SECURITY NOW!

Transcript of Episode #325

TCP Pt. 3 - Necessary Refinements

Description: After catching up with the week's news, Steve and Leo return this week to their "How the Internet Works" fundamentals series. They examine the challenges presented by "packet-based connections" to further understand the operation of the Internet's most popular and complex protocol: TCP.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-325.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-325-lq.mp3

Leo Laporte: It's time for Security Now!. Hello, everybody. Leo Laporte here. And there he is, Mr. Steve Gibson of GRC.com. He's cheerful.

Steve Gibson: I cracked you up.

Leo: How are you cheerful? I would be so - in fact, after this show I'm often a little bit glum because I always feel like the bad guys are winning on this stuff.

Steve: Yeah, we're just spectators, Leo. We and our listeners, we're sitting back, watching all of the packets fly. And lord knows what's going to happen.

Leo: Better sitting back than sitting ducks. And I guess if you listen to this show, at least you can say "I'm not a sitting duck."

Steve: I think that's absolutely true. So no duck.

Leo: So today, Part 2, no Part 3 of our TCP explanation.

Steve: Yeah. What I wanted to talk about, and I thought I was going to initially, was this interesting next-generation application layer protocol called SPDY, which Google and the Chromium Project are working on. The thing that put it on my radar was that it's going to be in our Kindle Fires when they come in a couple weeks. That's the technology used for dramatically speeding up web access. And we've often talked about HTTP, which is the traditional web protocol, the application layer protocol, meaning that we have the underlying protocols which we've been talking about recently, the IP protocol and then ICMP, UDP, TCP. Those are transport layer protocols, part of what's regarded as sort of the underlying transport of data. But the application layer runs on top of that. That is, TCP is the carrier for HTTP packets.

Well, what Google, because they're so web focused, and I'm really glad that they are, what they recognized was that the web has changed dramatically in the last decade, certainly in the last two, since HTTP was originally conceived. The nature of pages, the type of content, the way pages are being used is very different from what it was. And HTTP is beginning to show its age. There are many things we could do now, taking advantage of higher bandwidths, stronger servers, and helping us to deal with just the complexity of today's pages.

So this SPDY protocol is really interesting in that it addresses those things. But as I sort of laid out what I want to talk about, I realized that many of the problems it solves require an understanding of TCP that we don't have yet. So I said, okay, we've got to lay down another layer of TCP understanding in order to get what it is that the SPDY protocol which runs on top of TCP helps to solve.

So today we're going to look more closely - we've already looked a couple times at some aspects of TCP. Now we're going to look at what it takes to get as much speed as you can, because that's of course what everyone wants, over a packet-switched connection. Because it's very different, when you think about it, to be sending blobs of data off. And the question is, how fast can we go, and how do we know how fast we can go. And that's what happens, you just saw it, if you try to go too fast.

Leo: You fall apart.

Steve: Yeah, the message does not get through. So that's today's topic. It's going to be a great one.

Leo: Excellent. I imagine there's a few security updates in the hopper today.

Steve: Things have been quiet, actually...

Leo: No.

Steve: ...on the update scene.

Leo: Yay.

Steve: Happily. We do have a bunch of news, though. I got a lot of it from the SANS newsletter, which just sort of - this is a newsletter I've been subscribing to for years, and I would really recommend it. I've referred to them throughout the entire length of the podcast.

Leo: Some day I would love to get you just to write down the stuff you do to keep up on this stuff - the newsletters, the websites, all that stuff.

Steve: I'll tell you, what I do often now, thanks to Twitter, is watch my own feed because people are just - our listeners are just...

Leo: Great use for Twitter, yeah.

Steve: ...so great about sending me stuff. I'll see something and click a link and open a tab. That's actually one of the reasons I end up with so many tabs in my browser is they're placeholders for things I want to get back to for research that I'm doing. People mention things, and I go, oh, yeah. That looks good. I'll come back and read that later. But...

Leo: Well, so it's @SGgrc, by the way. So if you want to follow Steve or feed Steve, you can do it there.

Steve: Yeah. So the EFF has done a study. They have something they call the SSL Observatory that we've talked about a couple times. And I have it in my notes here to do a complete podcast on it because it's very interesting. It's basically an Internet-watching system that watches SSL traffic and builds lists of things going on. And what came to their attention was that four major certificate authorities have been compromised in the last four months.

Leo: Four.

Steve: Four.

Leo: So we all heard about the Dutch one. But this is four.

Steve: Yeah. DigiNotar, of course. And Comodo, I'm not even sure if Comodo was part of this. But there are the so-called Certificate Revocations Lists, these CRLs we've talked about as one of the means for a browser to verify that the certificate it has is still trusted. Because certificates expire, but it may sometimes be necessary to revoke them. I've, for example, taken advantage of that system when - and I don't remember now what it was. It was a few years ago. But I needed to supersede a certificate that was otherwise still valid with another one. I had to change something about it, or I was experimenting. I don't remember what now. But I found that - and it was funny because I got some feedback from people who said, hey, my browser just said a certificate was revoked. It's like, yeah, but it was on purpose because I've got a new one. And so somehow they weren't seeing the new one, or their browser was telling them something was going on with GRC.

But in the CRL, the Certificate Revocation List, there's a reason for the revocation specified. And they are - there's a null, it could just be not specified; or the affiliation was changed; the CA was compromised, that is, the Certificate Authority who issued the certificate was compromised; the certificate was put on hold; the certificate was formally ceased operation, so it's a way of sort of like formally shutting down your use of a certificate; a key compromise.

And in the list of these, surprising how many, in terms of count, key compromises there were. So that would mean that, for example, the owner of the certificate, like an Amazon.com, just to draw an example, would have lost their private key, which the certificate contains the matching public key. So if they realize that they no longer control the key for the certificate they're issuing, that's not safe because it allows other people to intercept and decrypt their traffic. So key compromise would be another reason that you would administratively revoke a certificate. Also privilege withdrawn; superseded, as I had mentioned before; and unspecified reason.

So this SSL Observatory has been collecting data. And what they found was that there had been, in the last four months, a substantial jump in the number of certificates revoked for reason of CA compromise. And when they weeded out the duplicates and the same CA saying that it had been compromised and so a number of certificates were withdrawn, they found that there were four which were now claiming that that was the reason for withdrawal. So that represented an interesting uptick in the rate at which this is happening. And I liked what the SANS editors had to say.

An editor whose last name is Liston said: "The entire SSL Certificate system is founded on the faulty premise that we should trust a corporation simply because they claim to be trustworthy. These companies have taken on a huge responsibility as a part of their business model, and have simply not taken the kinds of precautions one should take when voluntarily positioning oneself as the basket containing all of the chickens." And Murray...

Leo: In other words, don't put all your chickens in one basket.

Steve: Yeah, but unfortunately, as we know, that's the way the CA system works. We have 600-plus Certificate Authorities that our browsers trust. And the model, unfortunately, is such that every single one of them must do a perfect job in order for the system to work.

Leo: Ah. If any one is bad, like DigiNotar, then you're screwed.

Steve: Exactly. And that's why our really old-time listeners will remember the shock that I shared once, many years ago, when I looked after - a long time ago I remember there were, like, 12 Certificate Authorities. And that was VeriSign and Thawte and a couple others. It was like, oh, that seems manageable. Then I looked, and this list had exploded without my having noticed. And I remember sharing with you, Leo, and our listeners, it's like, oh, my god. This is not good. And that's what it turned out to be.

Another editor said: "We do not have to have a perfect system of key management. But vendors who want to offer services in the security space have to have good security. Their brands are essential to their viability, and they are very fragile." And of course that's exactly what we saw with DigiNotar, which is now bankrupt because they didn't have good security. In fact, we learned that some of their passwords were ridiculously weak, and they're gone now.

I also picked up a little blurb, also from SANS, that said that a U.K. high court judge has just given the British Internet Service Provider BT, which I think is British Telecom, two weeks to implement a plan to block a site, Newzbin2, which is a membership-only site known for making pirated content available. The ruling is the result of a lawsuit brought by none other than the U.S. movie companies.

Leo: Oh, boy.

Steve: Our friends at the MPAA. The judge decided that BT was aware of the copyright infringement activity occurring on the website Newzbin2, and had ruled in July that the company must prevent its customers from being able to access that site. The judge - now, here's also the point that was brought up as being a real problem by these editors. The judge also ruled that "the costs of implementing the order should be borne by BT."

So let's step back a minute from this. This says that a random service provider - I mean, BT didn't do anything wrong. They're just the target of this lawsuit brought by the U.S. movie companies. And a judge has ruled that they, the ISP, have an obligation to block their customers' access to this one particular website, which is known to make pirated content available, at their cost. So anyway, the editor, Murray, in his little editorial comment, said something that just jumped out at me. I loved - I wanted to share it with our listeners because it was just, I mean, I'm sure this will be retweeted by the people who are listening live and others when they hear it.

So Murray said: "It seems to me that BT is a victim here. They're being made responsible for the criminal activity of others. They are being forced to do something both expensive and ineffective." And here's this line I love. He said: "The Internet routes around censorship." I just - I love that. It does. We've talked about packet routing. Well, on a higher level, the Internet routes around censorship. I think that's just a beautiful phrase. And he said: "How much damage are the rest of us supposed to endure because the publishers cannot figure out how to offer their products at a price both profitable to them and not so high as to create a black market."

Now, okay. I would argue a little bit that the Internet creates so little friction in the area of piracy that it's not clear to me that you could have products which are both profitable sufficiently to support the expense of creating them, and also not so high as to create a black market. I mean, it's just too easy and simple to steal bits on the Internet. But I do love that phrase, "The Internet routes around censorship."

And the other editor, Liston, said: "This all sounded somewhat reasonable up until the last sentence. If the movie companies expect ISPs to block access to sites at their behest, then they really should be footing the bill. They own the copyright. They benefit financially from its protection. So expecting a disinterested third party, the ISP, to cover the costs of implementing a block on infringing websites seems a bit over the top. So anyway, I certainly agree with those things. And it's a little troublesome that an ISP is being told, you have to prevent your customers from accessing this particular website. A judge has ruled it, and they have to support the cost and provide the technology of doing so.

Leo: [Disgusted sigh]

Steve: Yeah. And, unfortunately, we talked a couple weeks ago about remote access to Medtronic insulin pumps. You'll remember that, Leo, because your comment was at the time correct, which was that it didn't seem like it was that big a problem, and you had to be very close to them. Well, turns out - this reminded me of Bruce Schneier's great quote, where he said, "Attacks never get worse. They only get better."

Leo: Oy. The attackers only get smarter.

Steve: Exactly.

Leo: They don't get dumber.

Steve: So a researcher who last year developed a method of taking control of ATMs so they would dispense cash at his behest, has now devised an attack that allows him to take control of certain wireless insulin pumps. The attack could be used to deliver incorrect doses of insulin to patients. The pumps in question, which are made by Medtronic, contain radio transmitters that allow doctor and patients to make adjustments. With specialized equipment, the attack could be conducted at a distance of up to 300 feet and no longer requires the attacker to know the device's serial number. The pumps at present do not use encryption while transmitting information.

So when we reported on this earlier, the distance was much lower, and you did need to know the serial number of the device. Well, the attack has improved, and it's no longer necessary to know that. And the distance has improved, as well. And I remember that in one of our Q&As we got feedback from a user. He and his son were both users of these devices. And I'm just recalling this now, I think this might have been the Q&A that Tom and I did, Leo. So I don't think it was you and me. But what I remember coming out of that was that the great danger would be that a large dose of insulin could be dumped into the person's bloodstream.

Leo: Could kill him, of course.

Steve: Exactly. And it could have fatal consequences. So I hope the security on these devices is fixed quickly; and that, if anyone has them, they may want to, I mean, our listeners, being security aware, may want to be at the front of the line for getting an improved device.

We've got troublesome legislation that's been submitted in the House of Representatives in the U.S. There is now a bill that would increase the government's authority to shut down websites that offer products that violate copyright and trademark laws. We've seen this sort of legislation being talked about before, but this is only last week. Another bill has been introduced. This proposed legislation would allow the Justice Department to obtain court orders requiring ISPs in the U.S. to stop resolving DNS for the offending websites. The sites would still be accessible outside the U.S. The bill would also allow the government to order search engines to remove certain websites from their results.

And what's really odd - and I don't understand what this means, and unfortunately things that are not well specified are always sources of trouble in the law. It says the attorney general would also be granted the authority to block distribution of workarounds to allow access to blacklisted sites. So our technical listeners - which would probably be pretty much everyone here who's been following along.

Leo: Everyone who's still awake, yeah.

Steve: Yeah, we know that DNS prevents - blocking DNS prevents a user from resolving the domain name into the IP address. So all you have to know, if you were a pirate who wanted to access these sites, is the IP address. And you could use somebody else's DNS servers to get that, rather than using your own ISP's DNS servers. And so it's - I don't know what it means for the attorney general to be granted the authority to block distribution of workarounds. But, you know, that's worrisome.

Leo: Doesn't sound good, yeah.

Steve: It does not sound good. And Symantec also noted that they had seen a concerted attack against chemical and defense companies, with targeted attacks using a tool that they call "Nitro," which is based on a readily available Trojan known as Poison Ivy. And these attacks are being launched largely at computers in the U.S. and the U.K. and Bangladesh, for some reason. But there were 17 other countries targeted, as well. And they're being launched through email attacks sent to IT departments at these targeted organizations, pretending to be requests for meetings or warnings about unpatched Adobe programs. So unfortunately they're email that IT departments would tend to believe because who wouldn't believe something about Adobe.

Leo: Yeah, of course.

Steve: And unfortunately these targeted attacks are infecting people with this Poison Ivy malware or trojan that allows people to gain backdoor remote access.

And a number of people in Twitter sent me this news. You may have seen it also, Leo, that there is now a MAC OS X bitcoin mining malware.

Leo: Yeah.

Steve: So it's malware known as DevilRobber, which has been detected on Mac OS X computers. It makes its way onto computers by being bundled with Mac applications available on filesharing networks. DevilRobber has several components. It attempts to steal usernames and passwords; tries to steal users' bitcoin wallets, so if there's existing money, currency in the bitcoin wallet, it tries to steal that; and also hijacks computers' processing power to conduct bitcoin mining. So anyway, that's also happened. And I guess that was...

Leo: Are you still hot on Bitcoin? I mean, I think it's kind of over; isn't it?

Steve: Yeah, it is.

Leo: I mean, it crashed, the market crashed for it.

Steve: Yeah. I wouldn't say I was ever hot on it. But I thought it was really interesting crypto technology. And it was cryptographic stuff that we've studied and understand on the podcast, applied to something completely different than communications. It was like, okay, we're going to make a currency, and make it secure. And they really did. They did the crypto right. But it's all the other, all the social side of stuff that has been, of course, the problem.

And I did want to mention that I'm seeing backpedaling on this DuQu trojan. Remember that we mentioned that there was a lot of fervor a couple weeks ago, saying that it was closely related to Stuxnet. Now people are beginning to say, well, now we're not thinking that's so much the case. It's turning up all over the place, all over the world. So it is propagating, and it's getting itself around, although researchers are not yet clear about this thing's intentions. So it's funny, as I wrote that note to myself, I thought, okay, what world are we living in where we're concerned about the intentions of something that's, like, loose on the Internet and operating autonomously.

Leo: Oh. It's the camel's nose in the tent, you know?

Steve: It does feel a little bit "Daemon"-like.

Leo: Hmm, what are the intentions of that malware?

Steve: And so two things from the Twitterverse. Matt Yakel sent - he said, "OK, SGgrc, you're right. My Firefox 7.0.1 was about 800MB of RAM this morning. I killed it and restarted, now 250MB. Two minutes later it's up to 320." And he did the #FAIL in his tweet. And then along the same lines, Sean T., whose handle is @SeanT6, says, "Hi, Steve. Try out Memory Fox. It really works and reduces Firefox memory use. It's at the Mozilla add-ons page. And when I went over to get the actual name for this person who was using the handle SeanT6, and I saw that he was Sean T., I saw that he had also tweeted: "Testing Firefox 8, I can say Mozilla has awakened." And I'm not really sure what he means by that, but then he continues, "But another add-on called Memory Fox has fixed the memory bug. I need to load 100 tabs." So, like me...

Leo: Why, why, why? Need or want? Need or want, that's my question.

Steve: Yeah. Sean is a tab user, the way I am. And so I have hope. I found Memory Fox. I haven't checked to see whether it's compatible with my Firefox 3.something or other, because I went all the way back to 3 in order to protect myself. I'm not sure what I'm going to do. People said Firefox 7 fixed the problem. Now we know from Matt that 7 did not fix it, that it's still got a problem.

Leo: Can I ask you, though, I mean, we get, on the radio show, I get calls all the time. "Oh, my gosh, my Windows has no memory free." And modern operating systems are designed to let apps use all the memory they need as long as it's available. I mean, why let - if you have a ton of memory, as we all do now, why let it sit fallow if an application can use it? The key is not whether the application's using it, but whether it releases it when politely asked; right? Or it's not needed.

Steve: Well, kind of. I noticed, for example, that when I'm running IIS, Microsoft's Internet Server - I run a copy on my system here because I use it for developing websites and all of my JavaScript code and everything. So, for example, at this moment the IIS process is using 5.8MB of memory, yet its virtual machine size is 260MB. So what that means is that it has allowed itself to be moved out of memory. It's actually, it's got a small little footprint in actual memory; yet most of it, again, it's like 6, little less than 6MB of actual memory being used. But it's allocated 260MB, most of which has been swapped out and is sitting on the hard drive. Which is really good behavior. Firefox, on the other hand, right now, I'm looking at the memory stats, is using 271MB of my memory. And I've got one, two, three, four, five, six, seven, eight, nine, 10, 11, 12, about 15 tabs open.

Leo: But how much memory do you have?

Steve: 280MB.

Leo: Yeah, but wait a minute? How much memory?

Steve: 280MB.

Leo: But how much RAM do you have?

Steve: 4GB.

Leo: Okay.

Steve: Yeah.

Leo: Well, shouldn't it just use as much as it needs until somebody else needs it, and then release it? I don't think paging, unless you need to page, is a good idea because that slows you down.

Steve: Well, okay. But why is it, then, that I come in in the morning, and I'm getting a message saying that I'm out of, I'm completely out of all memory.

Leo: That you don't want to see.

Steve: And Firefox is 1.5GB.

Leo: That is a memory leak. But I'm saying that people often will get upset because there's no free RAM. But that is how modern operating systems work. Programs allocate RAM as needed and then release it if other programs need it. There's no reason to release it prematurely. It slows down operations. So if Firefox is prefetching a lot of pages or doing - I don't know what it's doing with all that memory. But let's assume that it's doing something reasonable with that memory, as long as it releases it when it's needed. But otherwise you've got 8GB just sitting there. You want to use as much RAM as necessary, as long as there's free RAM when needed; right?

Steve: Well, there really is no "give us back the memory you're not using" API.

Leo: Sure there is.

Steve: No, there isn't. You can't ask an application to give up memory.

Leo: Really? I thought all modern operating systems...

Steve: No. They're not elastic.

Leo: Are you sure?

Steve: Oh, yeah. They're not, I mean, so I've been programmed...

Leo: Well, I'm not talking about malloc and calloc and all that stuff. I mean - well, all right. I may be wrong.

Steve: Yeah, there isn't a, hey, do you really need what you've got? See, I think what...

Leo: Well, I think that there's a way to mark the memory you've allocated as releasable.

Steve: Well, okay. But that, yes, you're able to lock it in memory, or you're able to say - you can release it for the OS to swap it out.

Leo: Right.

Steve: But the presumption is, when you access it, it'll be there. So you can definitely page lock memory to keep it in active RAM. Or it is much better behavior, and I think that I'm sure this is what you're thinking of, to let the memory that you don't actually need in real-time, sort of more like reference stuff, to be swapped out. The problem is that, for example, 32-bit operating systems have an actual physical limitation of 4GB of allocation. That is, that's what 32 bits can access. So you really need - we really need to go to 64-bit OSes in order to get that much more memory. But I guess I'm arguing that this kind of memory consumption for pages that scroll, called a browser, is ludicrous. I mean, it represents a level of slop and waste and poor coding that is just egregious. I mean, granted, browsers are doing a lot these days. But 1.5GB is just crazy.

Leo: I stand corrected.

Steve: Yeah. They're just, they're - yeah, anyway. I did want to mention, I had in my Miscellania to mention what I mentioned to you before the show, and that is, for those who are using Gmail, Google just released and the iTunes store just posted, but you said just pulled away, pulled back, a native Gmail app for iOS for the iPhone and the iPad. I did get it on my phone because I had sucked it in, I guess, before they took it out. But you said that they had just yanked it.

Leo: Yeah, apparently Google yanked it because of a notification issue. I see it's not there right now. It'll be back.

Steve: Yup. So, soon there will be a native Gmail app, which I think a lot of people will be really happy with. And also in Miscellania I just sort of thought I would, as I'm tracking XP, the operating system I'm still happily using - and, by the way, we have 887 days of support remaining. I saw a CNET story with the latest desktop operating system market share. XP is still more than any other desktop operating system in the world at 48 percent. Windows 7 is in second place at 35. Vista at 9. Mac OS X v10.6 has a 4 percent share; v10.7 has a 2 percent share; Linux a 1 percent share; and all others combined 2 percent. So XP is still in the lead.

Microsoft, of course, is famously pushing companies away from XP, trying everything they can, talking about how insecure it is, and Windows 7 is much better and so forth. But there's just so much resistance, mostly on corporations' part, to move. They've got their IT departments who understand XP. It does everything they need. And that's my problem, too. It does everything I need. And I look at 7, and I think, well, okay, it's bigger, and I've got to fight with it to get a UI that I'm comfortable with and so forth. So XP is still No. 1 at 48 percent, with Windows 7 in second place at 35.

And I did want to share a short blurb from a nerd, a Nerd On Site nerd, who sent a nice note saying, "Steve, I have my own copy of SpinRite that I've been using on clients' machines. I have on occasion suggested it to clients and other nerds," which of course I appreciate because that's how we sell more copies of SpinRite. We're, as you know, Leo, we're relaxed about licensing. I like people who are consultants to follow our consultants agreement, which says, if you have four copies, then you are free to use it on your clients' machines.

And the reason I sort of have it that way is that allows a person to get one copy, try it, see how they like it. If they realize, hey, they're profiting a lot from their use of SpinRite, well, it'd be nice if GRC was, too. And so if you maintain four copies, we call that either the consultant or the site license, meaning that a company that was using it on all of their machines to maintain their computers could also do so if they just keep four copies.

Anyway, going on, he says, "It's a nice tool to have along. I've turned non-booting PCs into booting ones, and in one case a workstation on a domain with a bad sector in the user's local profile, which caused them to instead log in as a temporary new profile every time they tried. SpinRite read and restored the data, and in a few minutes they were able to log back on into their original profile and could finish up while I sourced a replacement drive and then cloned it for them. Thanks so much for a great product." So thank you, Nerd.

Leo: Yay. Nerd On Site. Already, Stevie. Let's talk TCP.

Steve: Okay. So everybody needs to close their eyes and put their propeller beanies on because this is really cool conceptual stuff. So we have a network which, as we know from the How the Internet Works episodes we've done so far, exists sort of as a loose confederation of routers where we send data to destinations by addressing them with an IP address, and the routers along the way each forward the packets of data, the individual packets of data that they receive to the destination.

Now, the question that we haven't looked at yet - we understand about packets and IP addressing, about the TCP protocol that lives inside the IP protocol. TCP adds the abstraction of a source and destination port, which it adds to the IP packet, which has the source and destination IP address. We've talked about the need to number the bytes. We use a synchronizing approach where a SYN packet is sent to establish the numbering that the sender will be using for the subsequent data that it sends. The recipient of that sends back a SYN/ACK which is acknowledging the receipt of the SYN and also sending a SYN to establish its packet numbering. So we've got the mechanism of data going back and forth.

What we haven't talked about yet is speed. And speed is really tricky. It's actually incredibly tricky, so much so that, even recently, researchers are still working on refinements and tweakings to the question of how do we make this system work robustly and give us as much speed as we can get. Now, back before the Internet, this wasn't a question. We would have a modem at each end, and it was the connection itself, the baud rate, as we called it, that established the rate at which the sender could squeeze data through the phone line to the other end.

And so the application would just dump a blob of data into a buffer on the modem and basically sit around twiddling its thumbs while the modem went [imitating modem sound], got that data, through audio tones of various levels of complexity, through to the other end. So it was a matter of, like, just put as much data in the buffer and wait for that buffer to drain to the other end.

Well, that model isn't anything like what we have now. Now we have this loosely connected network. And imagine that a sender wants to send a big file. And the problem is that the data needs to be packetized. The packets need to be sent from the source to the destination over a set of links that we know nothing about. We don't know how fast they are. We know, we may know the speed of our own connection to our ISP. We're buying a certain amount of bandwidth from our ISP. But the data may go over a slower part, or there may be a router that it's passing through which is not functioning right or bogged down carrying a lot of traffic because it's a very centralized located popular router. And so the router may be having trouble.

So when you think about it, this question of speed is a real dilemma because we may be able to send packets out at a speed faster than they are able to arrive. And then there's the other question of what about the sender being able to receive them? You and I, Leo, were just talking about buffer space in the operating system and not having enough memory. So we also need some way of knowing that the sender can receive the data that we're sending. The packets are coming in, whether the receiver wants them or not. What if it doesn't have enough memory? It's like, aagh, stop.

So the designers of TCP knew there were going to be all these problems. And over time the approach has evolved. But one of the first things that happened was a concept known as a "receive window." TCP uses the term "window" sort of from a - just sort of as an abstraction for something going on that you need to watch. So every acknowledgment for data received - remember that the ACK packets are coming back acknowledging the highest numbered byte which has been received so far. So that if acknowledgments don't come back to the data that the sender has sent, the sender thinks, oh, well, maybe the recipient didn't get it. It got lost in transit on the way. Or it could have - the acknowledgment packet could have gotten lost in transit on the way back. So the sender sends it again.

And so these acknowledgment packets acknowledge the highest numbered in order byte that has been received. And I say it that way because remember the other thing that can happen is the Internet can deliver things out of sequence. And if packets come in out of order, the receiver will hold ones that it wasn't quite expecting yet, hoping that the missing pieces get filled in. But the way TCP protocol handles this is it only acknowledges the latest in-order packet. That way, if one was missing in the middle, then that one would never get acknowledged. The data that it contained would never get acknowledged, so the sender would know that one needs to get resent, even if it had already sent some other ones later on in the stream. So as soon as that packet got received, then the recipient that was still holding the later packets could jump its acknowledgment way forward so that the later ones wouldn't have to get reset. It's really very clever. I mean, when you look at something basically simple, perfectly thought out, it's very clever.

So the question is, one question, is how much can we send? Not how fast - we'll get to that in a second - but how much? That question is answered by every acknowledgment which comes back to the sender containing a 16-bit field in the TCP header which is called the "TCP receive window." And that's just the name of this 16-bit blob, this 16-bit data field. And what that means is the value in that field is the amount of buffer space, the amount of received buffer that was available at the other end at the time the packet was sent. So when that acknowledgment was sent, the acknowledger, who's receiving data and acknowledging it, says, okay. At the moment I've got this much space available in my receive buffer. So it sends that, along with the acknowledgment, to tell the sender that, as of - and think about it, that the acknowledgment also contains an acknowledgment of the highest numbered byte received. So it has that and, at that time, how much receive buffer was available.

So again, just so cleverly, what those two pieces of information tell the sender, even if the sender has already sent additional data, because all of this is sort of happening asynchronously, it's all - we're wanting to keep everything moving. So there is permission in the system for data to be sent ahead of being acknowledged. And that's important because otherwise nothing would go very fast. If we had to wait for every single packet we sent to get acknowledged, then our total bandwidth would be constrained by something called the "bandwidth delay product" of the network. That is to say, we know that packets can be typically like 1500 bytes, which is the maximum size. We have to subtract some from the header for the packet. 1440 is the typical size. It's the number I was trying to remember a couple weeks ago, and I had...

Leo: The MTU says it's 1500. But after overhead, 1440 is what's...

Steve: Right. MSS is the Maximum Segment Size, 1460. Anyway, so imagine that we sent 1460 bytes, and they got there and then were acknowledged, and we couldn't or didn't send any more until they were acknowledged. Well, so we would never be able to go fast because the roundtrip delay through the Internet of waiting for the acknowledgment of what we had sent would - and the fact that packets are of a certain limited maximum size - we would only be able to send the amount of a packet times the roundtrip time to get that packet acknowledged, which would really constrain the speed at which we could operate.

So the designers realized we had to make it possible to have data outgoing to a recipient in flight, essentially, and have that being in advance of acknowledgment. The acknowledgments could be delayed. And in fact senders don't even have to acknowledge every packet they send. If data's coming in quickly enough, and it's been only a short time since the last acknowledgment was sent, the TCP protocol allows the sender to wait a few, get a few more packets, so that fewer acknowledgments are being sent, just in order to, again, further decongest the Internet overall. So this receive window is extremely clever because, at the time that that acknowledgment is sent, the sender is saying I have received up to this much data from you, the sender. And at that time I've got this much buffer space available.

So every acknowledgment that is received by the sender tells it, at the time that the receiver had received this much data, this much buffer was still available. So that gives the sender permission, a kind of ongoing permission for how much data it absolutely knows it's able to send. Now, naturally, as always seems to be the case, there have been some attacks which have some warpages, some abuse of something so cool as this facility. So one interesting hack occurred to the hackers, and that is...

Leo: They spend a lot of time thinking about this stuff.

Steve: Exactly.

Leo: Constantly.

Steve: What would happen if we sent back an ACK packet that said zero, that is, with a window, a TCP receive window of zero, saying, hold on a second. I don't have any buffer space available. Stop sending. And it turns out that, I mean, the system has to allow for that. It has to allow that a recipient, I mean, who knows what the receiver is doing. They may be spooling it off to a printer, or to mag tape in the old days, or waiting for something to happen. And for whatever reason they just - there's nowhere they can put any more data that the sender wants to send.

So they send back an acknowledgment of what's been received with a receive window set to zero. What that tells the sender is, stop. There is no room available at the other end for your data, so you can't send any. Now, TCP connections are full duplex, meaning that they inherently go in both directions. They don't actually have to be, interestingly enough. It's possible for, after the connection can be established, for one side to send a FIN packet and shut down the traffic in one direction. That's sort of an unusual case, but the spec does allow for it. But in normal case the TCP connection is full duplex.

So what happens is, if the receiver has sent an ACK packet saying I don't have anywhere to put anymore data, don't send it, then the sender has to stop. It cannot send anymore data. What it does is it sends something that's called a "window probe." A window probe is an acknowledgment of the last byte number sent by the other end. And what that does - oh, actually I'm wrong. It's been a while since I actually wrote this stuff, because I have written all this. The window probe is a deliberate mistake. It sends one byte less than the last one. It acknowledges one byte less, and that forces the other end to correct it. That forces an acknowledgment packet to be sent by the other end, and every acknowledgement packet contains the current TCP receive window.

So this so-called "window probe" is a means for getting updated information on the availability of buffer space by the other end. It's very clever. It's a little bit of a hack itself, I mean, like a legitimate hack of the way the TCP protocol works. But that's what our TCP stacks do. If you tell it that you have no available buffer, then you'll find yourself receiving these little ACK packets because it kind of wants to tickle you to get an update for when you do have data available.

The reason hackers have found all of this useful is it is a very nice way of capturing bad guys, like creating a black hole or a honeypot, essentially, that worms will fall into. If you've got a worm which is creating TCP connections, trying to scan the Internet, and it creates a connection to you, you can tell it, "Ah, hi there. Happy to connect; but, by the way, I don't have any buffer space." And what it does is it hangs the connection. The other end really can't let go because there's stuff that they're trying to send to you, but you've said, I can't receive it just now. Hold on.

And so very little bandwidth is consumed, yet we talked about the amount of resources consumed by every TCP connection. A TCP connection is expensive in terms of all of the counters and timers and buffers that need to be allocated in order to manage that connection. So with us doing almost nothing on the receiving end, we're able to get the worm to stall a connection. And if lots of people were to do that, it's possible to bring worms to their knees because the connections can't get done what they need to, yet they still have to allocate statically all of the data required to manage that.

Leo: Of course, it would bring them to their knees if they had knees.

Steve: If the worms had...

Leo: Worms do not have knees. I'm sorry, Steve, I have to point that out to you.

Steve: Do not have knees. Thank you. Glad you corrected me. So now we understand how we manage the question of how much we can send. That's with the receive window. The second part of this is how fast can we send it? Because nothing limits us, technically, except our own local connection to our ISP. That is, whatever it is, the bandwidth we're purchasing, that's a constraint. But we've got a long way to go, often, between we who are sending the data and where we're trying to get it to. Maybe something along the way can't handle as much as we want to send. How do we know?

So what TCP does is something called - it's called a "slow start." And this slow start is one of the problems with the way browsers work in terms of us getting pages loaded quickly. The problem is the TCP stack running in our computer doesn't know how fast it can send something. Now, what we're often sending with a browser is just a little query, so it might just all fit in one packet. And so the packet is a packet. There's no way to send a packet slowly. The granularity of the speed at which we send data is in the timing of our packets. Although I guess you could send smaller packets if you wanted to, like, if you had less to send.

Normally, as I mentioned a couple weeks ago, packets are as large as they can be because we are trying to send as much data per packet as possible. But the sender of our data, that is, the server we're making a query to, the query might be very small, but we might be asking for something big, a huge file, for example, or a very large and complex web page. The connection in the other direction has to figure out how fast we're able to receive data.

Now, we know that, if we send data too fast, we can overload routers along the way. And we also remember that routers have permission to drop packets. That's part of this funky packet processing scheme, the whole packet orientation of the Internet. Routers that cannot send data out of their outgoing interface because there's just too much, there's not enough bandwidth, like many incoming interfaces are receiving data that are all trying to go out the same one, for example. Routers have the permission by decree, by design, to just drop packets that they can't send. And remember, they never send us an indication because that would further increase a congestion problem. So the designers said routers make a best effort to send the data; but when they can't, they drop them.

So what does that mean for us as a sender, trying to send data as quickly as we can, that is, with as high a packet rate as possible? Well, it means we need to kind of creep up on it. And that's what TCP does. It means we just can't send the data as fast as we possibly can because something somewhere is going to collapse. And, boy, if the whole Internet worked that way, if everybody on the Internet sent as much data as they possibly could, as fast as they could, nothing would work. I mean, all the routers would be collapsing and having problems.

So by universal agreement, TCP being this very clever protocol, it does something called a "slow start." It's generally given permission to send some number of packets without acknowledgment. So the idea being, let's just get things moving here. And typically that number is two. And so when TCP starts, it can send two packets off, one after the other. Without having the first one acknowledged, it'll send the second. But then it waits. And then the rule for throttling is we allow the number of unacknowledged packets to increase by one for every acknowledgment we receive.

So let's think about that. We allow the number of unacknowledged packets to be increased by one for every acknowledgment we receive. So we first send off two packets as our agreed-upon start. They get acknowledged. So we have our first acknowledgment. We were allowing ourselves to send two. Now, with the receipt of this first acknowledgment, we increase that by one to three. So we can now send three more packets off without any further acknowledgment. When an acknowledgment comes back for whatever we've sent before, we increase that to four. This is known as the "congestion window." It's not a window that's ever sent on the line, that is, it's not like the receive window, which is 16 bits of the TCP header that tells us how much data we're able to send ahead. This one is - it's a window. It's basically a counter maintained by the TCP stack for the purpose of avoiding connection, the idea being that at some point this is going to break.

If we keep increasing the number of unacknowledged packets we're allowed to send by one every time we receive an acknowledgment, at some point we're going to reach a limit. And the beauty of this system is that it will, as we start trying to send packets faster than the weakest link, literally link, between routers, at some point we find the point where the weakest link breaks. It drops the packets we're trying to send because we're trying to send them too fast. So acknowledgments from the other end stop because the data is no longer getting through.

And what TCP does is, if it has failed to receive - and this varies in strategies. Over time, the strategy, the actual congestion avoidance strategy has varied a lot. There's names like Tahoe and Reno, and a whole bunch of other ones that you'll see if you do some Googling and Wikipediaing, which specific exactly what the behavior is. But the idea is that, when the sender realizes that its data is no longer getting through because it's missing acknowledgments, it cuts back its sending rate quickly. Typically, it divides it in half. So it dramatically scales it back, and then goes back to increasing it.

So essentially what this means is that losing packets is the signaling function for "We can't send the data any faster," and that the TCP senders at each end of a connection, all over the Internet, are always sort of - they're trying to go faster than the maximum speed that is available between the two endpoints, that is, that weakest link, wherever that is, and they're always pushing it to the limit. So given that there is a point somewhere that is weaker than their ability to send packets, they're going to find it because they will pump packets out. As long as there's data to be sent and they've got a high bandwidth connection, the sender will increase the rate of sending, that is, the number of outstanding packets, the packets that are allowed to be out there on the fly as acknowledgments come back, aggressively keeps moving that number upwards until it pushes it too far. Then it backs off a lot, and then again moves forward.

So this is what's actually going on between TCP connections which are, like, probably, I don't know what percentage, but the vastly greatest percentage of traffic on the Internet are over TCP connections. All of our operating systems down in the kernel, in the so-called TCP stack, have these counters. And when we're sending a file, when we're uploading a big file or we're receiving a web page, the server at the other end is doing the same thing. It's pushing, on an individual connection basis, as many packets that have not yet been acknowledged out as it can, increasing the packet rate until it hits the point where it starts to fail or stutter. Then it backs off, to sort of allow things to recover, and then starts working back up again.

And so that ends up being a sort of a self-throttling system which, given the constraints, I mean, it really seems kind of funky and crude. And you'd think, well, gee, couldn't they come up with something better than this? I mean, like, we go until we're going too fast, and we only know we're going too fast because the system starts to break, and then we back off, but start trying again. So, and again, it's exactly because of this that networking engineers have for years thought, okay, can't we be smarter? How about if we back off less? How about if we back off adaptively? How about if we have more memory than just this congestion window? How about if we have more history?

All kinds of things have been done, trying to, like, find that sweet spot where we're sending data reliably and as quickly as we can, yet also sensitive to changes. And remember that it's not like the connection between us and some local destination is also constant. Routers are inherently, as when I'm receiving a web page, I'm getting a burst of data. It's [sound], out it all comes, and here's the page. Then I look at it for a while, click on a link, making another request, which creates another burst of data. So there's these bursts that are happening in a web page model. And certainly that's the case with images being downloaded and pages being displayed. So it's all very burst-y, which means that, if anyone has ever sat in front of a phone system with lots of lines, you'll notice, I don't know why this is, but they all start to ring at the same time. Have you ever noticed that, Leo?

Leo: Yeah, yeah.

Steve: It's like, it's weird. I mean, it's the strangest thing. Either I'm just noticing the pattern - but I've talked to people, and it's like they've noticed it, too. And, like, receptionists have said it's the strangest thing. It'll be just, like, quiet for a long time.

Leo: And then all at once.

Steve: A line will ring, and then another one and another one. And they don't seem in any way related.

Leo: Life is burst-y. Life is burst-y.

Steve: Yes. And so you can imagine, here's the plight of this router that's sitting around, like with all this extra bandwidth to spare. And suddenly all this data comes pouring into it that it's got to try to send out. Well, it may not be able to. So the state of the connection between endpoints is also varying with time because these packets are having to move across routers whose own traffic may be burst-y. And what they were able to handle a minute ago, they're no longer able to handle now because there's another burst coming through some other interface. So all of this has to be dynamic and adaptive. And, amazingly, it works as well as it does. It's just incredible.

And now that we know all this, in two weeks we can talk about SPDY. We can talk about what Amazon - Amazon. Well, yeah, what Amazon and the Kindle Fire are doing with their cloud-based browser enhancement which originated with the Chromium project, which is Google looking at how can we make the whole web faster. And, boy, they have succeeded, like 58 percent improvement.

Leo: Without changing underlying protocols, which is, I think, very interesting.

Steve: Yes, exactly.

Leo: Yeah. Well, I'm glad we got that out of the way. So you say two weeks because next week we're going to do a Q&A. We do feedbacks every mod 2 episode, so every even episode. So if you have a question for Steve about this or any topic having to do with security, or science fiction, or eReaders - I think that's your portfolio - you go to...

Steve: Oh, by the way, I have to say, Leo, there was a wacky bit of blurb that a bunch of people tweeted me. It turns out that, when you load an eReader full of books, it gets a little heavier.

Leo: Yeah, I know. Yeah, I saw that, too.

Steve: Oh, my god.

Leo: Your joules weigh something.

Steve: 10^-18 femtograms or something. I mean, absolutely. It's the electrons. There's more electrons.

Leo: You can calculate it. There's more electrons. You can calculate it. We know the weight of the mass of an electron, and we can easily calculate.

Steve: They don't weigh very much.

Leo: No, they don't. But there's a lot of them. So...

Steve: They do add up. They do.

Leo: Yes, to something. Somewhat less than a thimbleful. GRC.com/feedback for questions, suggestions, comments. We'll go through a few of those next week. GRC is a good place to go, of course, to find SpinRite, the world's best hard drive maintenance and recovery utility.

Steve: Yay.

Leo: Yay. All his wonderful tools, many of which, most of which, with actually the sole exception of SpinRite, are free. Although there will be a new commercial product emerging soon from GRC.

Steve: I wish soon. But started soon. Started soon. Yeah, I've got to get the Off The Grid project finished first. And that's almost done, by the way. It's all functional and working, as I told everybody it would be by this week. It is. After I let my newsgroup folks play with it, they came up with some really useful improvements to the UI, where it was a little - some things that I was assuming weren't really clear. So I'm reengineering that. But it's essentially done. I'm going to have a few other web pages, related web pages to catch up on. But it's there.

Leo: Yay. What else? So there's lots of freebies on the site, plus 16KB versions. Steve has both audio versions and transcriptions. We have audio and video, not the transcriptions, not the 16KB versions at TWiT.tv. And you can watch us do this show Wednesdays, 11:00 a.m. Pacific, 2:00 p.m. Eastern, that's 1800 UTC at live.twit.tv or TWiT.tv. We keep both alive. Either one will work. I think that is everything I need to say. Don't forget to sign up for the barbecue. Click that banner at TWiT.tv for our Ford barbecue coming up on the 13th. And Steve, I'll see you next time on Security Now!.