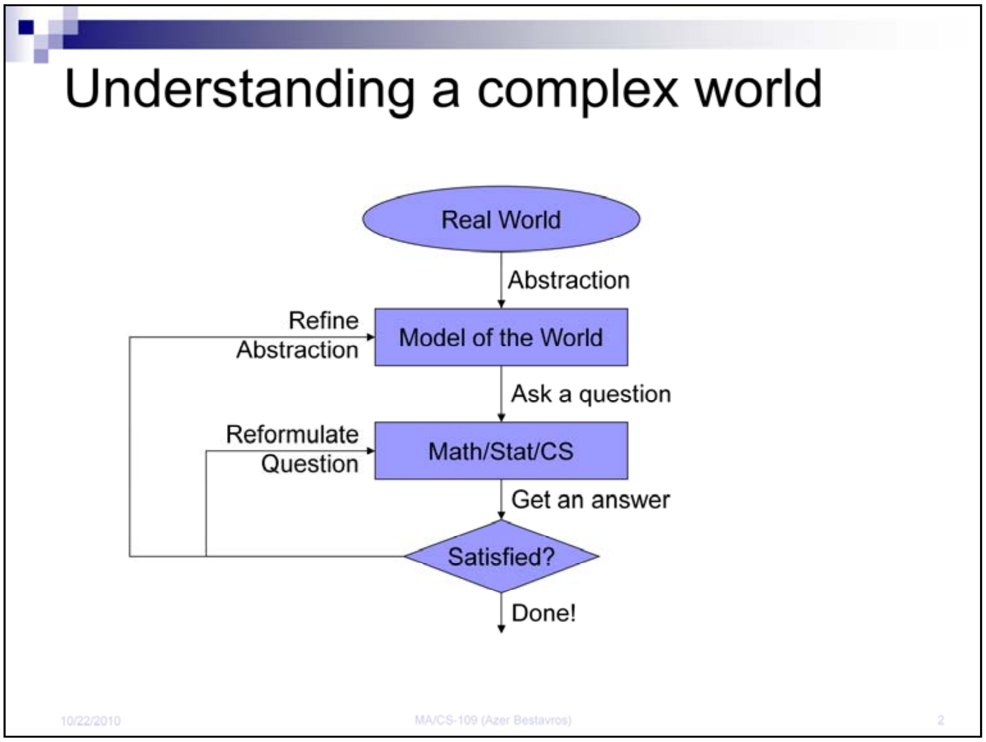
MA/CS-109: The Beauty of the Beast

Azer Bestavros



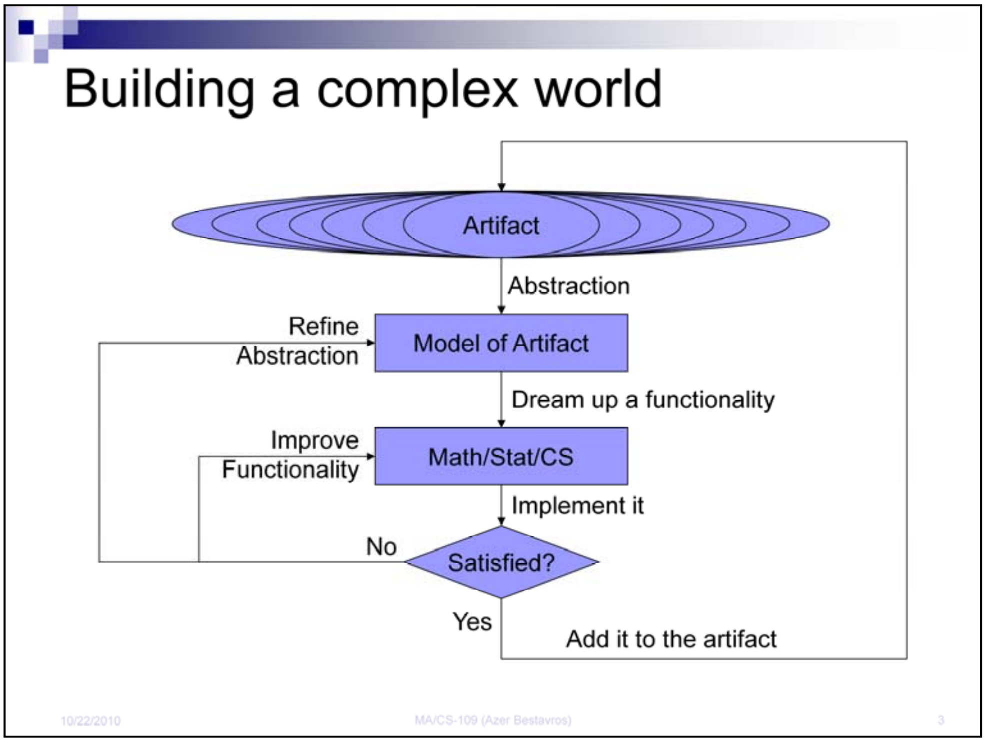
So far in this course, we have focused on how mathematics and computer science enable us to understand complex phenomena in the world around us. We did this by following a simple process:

We start by building a model (a simplification or an abstraction) of the complex world that we want to understand.

Next, we pose a question for which we want an answer – e.g., how fast will the H1N1 virus propagate within a particular population.

Next, we use approaches from mathematics or CS to reason about the model, and hopefully find an answer.

If we find an answer to the question, we are done. Otherwise, if we cannot find an answer, or if the answer does not make sense, we either reformulate the question and try again, or else we go back and come up with a better model of the world.

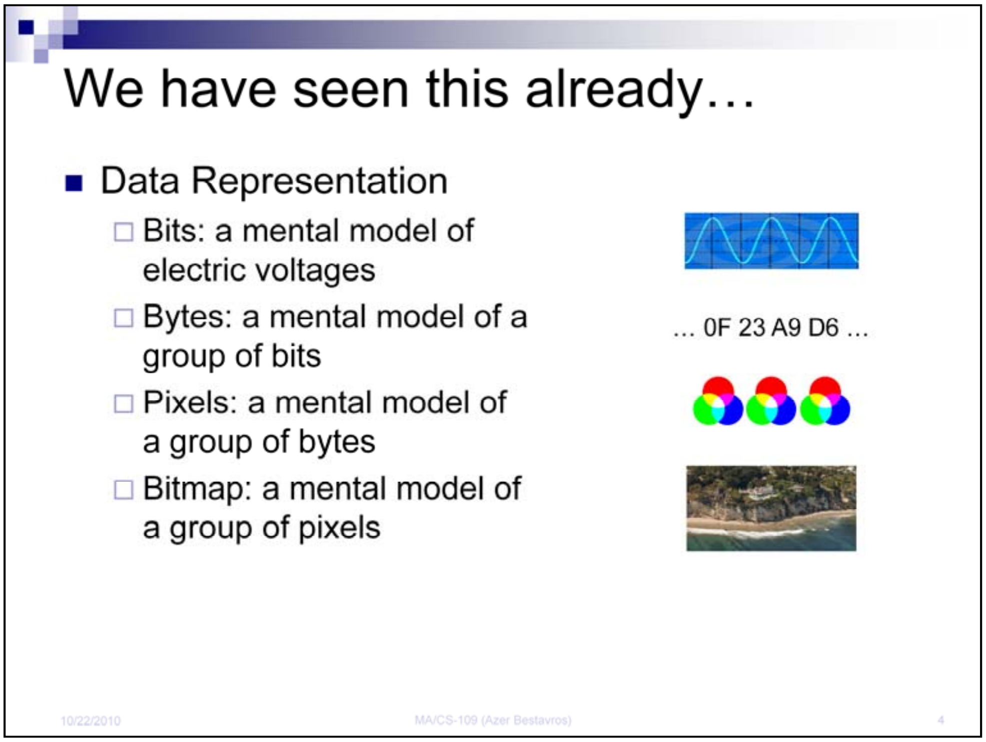


The process of “abstracting” the details of a “world” is at the heart of our ability to understand the complexities of that world. Interestingly, this same process of abstraction is at the heart of our ability to build ever more complex worlds (or systems). We do this by incrementally building complex artifacts.

We start with a very simple artifact that does a well-understood simple/abstract function.

Now we dream up a new capability that we want our world to support, and we use MCS to implement this new capability. If the functionality we implemented is to our liking, we deploy it in our world, which is now a bit more complex than before (thanks to this added capability). If the functionality is not to our liking, we can improve it or else modify our vision of the new functionality, and we repeat.

This process of building up more complicated artifacts by repeatedly abstracting an existing artifact and adding new functionality is not new!



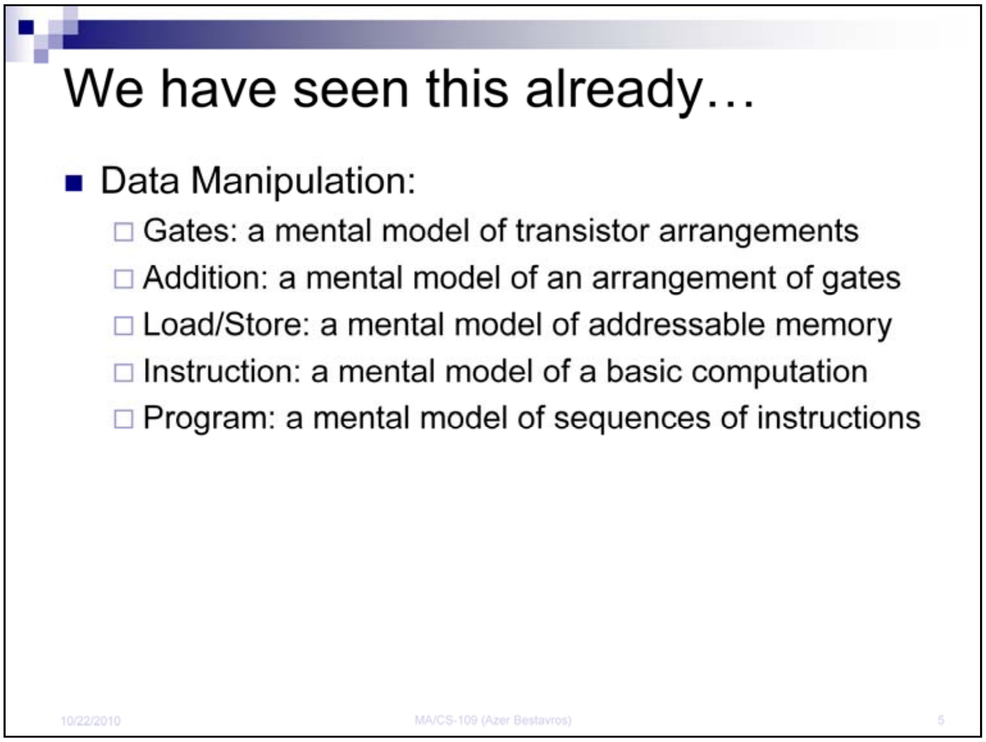
In particular, we have seen this already when we looked at how “images” are just bits…

We started with a very simple device that can be in one of two states (e.g., based on whether there is electric current flowing in a wire or not as is done with transistors, or based on the polarity of a magnetic field as is done on a hard disk, or based on whether a surface is reflective or not as is done on a CD or DVD, etc.)

We abstracted the capability of a device to be in one of two states by a single “bit” that can be either 0 or 1. Notice that the abstraction makes us not worry about how the “0” or the “1” is implemented -- whether as the presence and absence of electric current, or the polarity of a magnetic field, etc. -- which means that we do not have to be experts (e.g., in electric or optical engineering) to be able to reason about what we can do with a bit…

Next, we dreamt a new functionality, that of representing a character. For that we figured out how to store and retrieve a group of 8 bits (which we called a byte), and we added that capability to our world, which is now more complex as we can now think about a device that can store 256 different values.

Next, we dreamt a new functionality, that of representing color. For that we used a group of 3 bytes for a single pixel. And, from that we got to a bitmap image, and from that to a video stream as a sequence of images, etc.



Also, we have seen this already when we looked at how one can do computation.

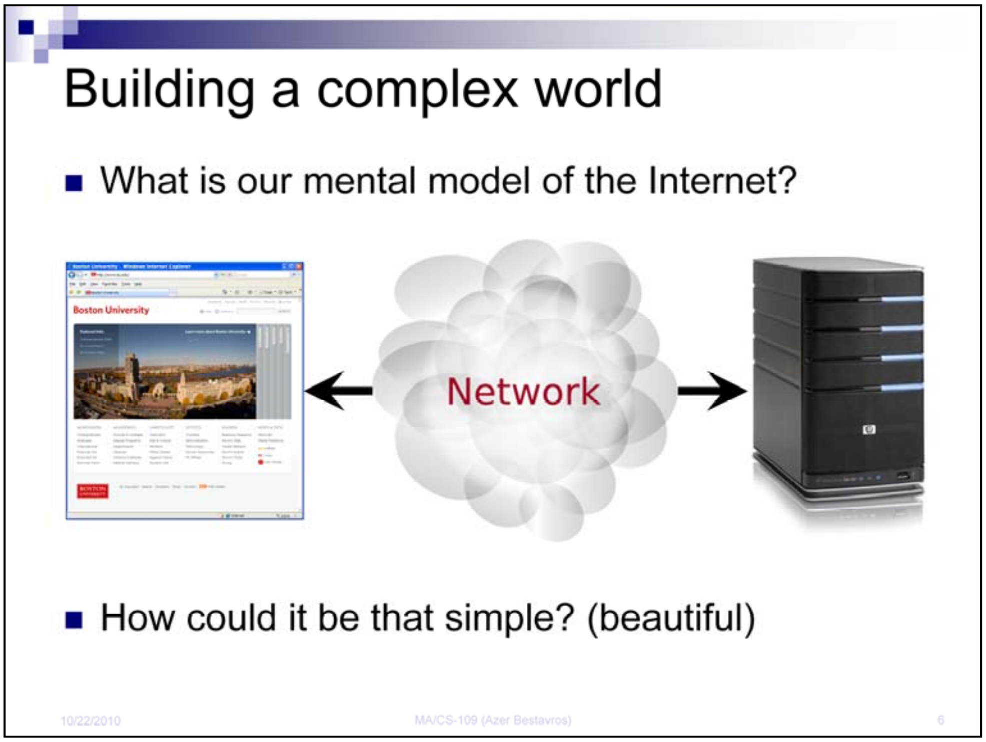
We started with a very simple device (namely two types of transistors) and we abstracted their functionality by thinking about them as switches.

Next, we envisioned a new functionality – that of doing some simple operations on one or two bits (e.g., inverting a bit, or computing some logical functions on two bits). We implemented these operations using the abstraction of a switch, and added the new set of devices to our world.

Next, we abstracted these devices by thinking about them as functions on bits – namely the NOT, AND, OR, NAND, NOR, and XOR functions.

Next, we envisioned a new functionality – that of doing arithmetic on integers (represented as bits), so we figured out how to do so using the repertoire of abstractions we have so far, …

And the process of inventing new capabilities using abstractions of previously introduced capabilities continues!!



We will do this one more time to see how we were able to build the Internet…

Before we see how the Internet is built up, it is important to ask what it looks like to us as users. What is our "mental model" of the Internet?

In fact, our model of the Internet (as users of the Internet) is of a simple "wire" that allows information from one computer to flow to another. For example, the Internet allows a (program on a) user's computer (shown on the left) to request information (e.g., a specific web page) from another computer (shown on the right), and it allows that second computer to send that information back to the first.

That's it… If you are using the Internet to access content such as web pages, music, and movies, this simple (and quite elegant) "abstraction" is all what you need. It's beautiful! This is exactly why even grandma can use the Internet ☺

But, how could it be that simple? How are we able to provide this beautiful abstraction?

Well, let's see how the process of inventing new capabilities using abstractions of previously introduced capabilities allows us to think of the Internet as a "wire"…

Nothing new under the sun!

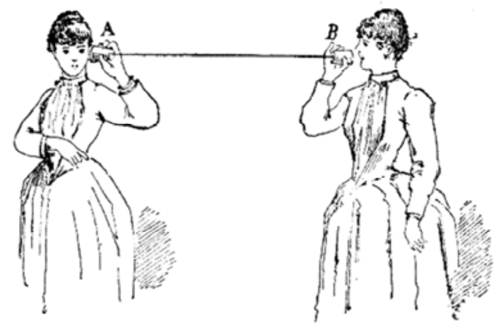
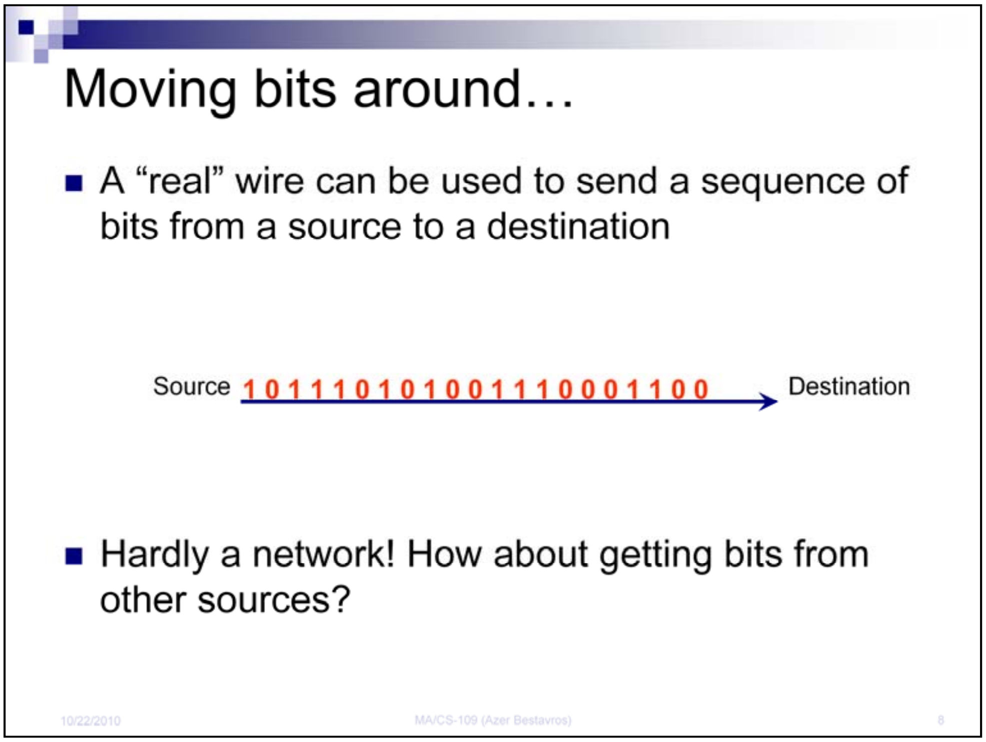


FIG. 76. Trådtelefon.

The 19th century 'tin can', or 'lover's' telephone
Source: Wikipedia

It started well before the Internet ☺



Let's start with the most basic of wires – a "real" wire (as in an electric wire, or a beam of light).

Such a wire would allow us to get a bit from a source (a sender) to a destination (a receiver). And, since any information we want to communicate is "just bits" -- recall, the mantra "It's all (just) bits" from previous lectures – a real wire is good enough to carry any information from one sender to one receiver.

But this is not good enough! The Internet enables connectivity between many, many, many (billions) of senders and receivers.

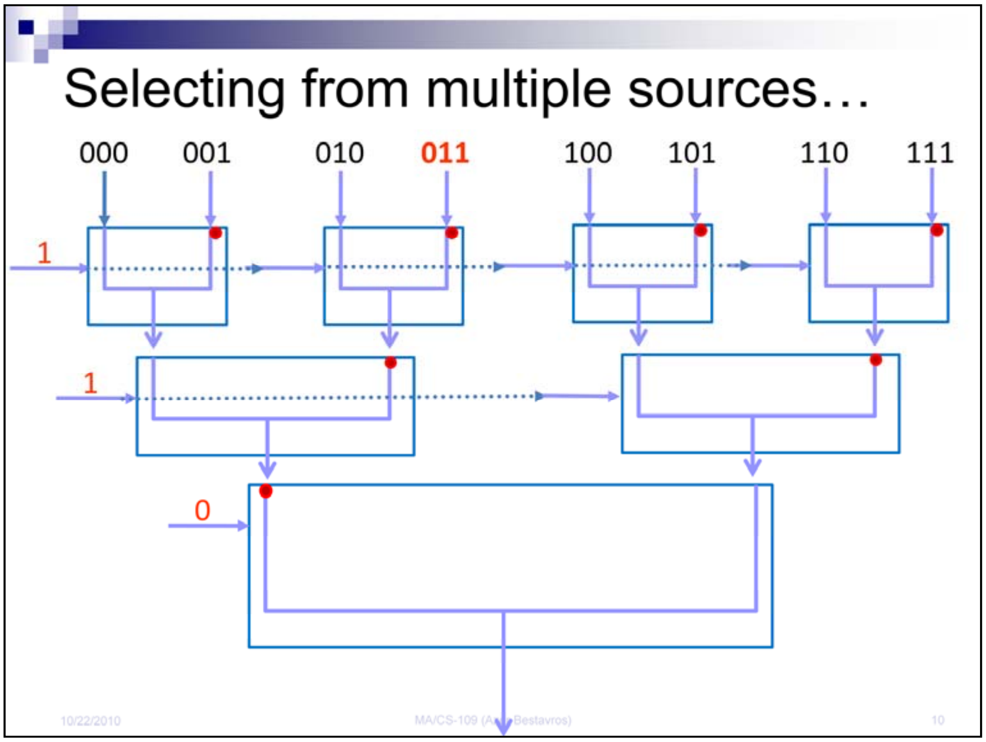
A simple "real" wire will not do it (you can't just run a wire between every pair of computers!)

So, now we think of a slightly more complicated capability that we may want to support.

How about allowing a receiver to get content (information) from any one of a number of possible senders?

Can we build something to do that much?

How many wires do we need?



Indeed, we can, and we have seen how to do so already.

Recall that to access data (i.e., bits) in memory we saw how we can do so using **multiplexers**. A multiplexer connects N senders (sources) of bits to a single receiver (destination). Each sender has an “address”. This address is nothing more than a set of bits identifying the sender (much like the mailbox number in a post office) and by providing this address to the multiplexer, the multiplexer simply “connects” the addressed sender to the receiver.

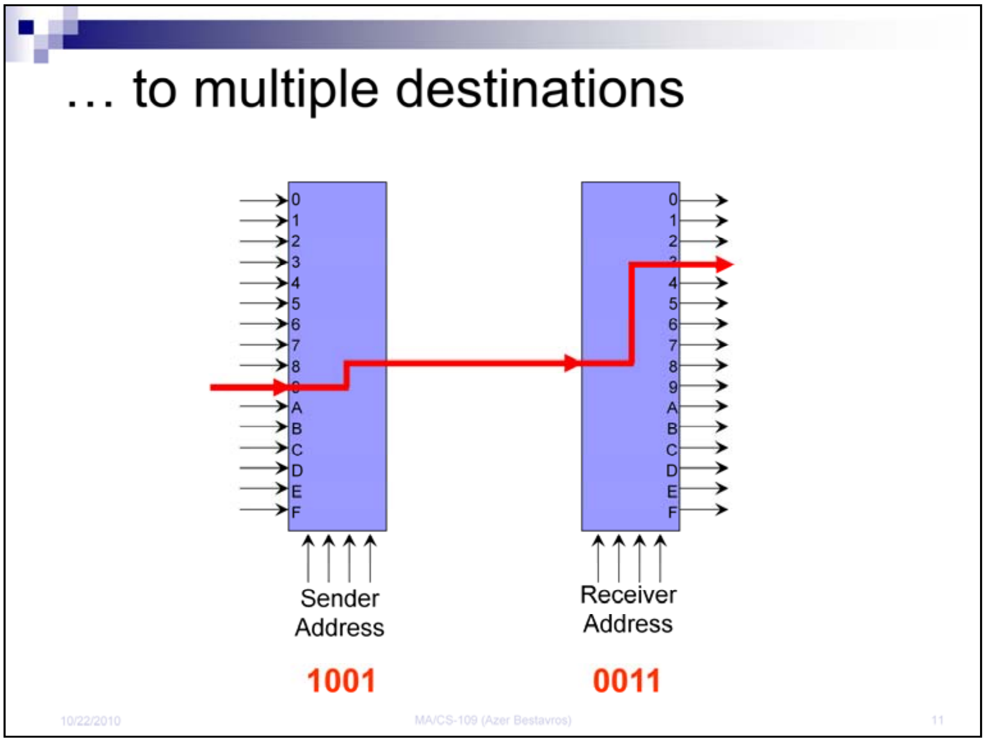
The function of the multiplexer is shown above when N=8.

For 8 senders, we need 3 bits to represent their addresses (from address 0 to 7 – and we know how to write these down in binary as shown on top).

Thus, the multiplexer will have 8 sources, one destination and a 3-bit selector (these are the wires on the left-hand-side). By putting the address of a sender on the selector wires, we can connect that sender to the receiver.

Great! We now have a network that connects any one of N sources to exactly one destination.

Check your understanding: What about N much bigger than 8? What if N is a million? How long would the address of each sender be? How many levels of multiplexing (i.e., selection lines) do we need? What is the name of that “function” (from number of senders/locations to the number of bits needed to address them)?



Well, building an entire network to connect millions of senders to one recipient is not exactly our idea of a useful network. What we really want is a network that could connect any one of any senders to any one of many (not just one) recipients.

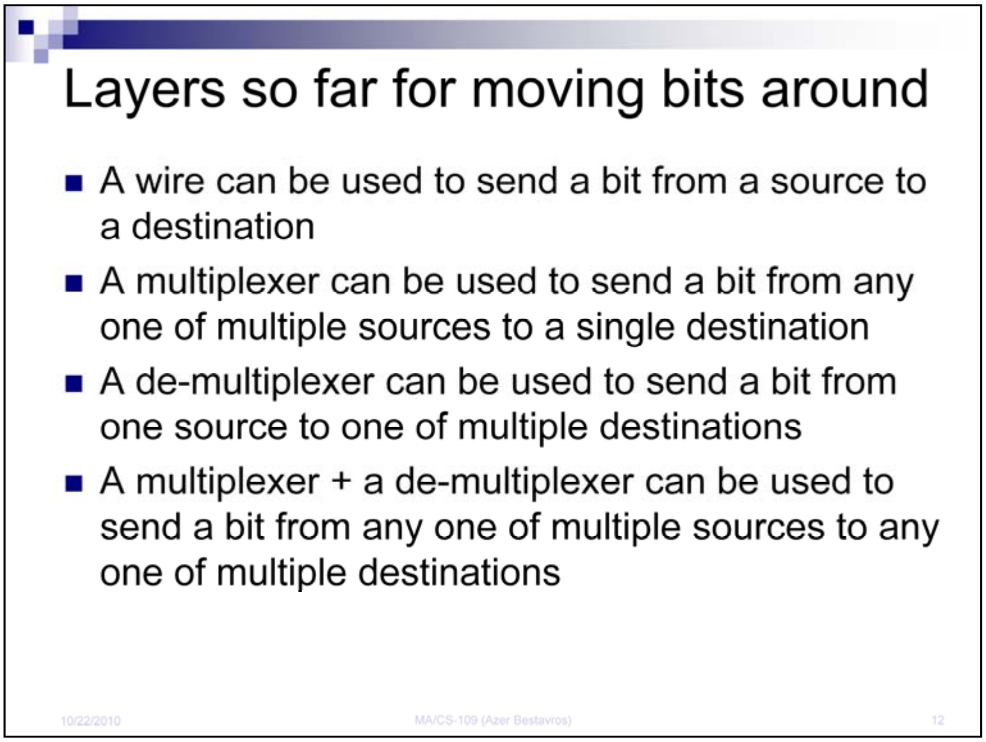
Well, just as we could build a multiplexer to go from N senders to one receiver, we can also build a device that does the exact opposite – i.e., it goes from one sender to N receivers. This device is called a “de-multiplexer” and you will just have to believe us that it can be built ☺

Like a multiplexer, a de-multiplexer has some selection lines that allow it to select which one of its N destinations should receive the bits sent from its one source.

By connecting a multiplexer’s destination (output) to a de-multiplexer’s source (input) we can connect any one of many senders to any one of many receiver.

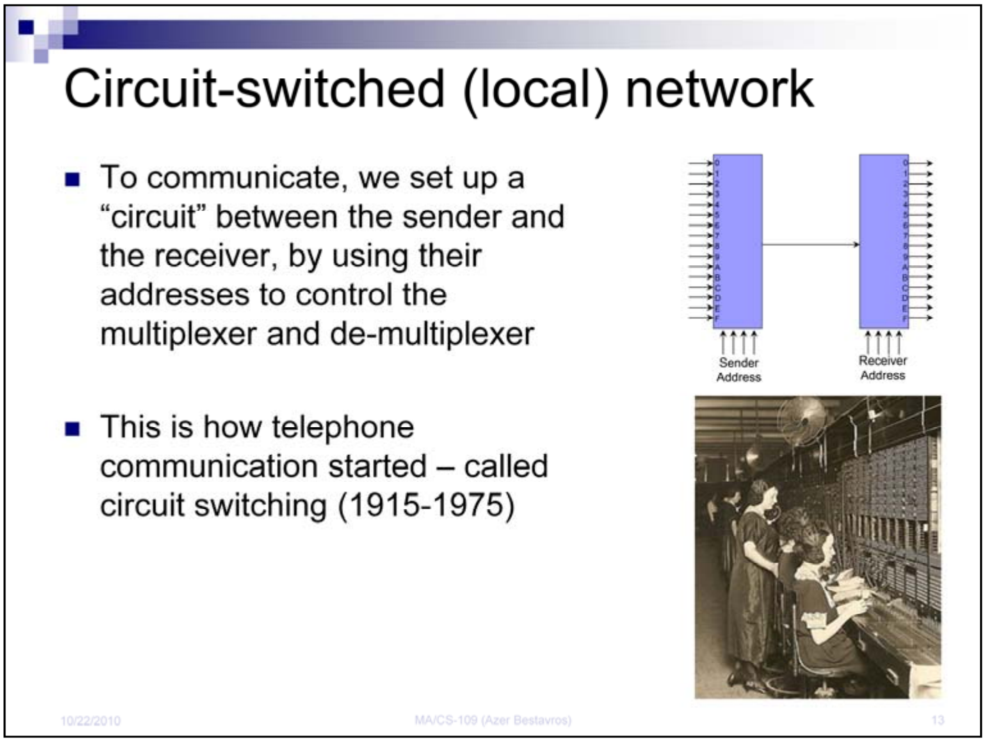
In the above illustration, we are able to connect 16 sources to 16 destinations, and by putting the right sender and receiver addresses on the selectors of the multiplexer and de-multiplexer, respectively, we can make any one of the sources communicate with any one of the destinations.

Voila!



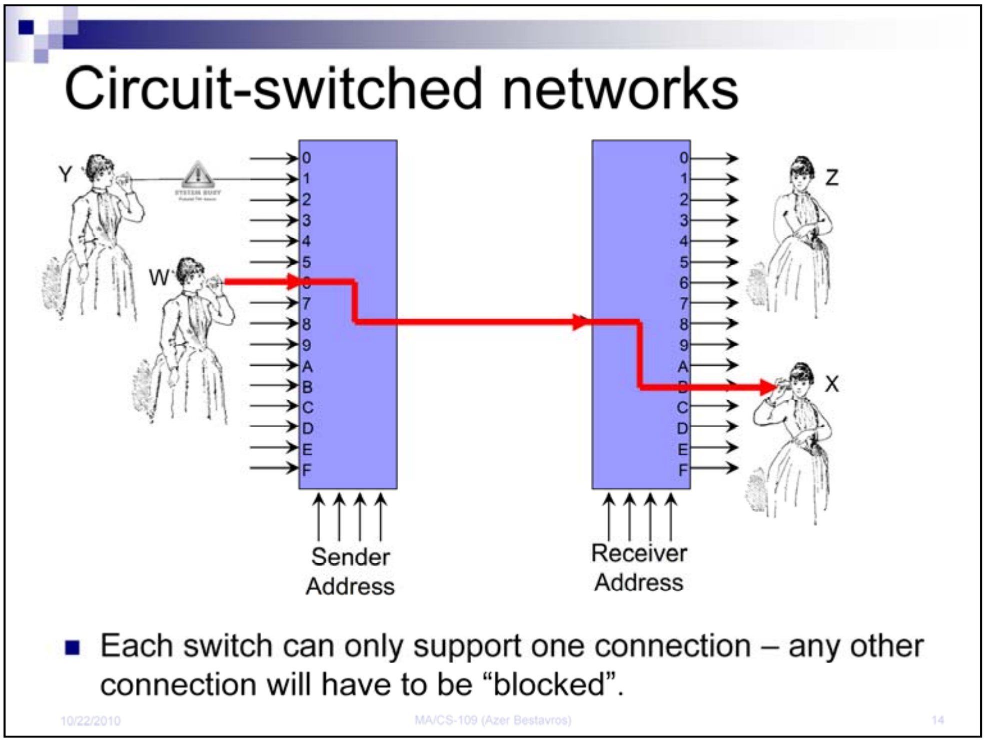
So, let’s take stock of what we did so far…

- A wire can be used to send a bit from a source to a destination
- A multiplexer can be used to send a bit from any one of multiple sources to a single destination
- A de-multiplexer can be used to send a bit from one source to one of multiple destinations
- A multiplexer + a de-multiplexer can be used to send a bit from any one of multiple sources to any one of multiple destinations

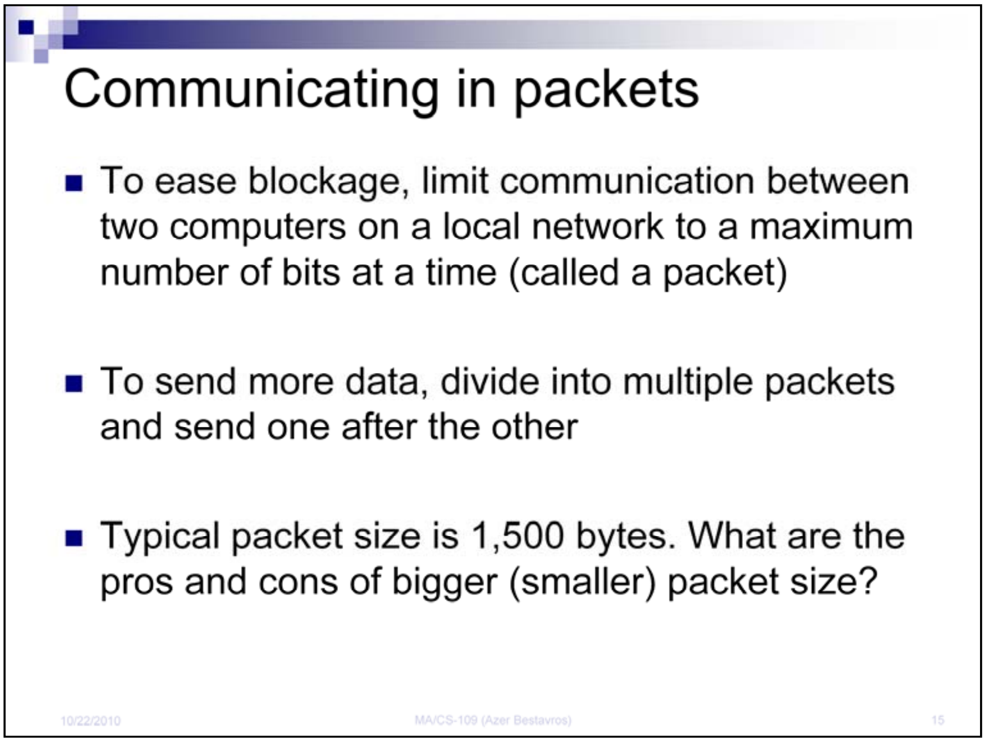


The abstraction (functionality) we built so far is precisely what switchboard technology was able to achieve in the early 20th century, except that humans (operators) were used to connect senders to receivers through "panel exchanges". The first Panel exchange was installed at the Mulberry Central Office in Newark, New Jersey. It was placed in service on January 16, 1915 (and the last one in Maine was decommissioned in 1975).

By connecting a sender to a receiver directly (through switches or multiplexers and de-multiplexers) is called "circuit switching" because we ultimately link the source to the destination with a "circuit" (another fancy work for a wire).



One problem with circuit switching is that it underutilized the capacities of the switches. Each switch can only support one connection – any other connection will have to be "blocked".

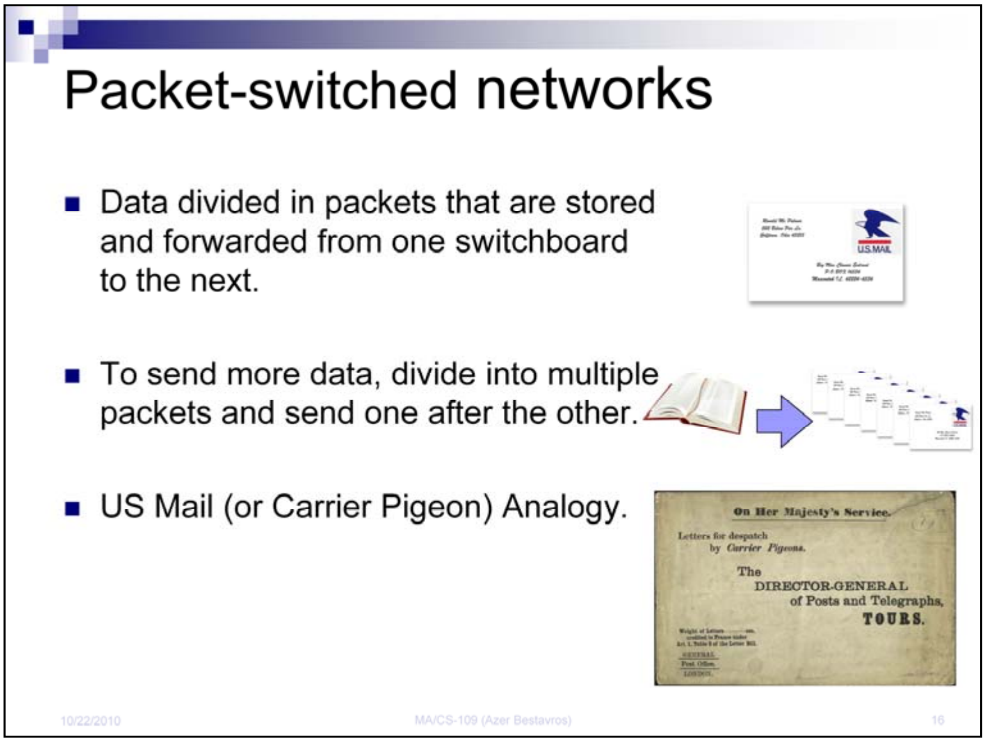


Circuit switching suffers from a big problem: If a sender and connected to a receiver through a set of switches (multiplexers and de-multiplexers), then no other senders or receivers connected to these switches can talk.

That's not practical because one connection between a sender and a receiver may last a very long time during which other potential connections cannot go through (you experience this when you make a phone call and you get the busy signal, or a message that "all lines are busy").

How can we mitigate this problem? Well, let's think of adding yet another functionality. Rather than allowing a source to send information as bits *ad nauseam*, we package the information in groups of bits, each is called a frame or a packet of a predetermined maximum size (typically 1,500 bytes on the Internet). To send information (e.g., a web page or a movie), we package that information into one or more packets and we send these packets one at a time. So, at any point in time, a source-destination pair can only hold the switches for no more than the time it takes to send one packet (which is not much at all – typically much less than a millisecond).

Check your understanding: How many packets would be needed to send a tweet? How many packets would be needed to send an image of resolution 3,000 x 2,000 pixels, where each pixel can be any one of 16 million colors?

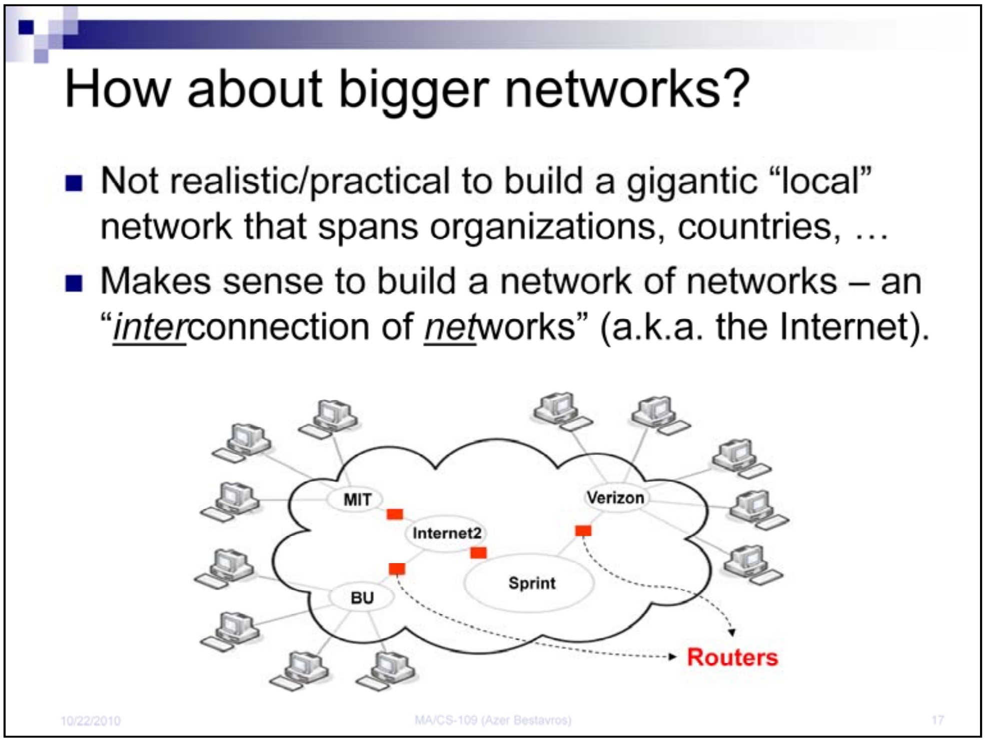


Circuit switching suffers from a big problem: If a sender and connected to a receiver through a set of switches (multiplexers and de-multiplexers), then no other senders or receivers connected to these switches can talk.

That’s not practical because one connection between a sender and a receiver may last a very long time during which other potential connections cannot go through (you experience this when you make a phone call and you get the busy signal, or a message that “all lines are busy”).

How can we mitigate this problem? Well, let’s think of adding yet another functionality. Rather than allowing a source to send information as bits *ad nauseam*, we package the information in groups of bits, each is called a frame or a packet of a predetermined maximum size (typically 1,500 bytes on the Internet). To send information (e.g., a web page or a movie), we package that information into one or more packets and we send these packets one at a time. So, at any point in time, a source-destination pair can only hold the switches for no more than the time it takes to send one packet (which is not much at all – typically much less than a millisecond).

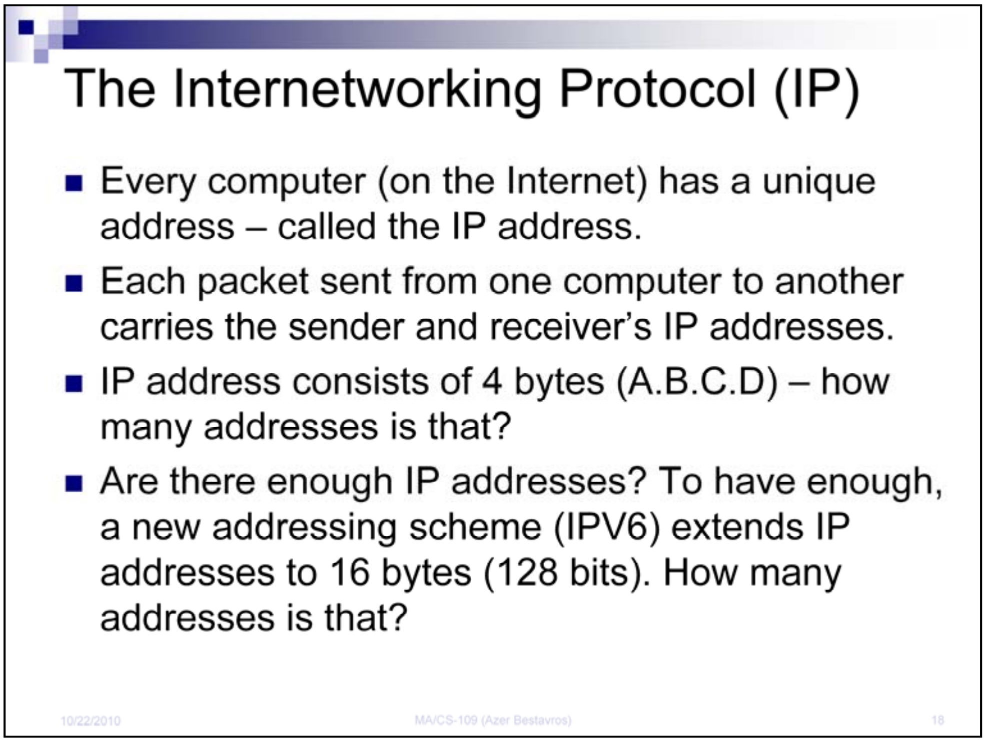
Check your understanding: How many packets would be needed to send a tweet? How many packets would be needed to send an image of resolution 3,000 x 2,000 pixels, where each pixel can be any one of 16 million colors?



So, far we built enough functionality (and associated abstractions) to allow us to send/receive packets of information between sources and destinations, all of which are connected **directly** to a big switchboard.

While good enough for a small group of people, this does not work beyond a “local community”. It is not realistic/practical to build a gigantic “local” network that spans organizations, countries, … where every computer can be connected to every other computer **directly**.

So, now, with our repertoire of abstractions, we need to add new functionality to allow packets to be send from a source to a destination by going through intermediaries (which we will call “routers”). This abstraction has it roots in how we send letters through the post office. A letter is not delivered directly from the hands (mailbox) of its sender to the hands (mailbox) of its receiver, but rather it goes through intermediaries.



This is precisely the next capability (functionality and associated abstraction) that we build up for the Internet. It is called the Internetworking Protocol (IP).

Every computer (on the Internet) has a unique address – called the IP address. IP address consists of 4 bytes denoted by (A.B.C.D), which allow us to have unique addresses to ~ 4 Billion computers (also called "hosts").

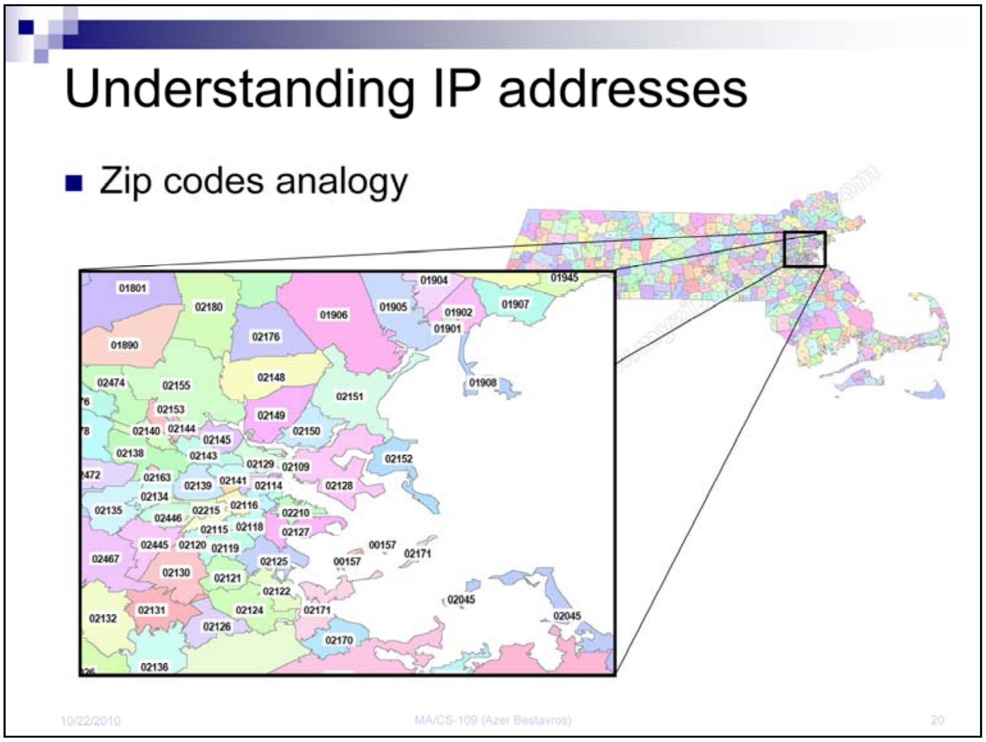
Each packet sent from one computer to another carries the sender and receiver's IP addresses

Side Note: Are 4 billion computers enough? Clearly not, especially if we want to connect computers to everything around us. Think about the number of computers per capita and think about wanting to connected them all into a global network! This is why a "new" version of the Internetworking protocol (called IPv6) allows the IP address to go to 16 bytes (128 bits). Since IPv6 addresses are 128 bits long, the theoretical address space if all addresses were used is 2^{128}, which when expanded out is 340,282,366,920,938,463,463,374,607,431,768,211,456, which is about $3.4*10^{38}$ addresses. That's about 340 trillion, trillion, trillion addresses. To grasp how large this number is, consider that it is enough addresses for many trillions of addresses to be assigned to every human being on the planet (that's a lot of computers per capita). The earth is about 4.5 billion years old. If we had been assigning IPv6 addresses at a rate of 1 billion per second since the earth was formed, we would have by now used up less than one trillionth of the address space! The earth's surface area is about 510 trillion square meters. If a typical computer has a footprint of about a tenth of a square meter, we would have to stack computers 10 billion high blanketing the entire surface of the earth to use up that same trillionth of the address space.

IPv4 address examples [Demo]

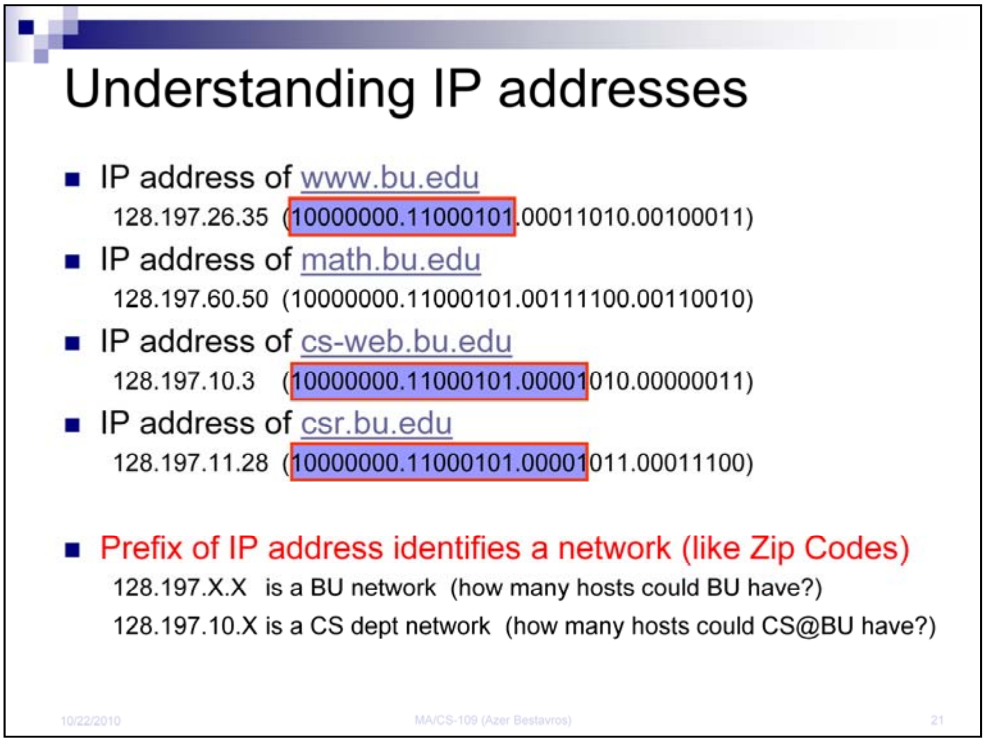
- IP address of www.bu.edu
  128.197.26.35 (10000000.11000101.00011010.00100011)
- IP address of math.bu.edu
  128.197.60.50 (10000000.11000101.00111100.00110010)
- IP address of cs-web.bu.edu
  128.197.10.3 (10000000.11000101.00001010.00000011)
- IP address of csr.bu.edu
  128.197.11.28 (10000000.11000101.00001010.00011010)
- IP address of www.facebook.com
  69.63.181.12 (01000101.00111111.10110101.00001100)

One can lookup the IP addresses of various hosts as illustrated in the examples given. One way to do this is using the service available at: http://www.kloth.net/services/nslookup.php (we will learn more about DNS and name lookups in later lectures and labs).



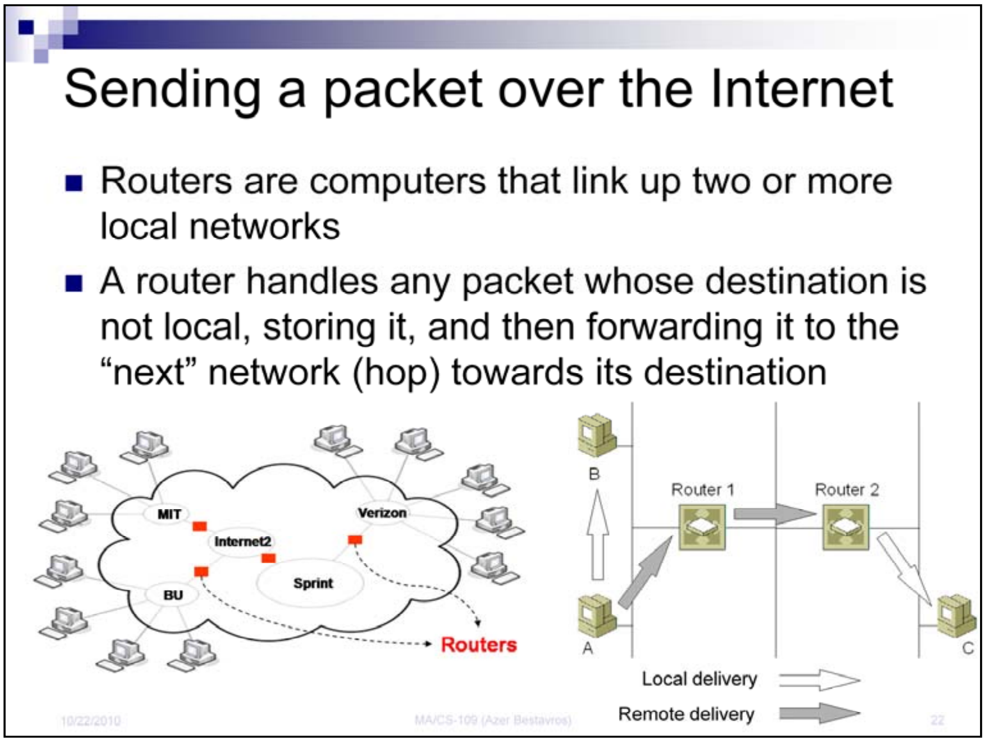
The prefix of a zip code also allows us to map "geographically" where a specific address is on the map – for example when we see a zip code starting with "01" or "02" we know that it is somewhere in the Northeast. Zip codes starting with a "9" are in the west.

Just like zip codes, the prefix of a any local network address gives us information as to where that network is, except that the "location" of the network is not geographic (as with zip codes) but rather it is the location within the hierarchy of networks on the Internet. For example, the CS network has IP prefix "128.197.10". A prefix of that is "128.197" which the BU network, which tells us that the CS network is within the BU network.



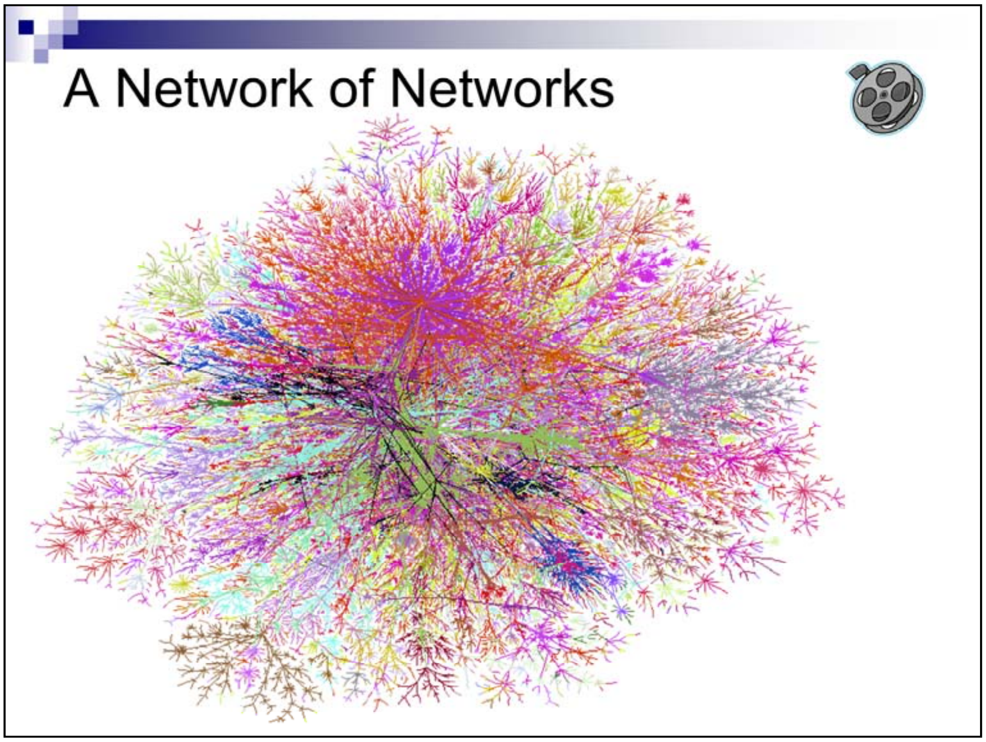
One can think about an IP addresses as consisting of two parts – a prefix that identifies (uniquely) the local network on which the computer resides, and then the remaining suffix which identifies (uniquely) the computer on that local network. A good analogy here is that of US mail addresses – the street number identifies a particular mailbox/house within a local group of houses who share the same zip code (zip codes would be analogous to the IP address prefix).

The examples shown above illustrate this.



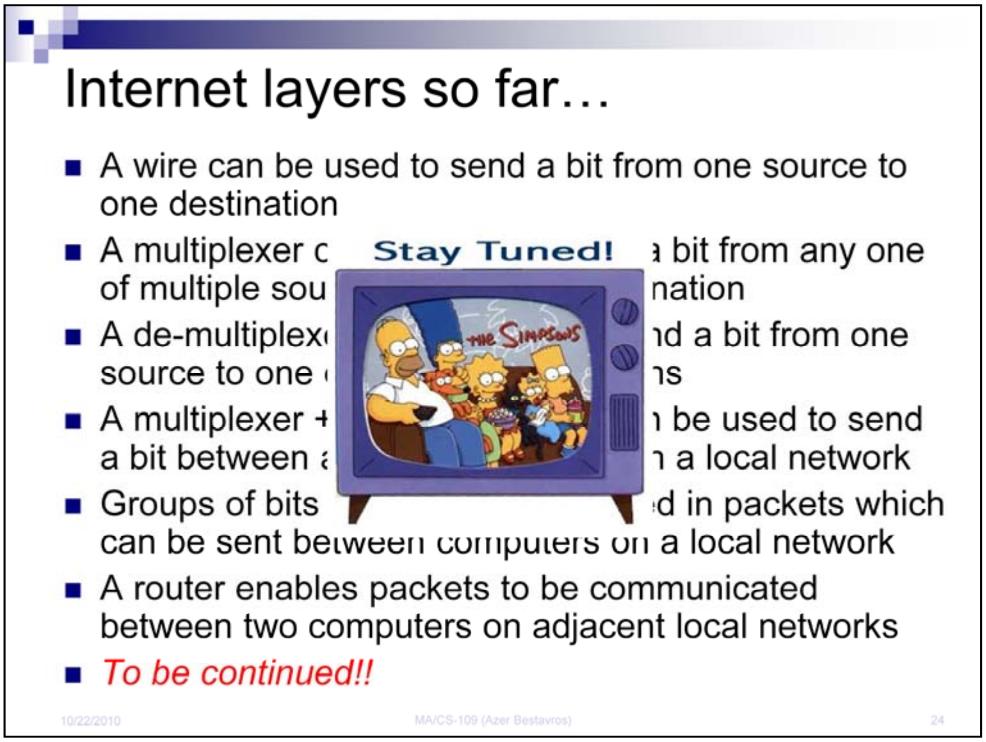
Routers are computers that link up **two** or more local networks. Routers are distinguished from hosts in that hosts (computers on the edge of the Internet such as your notebook computer or iPhone) are only connected to a single (local) network.

A router handles any packet whose destination is not local, storing it, and then forwarding it to the “next” network (hop) towards its destination.



That’s what the Internet looked like in the early 2000s (it’s many times bigger right now!) Each line in the picture represents a connection (wire) between two machines on the Internet (e.g., two routers or a computer and a router).

A nice video explaining the Internet being a network of networks, and the necessity of having agreements for it to operate (agreements are what we will call protocols in later lectures) is available at:
http://www.cs.bu.edu/~best/courses/cs109/modules/internetpeering



So far, through repeated abstractions and addition of functionalities, we are able to build up the following Internet layers :

1. **Physical Layer**:
    - A wire can be used to send a bit from one source to one destination.
2. **Link Layer:**
    - A multiplexer can be used to send a bit from any one of multiple sources to a single destination.
    - A de-multiplexer can be used to send a bit from one source to one of multiple destinations.
    - A multiplexer + a de-multiplexer can be used to send a bit between any two computers on a local network.
    - Groups of bits (~1.5kB) are arranged in packets which can be sent between computers on a local network.
3. **Network Layer:**
    - A router enables packets to be communicated between two computers on adjacent local networks.
    - *A set of routers enable packets to be routed between any two computers on the Internet… Stay Tuned for that!*