic, I am going to perform the same hashing operation to figure out what is the number that maps to Titanic and then I’m going to find the nodes in the key space who are responsible wherever the file Titanic falls in that key space and I’m going to tell them, “Hey, I’ve got Titanic,” and if somebody in the network wants Titanic, you tell them to go contact me.

[0:35:27.7] PM: Yes. You basically leave an envelope which says Titanic on the outside. On the inside it says how to find you. So anybody who’s looking for Titanic, they’re going to also end up discovering that same node that’s responsible and they’re going to look at your envelope and just see where to go next.

[0:35:48.0] JM: Does Kademlia account for the fact that when people are looking for a file they don’t often know exactly what the file name will be? In other words, they need to search. They need to do some kind of fuzzy matching. Does Kademlia itself account for that?

[0:36:02.8] PM: Kademlia itself doesn’t. Kademlia is an exact search due to the hashing function that’s applied to the query. So people use combinations and sort of compositions of multiple instances of Kademlia or just different name spaces of different key spaces to create indexing effects. But in general, there’s a limit to this, which is purely information theoretic.

This is actually the reason why the first file sharing systems were not very high quality products. There was a lot of noise, because you could only find things using exact matches. Later, with the advent of Google, it became clear that in order to rank who gets meaningfully ranked searches, you need sort of structural semantic information, and this is kind of beyond Kademlia.

[SPONSOR MESSAGE]

[0:37:06.7] JM: Stop wasting engineering time and cycles on fixing security holes way to late in the software development life cycle. Start with a secure foundation before coding starts. ActiveState gives your engineers a way to bake security into your languages' runtime. Ensure security and compliance at runtime.

A snapshot of information about your application is sent to the ActiveState platform; package names; versions and licenses, and a snapshot is sent each time the application is run or a new package is loaded so that you identify security vulnerabilities and out-of-date packages and restrictive licenses, such as the GPL or the LPGL license and you identify those things before it becomes a problem. You can get more information at activestate.com/sedaily.

You want to make sure that your application is secure and compliant, and ActiveState goes a long way at helping prevent those kinds of troublesome issues from emerging too late in the software development process. So check it out at activestate.com/sedaily if you think you might be having issues with security or compliance.

Thank you to ActiveState for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:38:37.1] JM: If somebody logs on to the network, they're looking for Titanic. They search the correct – They don't search, but they enter the correct name. They enter Titanic in the specific way that it's indexed in the Kademlia network and they are routed to me. They ask the network and the network finds a path to me and says, "Hey, this guy has Titanic. Go ask him for the file." Then the person asks me for the file. I've got a movie file that I'm going to give them. How do they know that I'm not going to give them a virus instead of giving them the actual Titanic movie?

[0:39:18.2] PM: Well, they don't know because this is just implied in the interaction. They're looking for something that advertises something. I mean, there's no – The short answer is they don't know and there was a lot of sort of bad content and viruses that were being spread using precisely what you're describing. So the way things work now – So nowadays, the mitigation of false content happens from the way in which you discover the files. Because nowadays people first look on the web for rankings of, let's say, Bit Torrent files, which are ranked according to the Bit Torrent – They're described through their Bit Torrent addresses.

So, first, you benefit from Google's ranking on those. So you actually are getting a highly vetted Bit Torrent link. Then you go to Bit Torrent and you're not looking for Titanic anymore. You're looking for a specific Bit Torrent source of Titanic. You see, you find the quality – You create the step where you want to find, see about a quality link happens on Google and then you go to a file setting system already knowing which file you want represented, for instance, with a Bit Torrent address and then the Kademlia network is just used to find this specific file.

Usually, the Bit Torrent files are also signed. When you discover the file on Google, you also discover the signature of the file's content. Eventually, when you downloaded some Bit Torrent, you can verify that you're getting the content that was advertised on Google. That's why it works today. Peer-to-peer systems never actually fix the problem of finding quality information. They just became systems for looking up specific file replicas identified essentially by their content digest, so like a shard. You actually use Google to rank just through the meta information.

[0:41:37.0] JM: I think this gets at what you've said earlier, that Kademlia has a scope. Kademlia itself is not a peer-to-peer network. It solves a problem within a peer-to-peer file sharing network. It solves the problem of a distributed hash table.

[0:41:53.2] PM: Yes.

[0:41:54.0] JM: So things like building a search index or ensuring that the file that gets delivered is secure is not a virus. Those are not exactly in the scope of Kademlia.

[0:42:05.8] PM: Yeah, not at all. Yeah.

[0:42:07.4] JM: When Kademlia came out, this was 2002, and it was revolutionary in how fast it could route people to the right location. What was the public reception to it in 2002? What were the early applications of it?

[0:42:22.1] PM: I think one of the earliest ones was my friend Jed McCaleb's eDonkey file sharing system and then Overnet. I believe from there it started spreading to most of the other clients.

[0:42:38.2] JM: Right. Now, people today often associate Kademlia and other peer-to-peer systems with Bitcoin, because it's kind of the trendiest application of peer-to-peer networking, but Bitcoin itself does not use Kademlia. Bitcoin is a peer-to-peer system, but it uses flooding. Why does Bitcoin use flooding and not Kademlia?

[0:43:01.6] PM: So Bitcoin is solving a different problem. So Bitcoin is peer-to-peer at the level of the exchanges. All exchanges talk to each other as peers. Meaning that they have to replicate – Or rather just the replicas of the chains even in a single exchange or cross exchanges. But there are fewer nodes. There are usually company or corporate nodes that have a lot of resources to start a whole blockchain. There's a small number of them and they all have to have the entire database, the same copy of the entire chain.

Whereas in Kademlia, everybody has a piece of a database and nobody has the whole, so that you can benefit from as much storage as possible. In the blockchain setting, you're not trying to benefit from storage, because everybody – At least all the exchanges want to have the entire blockchain there. So then the algorithm is completely different, because there's no need for looking up anything. Whenever you want to make changes, you have to tell all of the other participants, or if you have to reach a consensus of some sorts, you have to essentially talk to all of the other participants, because that's the definition of consensus.

It's a different application, but it is peer-to-peer in the sense that they're all equal and they behave according to the same rules against each other. Nobody is more important than the other.

[0:44:32.4] JM: IPFS does use Kademlia. IPFS is a system that has a cryptocurrency associated with it, but IPFS itself is not a currency. Why does IPFS use Kademlia?

[0:44:46.9] PM: I believe that they use it for the same – Without going in great detail, because I'm not familiar in the detail, but they generally use it similarly to the file sharing case that we discussed, but essentially they can – They have lots of files that are stored on different people's machines, and then they have meta-information, which says where the files can be found. Well, this meta-information is essentially key values, that the key is file block and the value might be where it's stored. They use the Kademlia algorithm precisely to index those most likely, and they have probably more sophisticated users on top of this. But this is likely a good example.

[0:45:36.6] JM: At some point in your career, your focus shifted from distributed hash tables, which are commonly used for decentralized distributed systems. You started focusing more on centralized infrastructure. You worked at DARPA, you worked at Google for a while. Tell me about that shift in focus going from the world of decentralized distributed systems to the centralized distributed system world.

[0:46:06.8] PM: So my longer term vision for – I think both peer-to-peer systems and distributed systems essentially embedded in nature. Distributed systems just correspond to kind of coherent organisms, so animals. Whereas social interactions between them using social protocols correspond to essentially peer-to-peer systems.

In the software world, you want to have both if you want to – So peer-to-peer systems with Kademlia became clear that they're possible, because you were always a little timid to even believe that it will work at scale, that the math will work, but it worked out. But then the longer term vision is now peer-to-peer systems, you have to connect very intelligent nodes that are not just routing and making hash table storage.

So let me give you an example of what this vision looks like. Every person could potentially have their own cloud application running in some commercial cloud of their choice; Amazon, Azure or in some other country. This cloud application can be essentially the representation in the digital world in the following sense. It can, first of all, do services for them. For instance, back up everything that's going through their personal devices as well as it can represent them through protocols in interactions with other [inaudible 0:47:35.1] personal cloud assistance essentially.

So these interactions can be much more complex than just doing either just file searches or crypto transactions. But in general you can have simply a standard where people can have – Okay. So this representation in the cloud. Now, because it's pretty clear that even for your own personal kind of node, representation for your presence in a peer-to-peer system, nowadays, just running a single computer doesn't capture everything that you might want to do. People are sophisticated users of the internet now and people manage databases and they provide services to other people.

An individual has to be able to modularly be able to sophisticate its applications that represented them in a peer-to-peer society on the internet. So how do people create sophisticated fully kind of self-sustained requiring no personnel software? That's the question of essentially building distributed systems.

But the reason I got interest in this is because companies build distributed systems and sustain them because they're able to benefit from people, engineers, like operations engineers who continuously fix problems that they're not fixable automatically. In some sense, corporate software is always leaking and there's lot of people and stuff to make it work. You can afford this if you want to build a personal cloud application, because by definition, every individual is just one person. So they need to be able to reliably create modular applications in the cloud that represent their interests without needing the sophistication of a whole company.

I believe this is possible. I believe that through proper engineering methodologies and techniques, in particular, languages and other such techniques, it is possible to make programming of large distributed systems simple and save the way programming for a single application used to be for a single computer.

[0:49:52.8] JM: Why the focus on the languages? The idea of looking to a language to simplify the interaction with distributed systems.

[0:50:03.8] PM: Well, a language is simply a development tool that allows you to sort of compose lots of pieces together and kind of gives you some facilities to make sure that they fit well so that you're managing your things. It basically kind of lets you know when you've made a mistake in your own kind of designs.

So, ultimately, the language is the point where the author – So, in my example, maybe a person who tries to build a cloud application. The author expresses themselves to the machine. So the language is what faces the human. First of all, by virtue of this, it has to be useful and natural.

Now, there's another detail which is that people like to think in one language. Generally, when people switch between languages, whether it's natural or programming, inefficiencies arise because languages present different mental models. So I think a lot of people hold the opinion that in a hypothetical world, ideally, everything that you need to express to make a program run should happen in one language because this is the most effective way both for the programmer to express it. But also if a software is expressed in multiple languages, there is no tool that makes sure that the parts that are in different languages talk coherently to each other, because such a tool will essentially – So people use sort of ad hoc solutions. They define protocol languages for connecting technologies and so forth, but this complicates the process. If everything is in one language, one is able to kind of have an end-to-end check of a large system directly at compilation time as supposed to writing integration tests.

I mean, the simple answer is that different languages mix mental models and ultimately has become – I hold the opinion that actually there is only one mental model that describes all programming workflows. So this is simply essentially directory cyclic graphs, which is from the compiler point of view, this is known as the SSA representation of functions, which means it stands for a single static assignment.

But the bigger point is that it's a dependency graph, a dataflow of calculations. So this is the basic way in which we describe a computation under the hood of any programming language or under the hood of how higher level cluster pipelines are created. So there is a uniform way, semantic, behind how we build things and it's not reflected in a technology, because technology grew overtime and that's why it kind of is very disparate.

[0:53:03.7] JM: All right. Interesting answer. I want to close off with just a broader question. So the two biggest distributed systems communities today, I think broadly speaking, you could say are Kubernetes and then the cryptocurrency ecosystem. You have these two disparate communities and the Kubernetes community is vibrant with corporations that are looking to revamp their infrastructure. They're looking to revamp their operations and there's a whole of money going into Kubernetes for those reasons.

Then you have the cryptocurrency ecosystem where they're trying to reframe the entire way that humans do transactions throughout the world as well as the immutability and the append only and censorship resistance and so on. Is there any overlap that you see between these two communities that has been interesting to you?

Because, I mean, maybe they're totally disjoint and it wouldn't make any sense for there to be overlap. But I just see it as strange, like I went to Cube-Con. I think you were there or maybe you didn't go. In any case, it was strange. You have this distributed systems conference and nobody was really talking about cryptocurrencies, which I mean distributed systems is so big that we need to have sub-conferences within that. But I don't know. I just found it interesting. I don't know if you find that thread interesting to pull on.

[0:54:31.3] PM: I mean, I think that's actually a very interesting observation. I haven't thought of it. I'm sure that my partner Joseph Jacks, my partner in business, would have an interesting answer to this. My interpretation is that perhaps the cryptocurrency community is not – I don't know this for a fact, but perhaps it's not contributing essentially tools back into the Kubernetes ecosystem.

I don't know if this the case, but the reason I say this is because the tendencies that people who are trying to promote new tools for working with Kubernetes tend to be at these conferences. I will say that the crypto community in general is interacting at least in meetings and discussions with the companies that are focused on building Kubernetes product. But from the projects that we have been looking at in terms of tools and the ecosystem, none of them have come from cryptocurrency companies.

Cryptocurrency companies are new. I suppose they haven't had enough time to create a quality product, because, I mean, any tool for Kubernetes probably requires at least a couple of years to come to a point of maturity where you would be able to show it at a conference. So maybe this is part of the reason why.

[0:55:55.2] JM: Yeah. We'll see. Petar, thank you for coming on Software Engineering Daily. I am excited about whatever you and Joseph are building. I know it's kind of a stealth project right now, but whatever it becomes, I'm sure it'll be exciting.

[0:56:08.3] PM: Thank you so much.

[END OF INTERVIEW]

[0:56:12.2] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]