Evan Nichols

Dr. Perry Alexander

The Origins of the Internet

30 October 2012

In 2011, The International Digital Centre predicted that in 2011, 1.8 zettabytes (equivalent to 1.8 trillion gigabytes) of information would be created and replicated. 1.8 trillion gigabytes. According to Mashable.com, that’s equivalent to 200 billion HD movies, each 120 minutes long. Or equivalent to every person in the world sending three tweets per minute, non-stop, for 26,976 years straight. #whoa. That’s a lot of hashtags.

With the exponential growth of technology in the 21st century, it’s easy to get caught up in the hype of “new:” drive to your nearest apple store, buy the newest iPad and swipe away, playing Fruit Ninja and reading iBooks to your heart’s content. But many people, probably safe to say *most* people, don’t know where the technology that drives their iPads, iPhones, and Xbox 360’s originated from.

My initial expectations for learning about the “origins of the Internet” were to take a look into hard code, to see how programs took data and processed it through. I was surprised to discover the Internet’s foundation lay not in hard code—we discussed very little actual programming syntax—but in philosophy and basic mathematics.

Englishman Alan Turing (1912-1954) first pioneered the concept of an automated computing machine, whose “Turing Machine” paved the way for modern computer algorithms. Turing was one of Britain’s most valuable code breakers during WWII. Along with his cryptography work, Turing also studied the fields of biology (morphogenesis) and artificial intelligence (the “Turing Test”). The machine’s concept is simple, as Turing described in his 1948 essay “Intelligent Machinery:”

“...an unlimited memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings.”

In even simpler terms, the reader “head” looks at a symbol, and makes a decision to either move left, move right and write another symbol. It may seem like too basic a concept to define technology as we know it now, but consider what all programs and computers do: they take some starting value and command, and carry out a string of actions based on that input.

Of course, the Turing machine alone does not cover all the necessities of a computing system. For a computing system—any system, really—to work, a “formal system” needs to be put in place. A formal system can be broken down into four main parts:

**Syntax:** this covers the alphabet and grammar. You define the symbols you will use in your language, and also how you can use them.

**Inference System:** If I know “A” and A-🡪 B then we known B is true. This part allows the system to solve problems and relationships logically; these are the basic concepts of propositional logic.

**Axioms:** something so simple it cannot be disproven. Axioms can be consider a necessary evil: they are the building blocks of a system, but they’re “bad” because you can’t prove them to be true (e.g. you can’t prove A=A without outside information). Axioms are taken as fact.

**Semantics:** This gives symbols meaning. In terms of math, we understand that two, parallel, horizontal lines are an “equals” sign. We know that an “x” or a “” are multipliers. We know this because we chose to give those symbols their specified meaning.

Another one of the concepts important related to computing is a concept you probably don’t even know you’re familiar with. Remember the last time opening up Safari, and, after a few clicks, the “rainbow wheel” popped up? And it just stayed there, spinning and spinning and spinning, and you couldn’t decide whether to let it spin more or quit?

You might not have realized it, but that was a perfect example of “the halting problem.” The problem asks if, when given a program and an initial input, whether that program will halt, or run forever. In 1936, Turing proved that there is no *one* algorithm that can universally decide the halting problem. So when the rainbow wheel pops up on your screen, give it just one more second. It might just work.