One Reason of Many
Regarding Why
Net Neutrality is an Impossible
&
Unworkable Concept

Authored By: James M. Aloye

Someone at the FCC got it right – finally: http://arstechnica.com/tech-policy/2014/07/fcc-republican-says-net-neutrality-rules-too-onerous-for-isps/

However, this is just one person, so it begs the question "Why do the other FCC commissioners have differing opinions?". Fortunately, that question has a very easy answer, and that answer is "the material involved is extremely technical which makes it very difficult to digest". Even smart people may take a while to fully be able to wrap their mind around it due to its complexity.

Eventually the technology and facts of "how stuff works" will force them to agree with the one guy in the FCC who actually understands the technical nature of the Net Neutrality situation.

I will explain it here so that you can understand it too.

Before I begin, let me say that I built and ran the AloyeNET dial-up Internet Service Provider (ISP) back in 1998 until 2004. It was a national ISP that competed with AOL, CompuServe, and other smaller ISP's. In the height of its heyday, AloyeNET had less than 10,000 dial up customers which was significantly less than my larger nationwide competitors. However, given my background, I fully understand what's involved in getting my internet customers data to travel from my ISP network onto the internet and to all the places that data needed to go.

I can tell you first hand that net neutrality is ridiculous and unworkable because applications and the internet don't work that way and I will explain why.

However, before I do, let me quickly define the term Net Neutrality so everyone knows what it means.

Net Neutrality is the concept that all internet traffic must be given the same level of priority so as to prevent pay-for-preferential-treatment from occurring for any one type or multiple types of traffic, including traffic generated by, traveling to, or received from any subset of internet customers or content providers.

That means Email Traffic, Web Browsing Traffic, Database Traffic, Video and Multimedia Traffic, and so forth cannot get preferential pay for speed treatment.

Hopefully that definition of Net Neutrality makes sense to you.

It sounds simple enough, but I argue that is not possible because applications don't work that way.

To illustrate this concept, imagine the FAA required that all vehicles (Airplanes, Cars, Trolleys, Trains, Busses, etc.) registered their flight plans with the FAA and traveled at the same speed, in the same sky, and interacting with the same airports.

You would laugh and say "Car's don't fly, neither do trains.", while that is true, that response doesn't get to the heart of the matter in a technically accurate manner.

However the explanation below does:

Cessna Planes and Boing 747 Jumbo Jets have different minimum altitude to air speed requirements in order for them to "stay in the air" and not crash. If you require smaller and lighter Cessna planes to fly at the same speed a Boing 747 must fly at, the Cessna planes will crash from going too fast.

They have different speed requirements to stay operational than Boing 747's do. Conversely, Boing 747's will crash if they are forced to travel at the same speed as a Cessna because 747's have minimum speed requirements that are much faster than Cessna planes have due to their weight. If they go slower than the minimum required speed at certain altitudes they will crash.

- Think of software applications or web applications as being the Boing 747 or the Cessna.

- And think of the minimum required air speed as being the application's minimum latency requirement for operating properly and not crashing or corrupting data.

With that in mind, let me use a slightly different example to illustrate the point.

The internet wire is similar to the road. There are a certain number of lanes, and by nature certain cars will merge onto the road before other cars either by position on the road, or as a result of one driver taking a more or less aggressive merging posture than another driver.

The routers or switches that your computer connects to can be viewed as an on-ramp, and each road has millions of on-ramps in this example.

Now, you already know that a car is a single object, it either merges or doesn't. But application traffic is not like a car. Application traffic is more like a 10000 cars daisy chained together or a train. Right now, in our example, switches and routers let anywhere from 1 to 10 of those cars (or DATA PACKETS) merge at one time, and then they have to wait a little bit and let another driver/conductor (or APPLICATION'S) cars (or DATA PACKETS) merge, and then the original Application can resume merging its next set of 1-10 DATA PACKETS, before it has to wait again and let others merge for a bit. Eventually all 10000 data packets will get past the router or switch and on to the internet.

The problem is, if too long a wait time occurs between the time the application's 1st of 10000 data packets merge and the very last data packet merges, many applications will "break".

Now, imagine that you are using a mission critical application for heart surgery. Your application is hosted and run from a datacenter in Cleveland but your hospital sits in Los Angeles, so you are using the application in the operating room in the Los Angeles hospital.

For the sake of argument let's assume this particular heart surgery application has a maximum latency requirement of 10 milliseconds between the end user interface and the dataset/API/or middleware layer. This means when you enter data, if it takes the data longer than 10 milliseconds to travel across the internet, the application will hang or lock up or crash and you will have to keep re-entering the data until it takes the data less than 10 milliseconds to get to its destination.

Also, the application has several layers or hops, so once you enter the data and see it on the screen the screen still needs to send the data to a database, which has a separate maximum latency of 32ms in this example.

So if the front end of the application cannot send the data to the database on its back end in 32 milliseconds or less, you will have to re-enter the data and start all the way back at the beginning again.

Hopefully this gives you a better understanding of how networks and data work.

Now, let us assume Net Neutrality is law and is being actively enforced by the FCC. If there is a high volume of data traffic that day, it means the heart surgery application may not complete its communication across the internet within the required 10ms or 32ms time period. This means the critical application fails and someone could die.

That is the whole reason why internet Service Providers (ISP's) offer packet prioritization services at an extra cost for companies who use applications that are important or critical. The ISP has to spend a lot of money and resources to build out their infrastructure to operate in a way that meets their customer's application speed requirements. They also have to pay for and set up data transmission quality of service agreements with other internet infrastructures that make up the internet between their network and the destination network, so that all the networks in between also make sure your application data travels at speeds within the minimum required latency.

Another example of how this works would be of a marketing company that uses a lot of video and voice. Let's call the marketing company, Company M.

Company M makes a lot of large videos and they use audio/video applications which require a minimum data communications latency of 20 milliseconds to function. Company M needs a guarantee for traffic related to the marketing application, but they don't care about speeds for traffic related to email communication or general web traffic.

Therefore Company M pay's their ISP a fee to configure an entire series of packet prioritization parameters at the ISP and also sets up Quality of Service data transmission contracts with all the other upstream ISP's (and has to pay each one of them for their time and for their engineers to build out and implement the QoS parameters) who's networks your traffic will travel across once its leaves your home ISP.

If Company M does not pay extra for a traffic latency guarantee for their application, they could have a lot of application outages or extremely slow response times for their application. If net neutrality prevents them from paying for a quality of service guarantee for their specific critical application traffic, the nation as we know it will be unable to continue as we know it.

That is why Net Neutrality has been struck down and has to be 'revised' and that is why it will never happen, unless some kind of engineering breakthrough occurs that alters the manner in which network routers and switches, or software applications work. Until that happens, if Net Neutrality becomes law, it will be un-implementable and will be continually struck down.

All this will continue to waste taxpayer money because the people making the laws have no idea or understanding of the technical details of the matter and therefore don't realize that the laws they are making are no less ridiculous as a law that requires pigs to flap their wings and fly to mars on their way from the slaughterhouse to the grocery store.

Because this is very technical and only a few people understand it and can laugh at it, the FCC folks and legislators have no idea how silly they look.