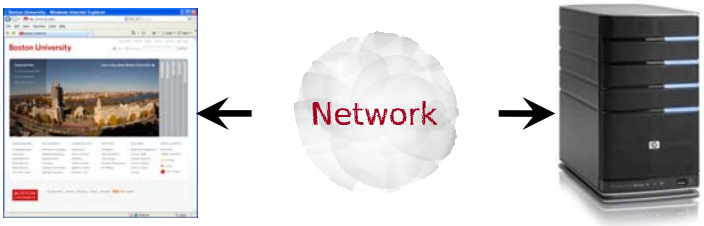

The Sequel
MA/CS-109:
Beauty of the Beast
Internet Protocols
Azer Bestavros

Building a complex world

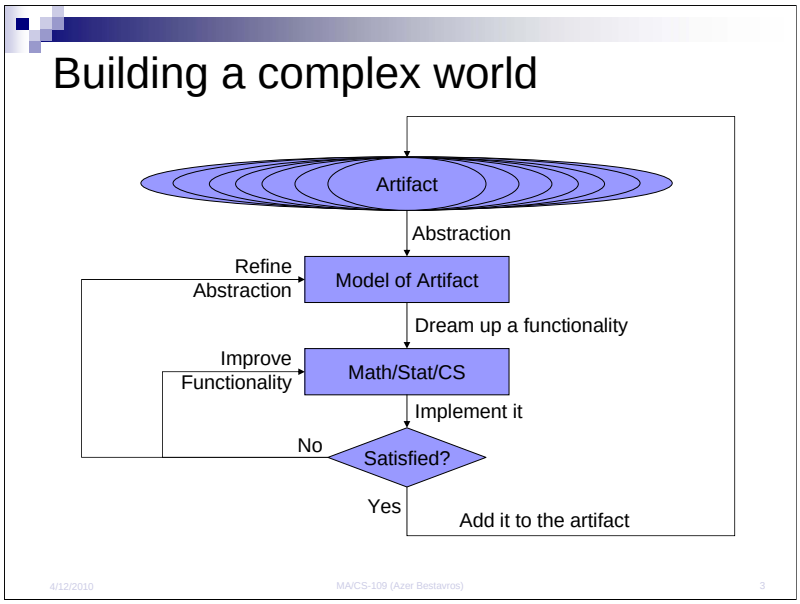
- What is our mental model of the Internet?



- How could it be that simple? (beautiful)

4/12/2010 MA/CS-109 (Azer Bestavros) 2

In our attempt to understand how the Internet is built up, we followed a process (of building up more complicated artifacts by repeatedly abstracting an existing artifact and adding new functionality).



Recall that the process of “abstracting” (out) the details of a “world” is at the heart of our ability to understand/comprehend 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 from simpler ones.

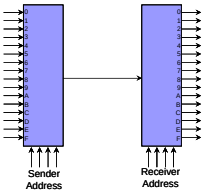
We start with a very simple artifact that does a well-understood simple/abstract function.

Now we dream up a new capability that we want our world to support, and we use concrete methods (e.g., algorithms) 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.

Let’s review what we did so far (following the above process) with respect to the Internet.

Circuit-switched (local) network

- To communicate, we set up a "circuit" between the sender and the receiver, by using their addresses to control the multiplexer and de-multiplexer
- This is how telephone communication started – called circuit switching
- If two computers on a local network are "connected" others cannot communicate – they are blocked



We built a circuit switched (local) network out of simple hardware such as multiplexers and de-multiplexers…

Packet-switched networks

- Data divided in packets that are stored and forwarded from one switchboard (router) to the next. Typical packet size is 1,500 bytes.

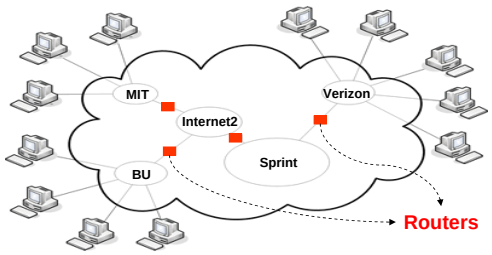
- To send more data, divide into multiple packets and send one after the other.

- US Mail (or Carrier Pigeon) Analogy.

Next we realized the need for packet communication between two local computers (two computers on the same local network)…

How about bigger networks?

- Not realistic/practical to build a gigantic “local” network that spans organizations, countries, …
- Makes sense to build a network of networks – an “*inter*connection of *net*works” (a.k.a. the Internet).



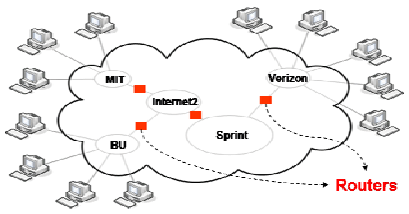
Next we realized the need to interconnect local networks together…

The Internetworking Protocol (IP)

- Every computer (on the Internet) has a unique address – called the IP address, which consists of 4 bytes (A.B.C.D)
- Each packet sent from one computer to another carries the sender and receiver's IP addresses
- US Post analogy

For that we built up the Internetworking Protocol (IP)

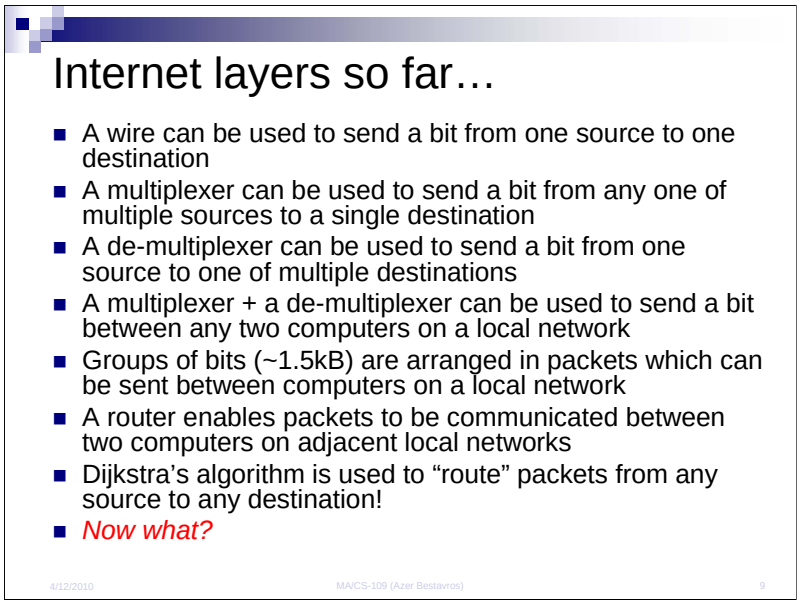
IP packet routing



- A router handles any packet whose destination is not local, storing it, and then forwarding it to the “next” network (hop) towards its destination
- Thanks to Dijkstra, routers can figure how to “forward” packets along the “shortest path”.

Recall that routers are computers that link up two or more local networks. A router handles any packet whose destination is not local, storing it, and then forwarding it to the “next” network (hop) towards its destination.

Using Dijkstra’s shortest path algorithm, routers can figure out how to forward packets along the shortest path.



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.

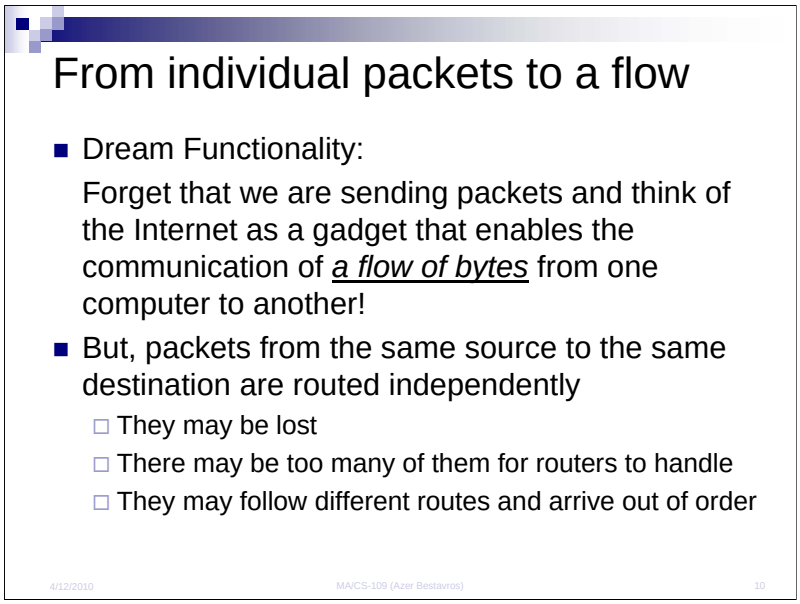
2.Link Layer:

- 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.

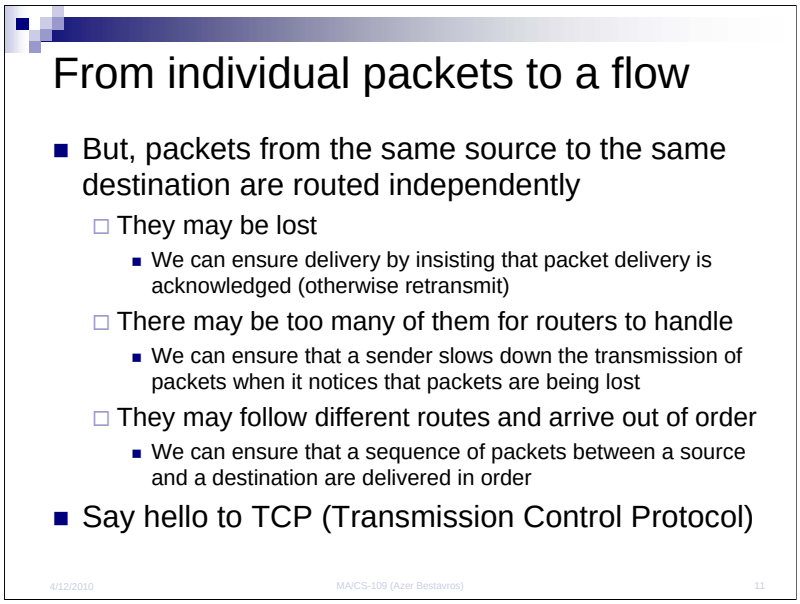
3.Network Layer:

- 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…

To the above three layers, in this lecture, we will add to more layers: the **Transport Layer** and the **Application Layer**.



To think about communication in terms of “packets” requires significant work to recover from packet losses, to make sure the network is not congested by an avalanche of packet transmissions, and to make sure that packets are delivered in order. Thus, it would be nice if we implement a functionality that gives us the illusion of a network that does not drop packets, that does not get congested, and that delivers packets in order.



Solving these problems can be done by making sure that the sender (and receiver) follow a specific protocol – called the TCP (Transmission Control Protocol).

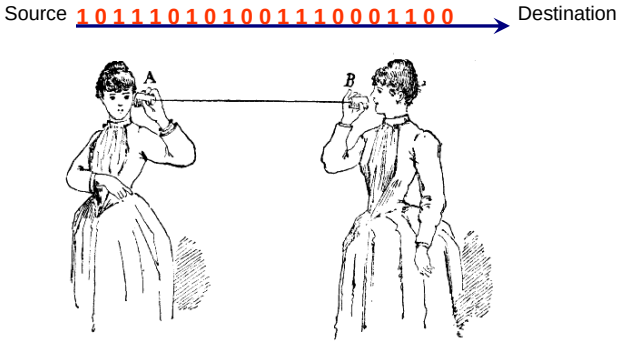
TCP deals with packet losses by insisting that the receiver acknowledges (to the sender) receipt of each packet. If an acknowledgment of a packet is not received within some timeframe (e.g., 10-100 milliseconds) then the sender will retransmit that packet.

TCP deals with network congestion by making sure that whenever the sender realizes that packets are being lost (see above), it “decreases” the number of packets it injects in the network per unit time (i.e., it slows down its sending rate). Also, when the sender realizes that there are no losses, it “increases” the number of packets it injects in the network per unit time. Typically, a sender cuts its sending rate by half when it detects congestion and increases its sending rate by a constant when it detects no congestion. This is called the “additive increase multiplicative decrease” rule of TCP.

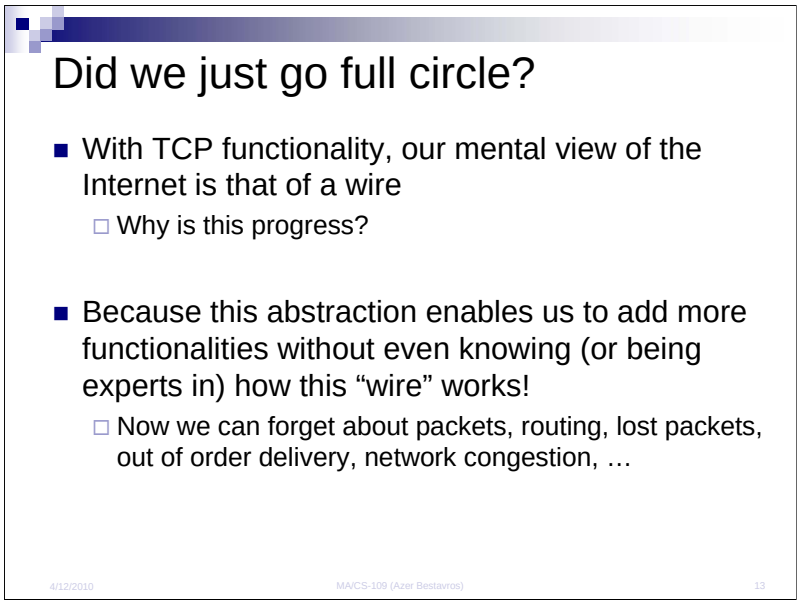
TCP deals with out of order delivery by giving each packet a sequence number. On the receiving end, packet delivery is delayed until all packets with a lower sequence number are received.

Our new mental model

- A sender (source) sends as much data (bits) as it wants to a receiver (destination)….



The TCP functionality allows the sender and receiver to have the illusion of a “wire” that carries information which is delivered reliably (nothing is lost), in order, and without ever overloading the network.



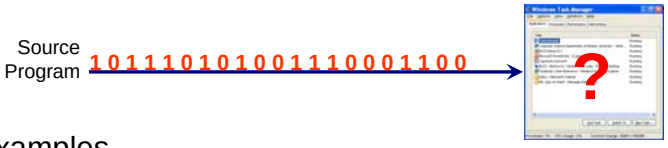
So, it seems that we have reinvented the wheel! We started with a wire and went full circle! Why do all that?

Notice that while our mental model is that of a “wire”, in reality that wire is not a physical wire, but a wire implemented by having many layers of functionalities.

Having an abstraction of a wire is useful because it allows the communicating parties (the sender and receiver) to not concern themselves with details of how the wire is implemented. And, since having physical wires between any two computers is not feasible, we ended up implementing the wire as we have seen using local switching networks, packets, IP routing, and TCP.

What can we do with a “TCP” flow?

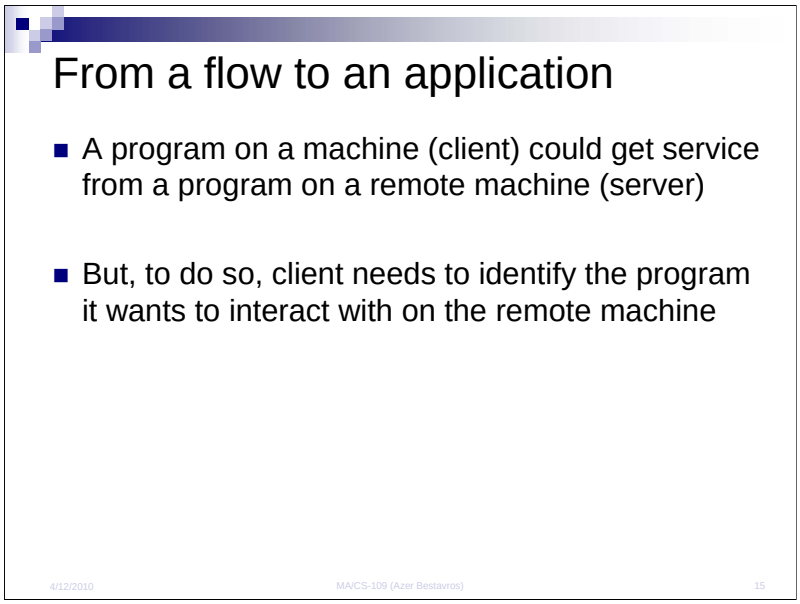
- Allow a program on one machine to interact with (send/receive bits to/from) a program on another



- Examples
  - Interaction between a web browser and a web server
  - Interaction between two bittorrent peers
  - Interaction between two skype programs

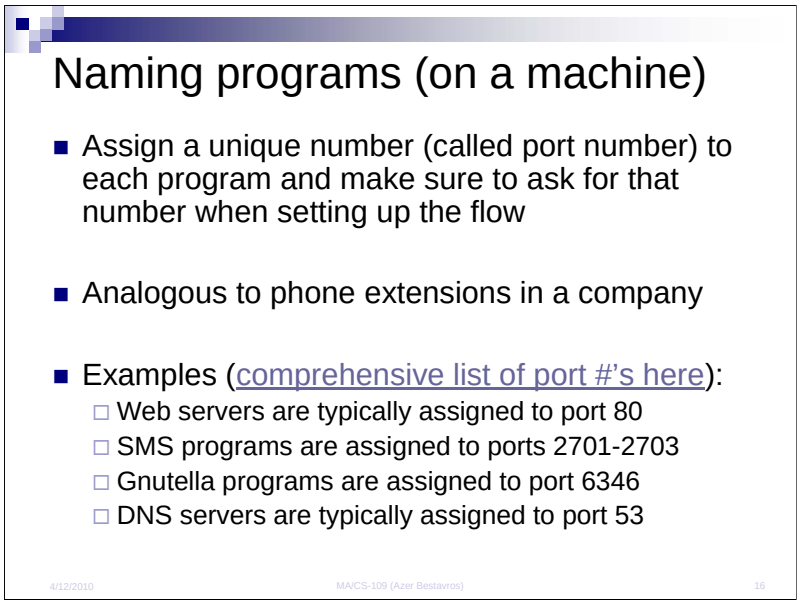
Using TCP, two programs on two different computers can “talk” to each other just as if there were a wire connecting one to the other.

Examples of programs that need to talk to one another include: a web browser and web server (the browser asking the server for web content, and the web server delivering that content), two bittorrent peers in a peer-to-peer file sharing system (one bittorrent peer requesting part of a file from the second, and the second sending such content back if available), etc.



In all of these examples (and many others) we have a program on one machine (client) requesting service from a program on another machine (server) which responds to the request. This is the so-called “client-server” model of interactions between computers on the Internet.

For the client program to be able to “connect” to the server program, the server program must have a unique identity.

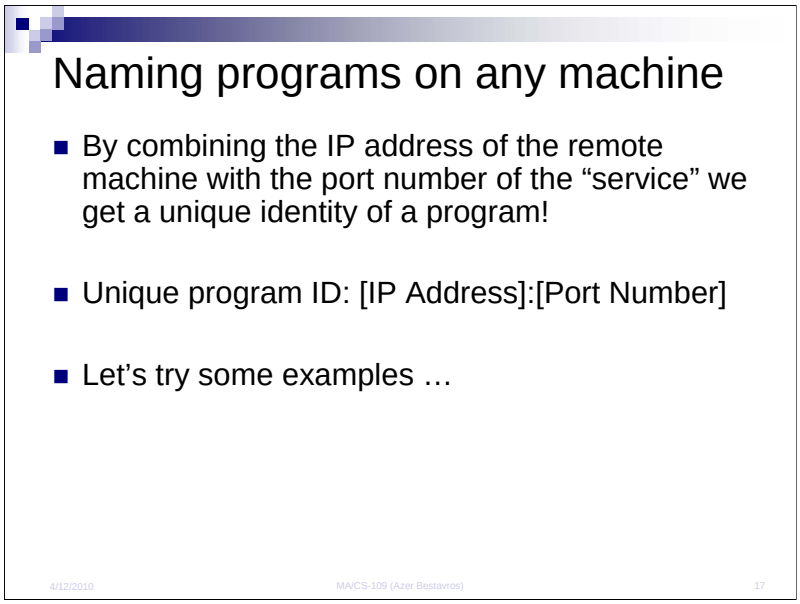


This is done by associating different programs (running on a host computer) with unique identifiers, called port numbers, based on the functionality offered by the program.

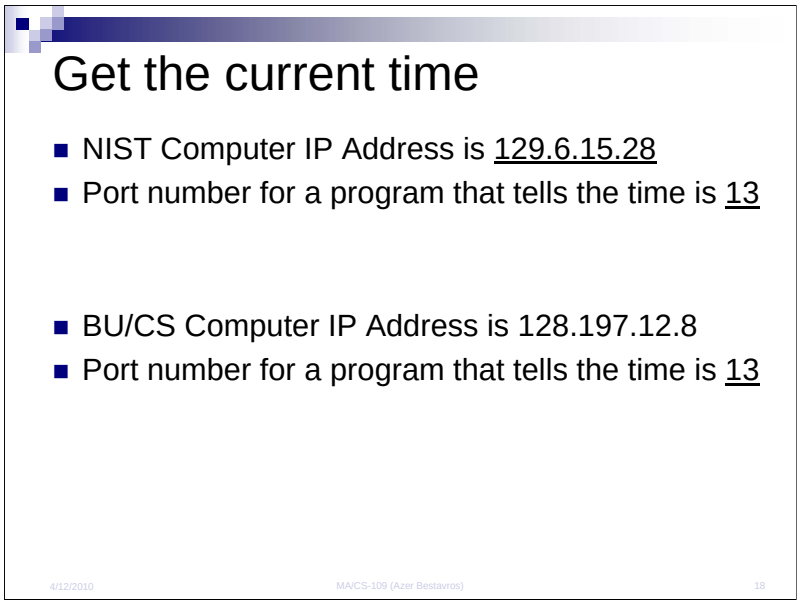
By analogy, to get service from a bank over the phone, a customer must know the bank's phone number and the extension of the department (within the bank) that is able to offer this service. Here the bank phone number is analogous to the IP address of the host computer, and the extension of the department is analogous to the port number.

For example, the web server program (the program that knows how to talk to your browser) is typically assigned port number 80.

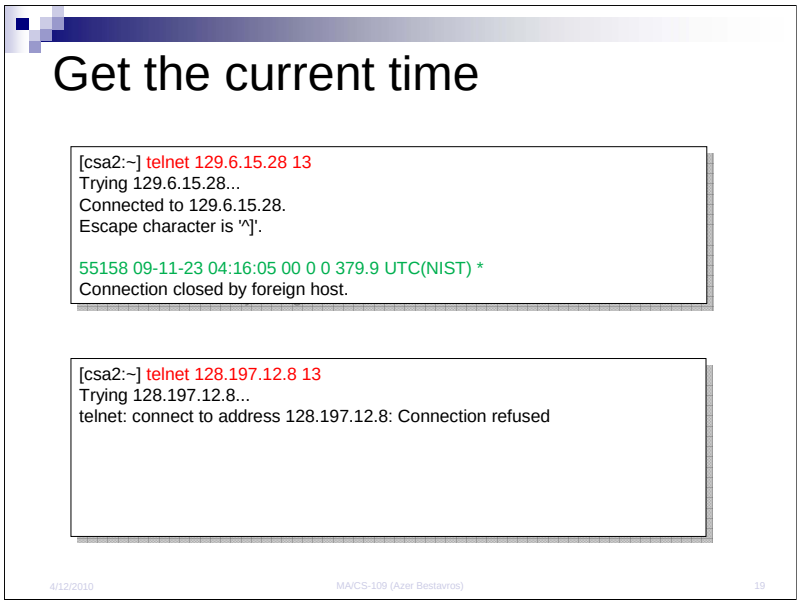
A comprehensive list of port numbers and what programs are associated with each is available at http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers



To summarize: The combination of IP address (to identify a computer) and a port number (to identify a program running on that computer) is all we need to communicate with that program.

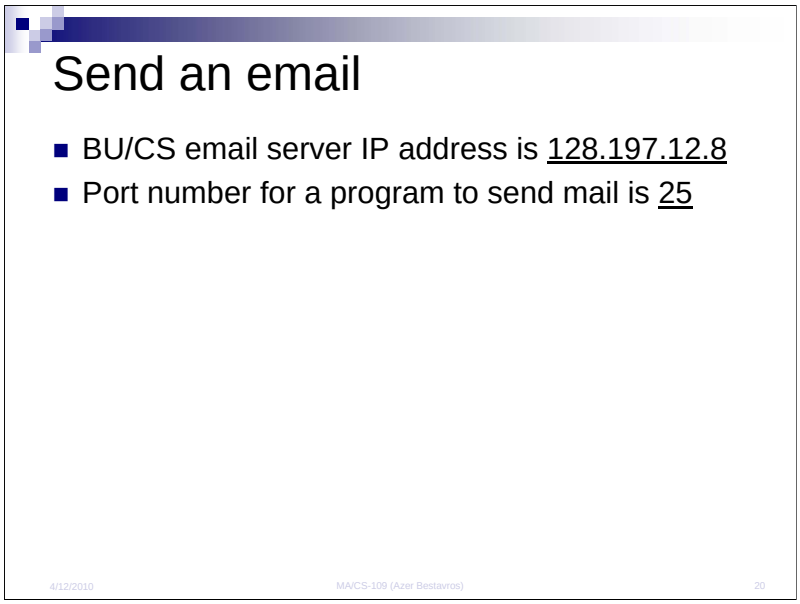


Port 13 is reserved for a program that when contacted returns the time of day. A computer that is known to run such a program is the National Institute of Standards and Technologies (NIST) – e.g., at IP address 129.6.15.28.

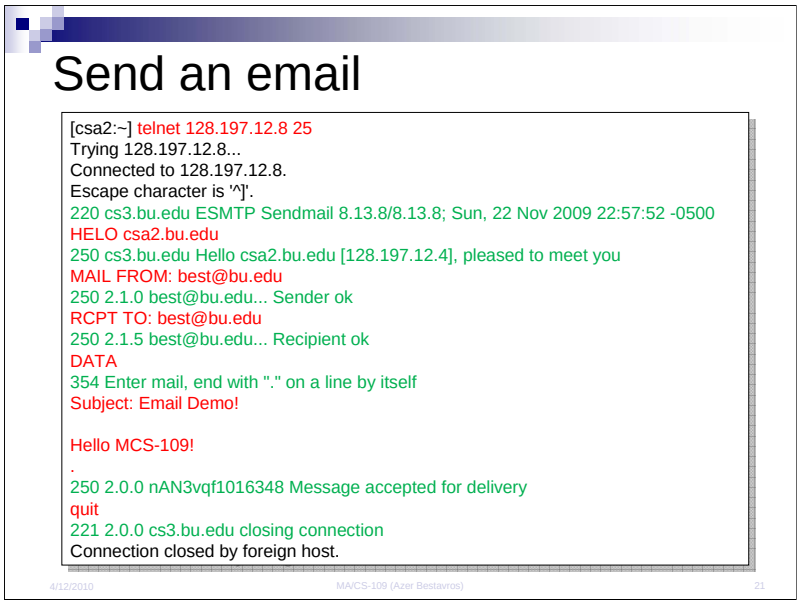


Results from contacting such a program were done live in class. The above are example outputs (obtained in Fall 2009). Text typed on the local computer (the client, which is my computer) is in red, text received from the remote computer (the server) is in green.

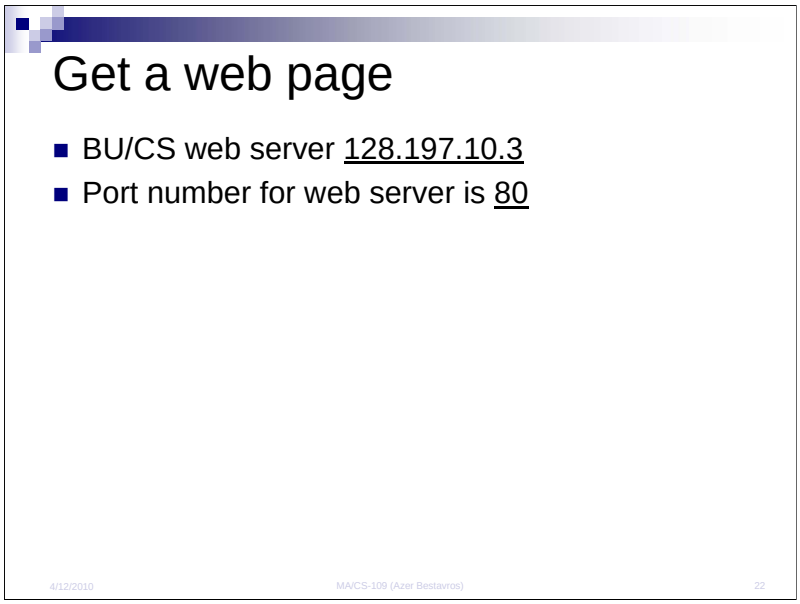
Notice that if a service (i.e., a program) is not enabled on a remote computer, the computer will "refuse" the connection (this is analogous to trying an extension that has nobody associated with it). The second example shows that.



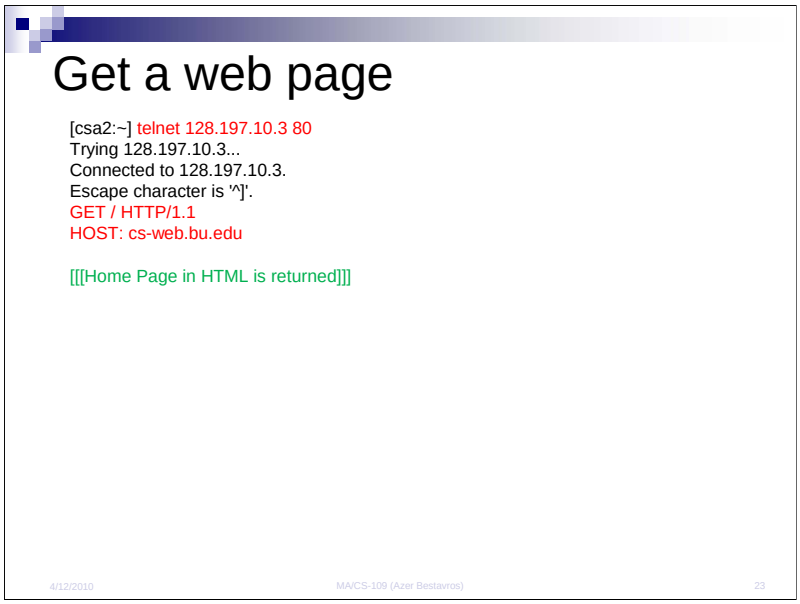
Port 25 is reserved for a program that can send an email using the Simple Mail Transfer Protocol, or SMTP.



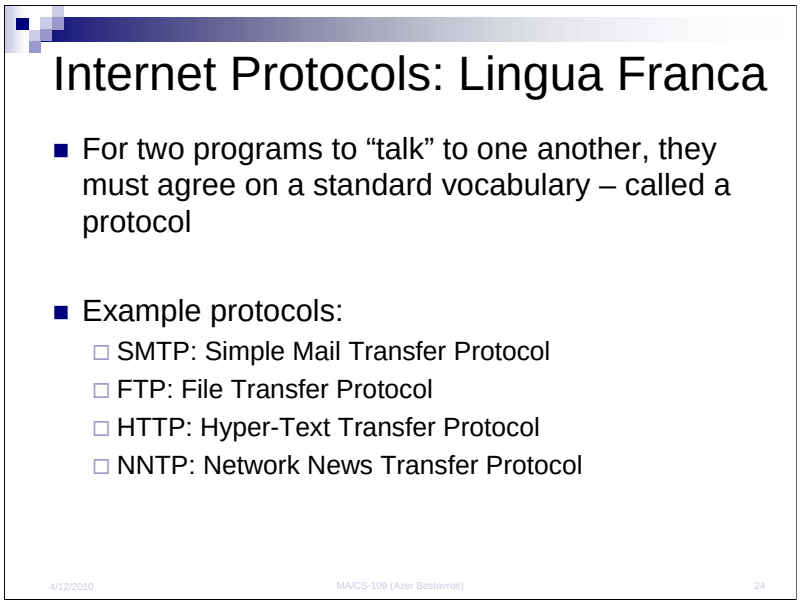
In class we tried the above exercise and showed that one can impersonate another person (e.g., I sent an email to myself claiming that it was sent by Professor Reyzin).



We can do the same exercise with a web server program, which is typically assigned to port 80.



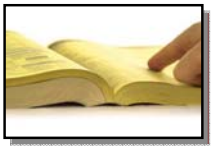
In class we tried the above exercise and showed that one can connect to a web server program (on port 80) and retrieve a web page.



As we saw in the connections we set up to “talk” to the mail server (SMTP) and web server (HTTP) programs, we used some special language to interact with these servers, e.g., “HELO”, “DATA”, “GET”, “HOST”, …

This vocabulary and the request/response sequences make up the a “standard convention” or “protocol” that enables programs to exchange information. Internet protocols such as HTTP, FTP, SMTP, NNTP, etc.

Naming machines



- IP addresses are problematic
  - They are hard to remember
  - They change
- We need the equivalent of the “yellow pages” – to go from friendly names (like cs-web.bu.edu) to IP addresses (like 128.197.10.3)
- Just have a “program” do it!!
  - How hard could it be?

4/12/2010 MA/CS-109 (Azer Bestavros) 25

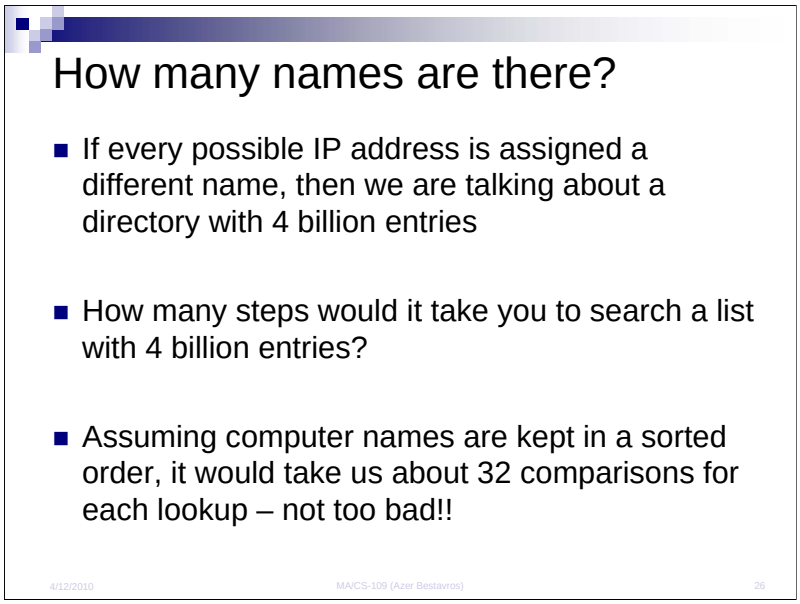
When connecting with the NIST server program (to get “time”) and when connecting with the mail server of the CS Department at BU (to send “email”), we had to know the IP address of the hosts (computers) on which these programs (called services) are running. Clearly, we can just use IP addresses to identify these hosts, but this would be both inconvenient (since it would be very hard to remember IP addresses) and impractical (since these IP addresses may change to allow for regular maintenance or if computers go down or are added to manage higher loads). Also, it may be the case that we may offer the same service on a number of hosts so that we are able to handle high traffic load. For example, to be able to meet the demand on a service like Google, Bing, or Facebook, we may have to rely on thousands of computers – often called a “computer farm”!

What we need (and what we use regularly on the Internet) are host names, like www.google.com or www.bing.com or www.facebook.com. Names like these are easy to remember and more importantly allow for significant flexibility in the management of the hosts that offer the service (by allowing multiple hosts to be associated with the same name, for example).

But, the IP protocol (e.g., for routing purposes) works with IP addresses and not “friendly” names such as google.com or bing.com. Thus, what we need is a service that given a friendly “name” can lookup the IP address of a host associated with that name. This is very much like a 411 service (or a yellow/white page service).

Implementing such a service is not hard; we know how to do this already… Just like we were able to offer services (such as “time” and “email”) by running a program on a host with a particular port number, we can have a program offer the service of “looking up the IP address of a computer given its name”. Knowing where this program is running (the IP address and port number of the host of that look-up service), we can figure out the IP address of any other host from its name.

Is it feasible to write such a program?



Clearly, the lookup service we are entertaining will have to store a table with all host names and associated IP addresses. Presumably, we could have as many names as there are IP addresses. For IP (version 4), we use 32 bits for the IP address, which gives us around 4 billion possible IP addresses. So, our table may have that many entries.

How hard is it to search for a given entry in a table with 4 billion entries? Well, if the entries are not sorted, we would have to compare the name we are looking for to about half the entries in the table (on average). That's 2 billion comparisons for every lookup – a lot!

But, we know better!

If we keep the entries sorted, we could use binary search, which is much more efficient as it would require a logarithmic number of comparisons. For a table with 4 billion entries, that's a mere 32 comparisons! Very efficient!

Ah… Now we know why designing an efficient algorithm is important!

Domain name registration

But, who is going to compile this list of names (and associated IP addresses)?

There are two questions here:

(1)Who gets to “own” a particular name such as “google.com” or “microsoft.com” or “bu.edu”?

(2)Who gets to “map” the names of hosts to IP addresses?

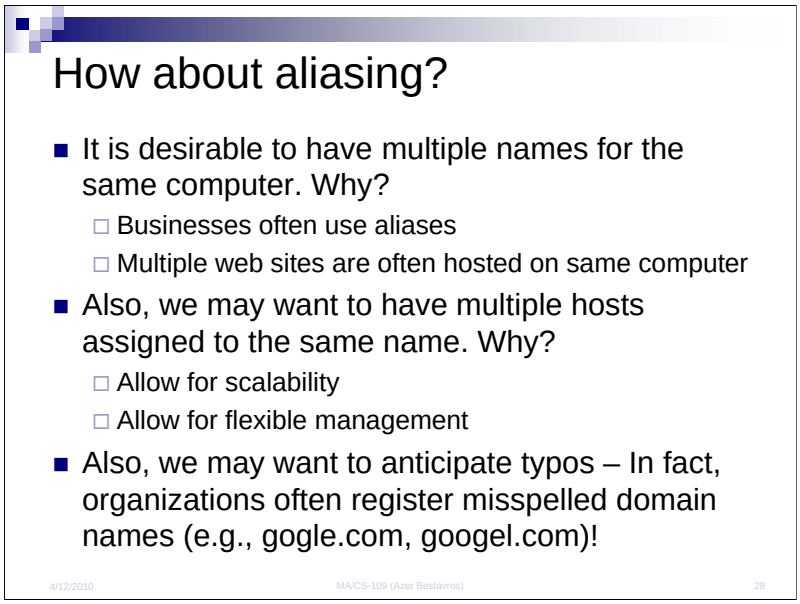
Focusing on the first question, we need an authority with which some legal entity (a company, a university, or an individual) could “register” a name and thus “own” it.

This authority is called the “Internet Corporation for Assigned Names and Numbers” (ICANN) (check http://www.icann.org/).

Created in 1998, ICANN is a non-profit corporation that took over from the US government the responsibility of overseeing various Internet governance issues (the most notable of which is the management of the Internet “name space”). For more information, check http://en.wikipedia.org/wiki/Internet_Corporation_for_Assigned_Names_and_Numbers.

There are a number of companies that allow entities to register names with the ICANN authority (for an annual fee). The slide shows an example of a communication regarding registration of a particular account.

Incidentally, it was not until 2009 that Internet “names” were allowed to be written in non-Latin characters – a feat celebrated around the world!

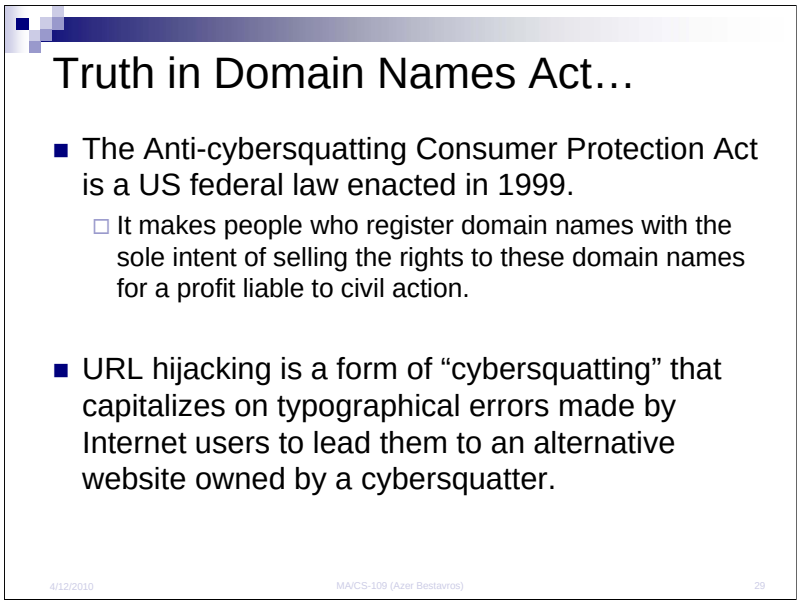


It may be desirable to have multiple names for one IP address (e.g., www.united.com and www.ual.com are names for United Airlines) – just like two individuals may have the same phone number.

Also, as we noted earlier, we may want to have the same name be associated with multiple IP addresses (e.g., as of Fall 2009, the BU web site is served by 4 hosts with IP addresses 128.197.26.3, 128.197.26.4, among others).

This is called aliasing.

Thus, it may be the case that we may have even much more than 4 billion name and IP address combinations.



As we noted, anybody (that includes you) could register an Internet name and “own” it. This is an important consideration for branding. Indeed, in the early days of the Internet (during the .com bubble), many Internet names were “snatched” by individuals and later sold to companies and businesses that wanted these names. Since this practice went a bit out of control, Congress passed a law that makes it illegal to grab an Internet name (or a variant thereof) for the sole purpose of selling it. This (now illegal) practice is called cybersquatting (Squatting means occupying an unoccupied space or building that the “squatter” does not own, or otherwise have permission to use).

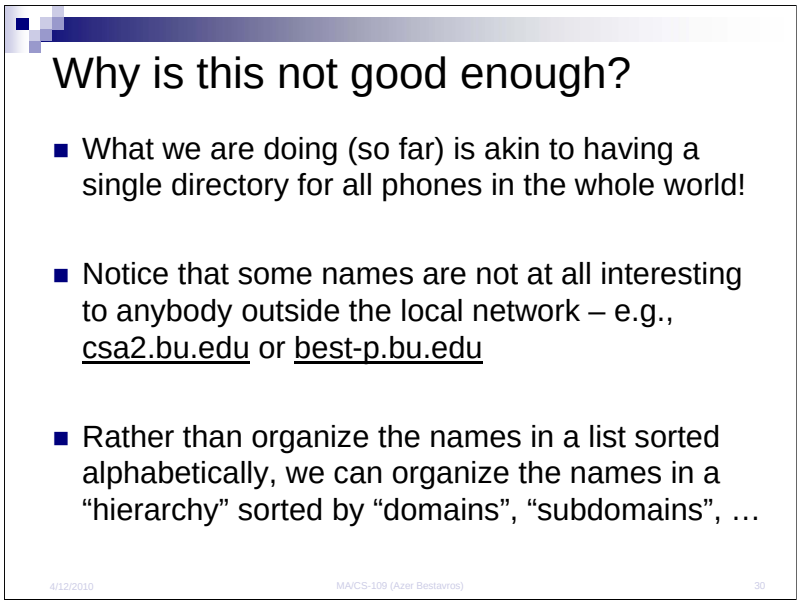
An interesting story related to cybersquatting is about trademark laws. As we emphasized throughout the course, the digital revolution (in our case the introduction of the Internet) tests our laws, which were written for a bricks and mortar world. Trademark law is no exception. Now consider the following (true) story involving a small computer business started by a person whose last name is Nissan”. Should “Nissan Computers” (as opposed to “Nissan Motors”) be able to register (and keep) the Internet name “nissan.com”? Here is a link to this true story!

Traditionally, trademarks are given protection under the law in order to avoid consumer confusion. Basically, trademark law considers both the locality of use and the industry sector for a trademark. You can’t start a restaurant called “My Little Italy” right next to another restaurant called “Little Italy” – but you can start it in the next town over. And, you can’t simply start a restaurant and call it “McDonald’s” or for that matter “McRonald’s”. If you do, McDonald’s (the one with the golden arches) will sue you! However, if you start an organic food store and call it “McDonald’s”, it would be hard to imagine that you will be sued.

Neither locality of use, nor an industry sector makes any difference on the Internet! When you type “nissan.com” you could do so from anywhere in the world and presumably be interested in any place in the world, and you could be interested in Nissan Motors or Nissan Computers.

Trademark law was designed for a world where information is "pushed" to consumers through ads for something specific (e.g., a car ad and a computer ad even if they use the same name would look very different and will not confuse the customer). On the Internet, a customer actively seeks a web page (by typing a name such as “nissan.com”).

It’s the law that has become confusing!

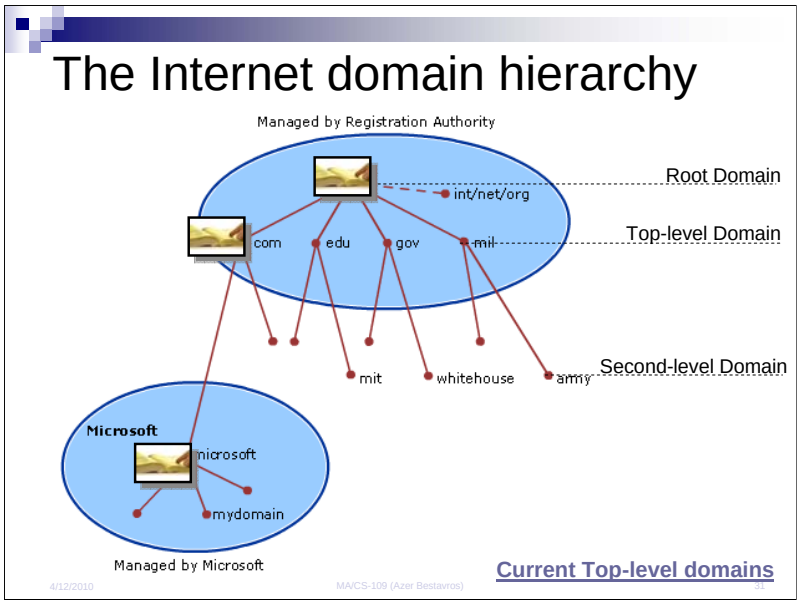


Now we turn our attention to the second question – that of deciding who gets to answer queries about the Internet directory of names.

Having a single entity do this is not practical (and delegates too much responsibility to a single authority – e.g., government or business).

Besides, many names on the Internet may be of little (or absolutely no) use to most people. For example, nobody cares what the names of network printers at BU are (except perhaps BU people).

This implies that we want to "distribute" the task of looking up names, by organizing things in a hierarchy of "domains".

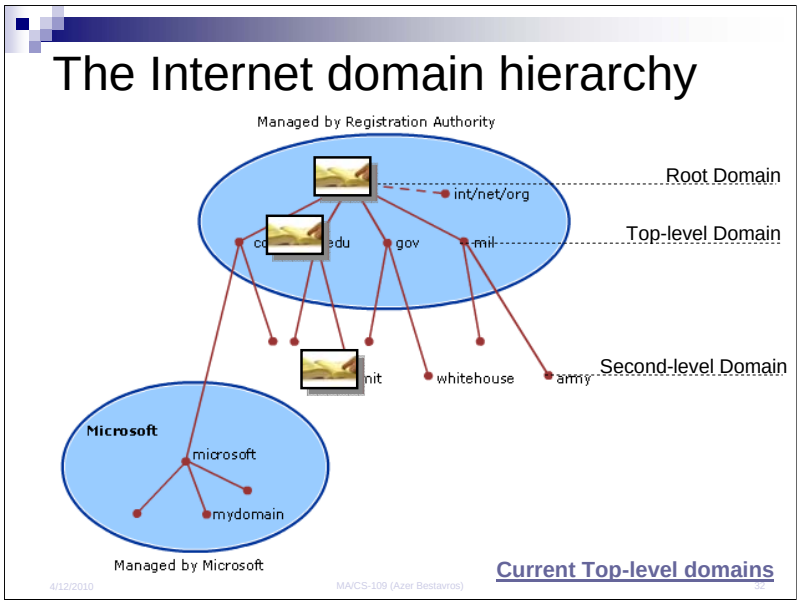


Internet names are organized in a hierarchy (or a tree). Each internal node of the tree will have its own "directory service" that is able to either translate a name to an IP address (if that name is within its domain) or else it would be able to return the IP address of another directory service (below it) that may be able to help.

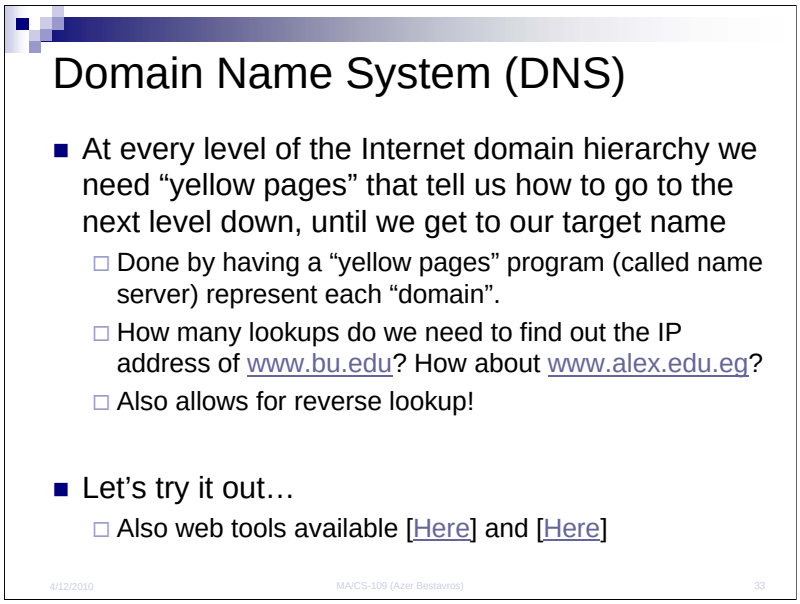
For example, consider a name like "mydomain.microsoft.com".

To find the IP address associated with that name, first one has to go to the directory service at the root of the tree (which is managed by the ICANN registration authority). Every computer comes with the IP addresses of that root authority directory service pre-programmed in it. So, everybody knows how to go to that root. This root directory service (which oversees the top-level domains) would direct us to the IP address of the directory service for ".com". By contacting the ".com" directory service (which oversees all domains under .com), we would get the IP address of the directory service for "microsoft.com". By contacting the "microsoft.com" directory service (which oversees all domains under microsoft.com domain) we would finally be able to get the IP address of mydomain.microsoft.com.

Interesting statistics about domains and how many hosts are listed in each, etc. are available from http://ftp.isc.org/www/survey/reports/current/report.bynum.



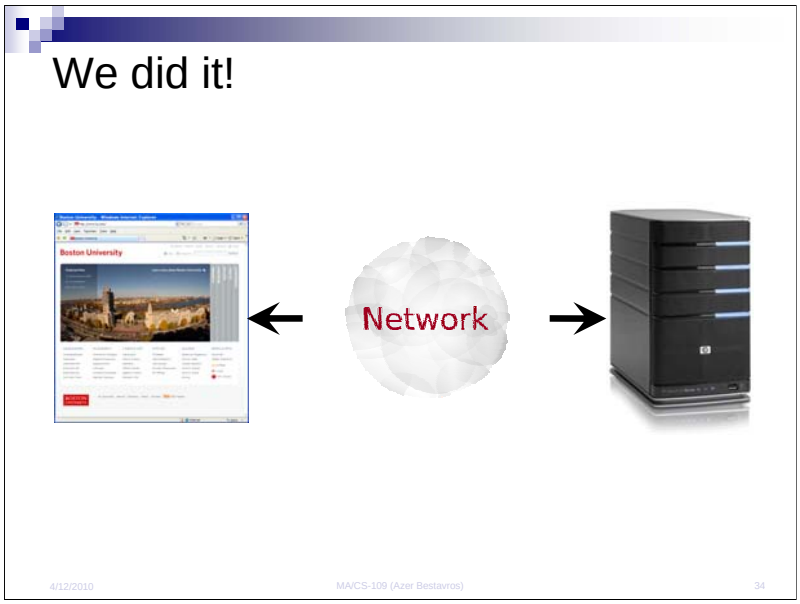
For Fun: As of Fall 2009, there were 791,457,160 host names (that's almost a billion – or one host per 5-6 people), of which around 15% are aliases. These hosts belonged to 269 top-level domains. In addition to .com, .org, .edu, .gov, there are also domains for different countries (e.g., .us for USA, .ca for Canada, .eg for Egypt, and .va for the Vatican). The statistics show that there were 4,339,451 second-level domains (such as bu.edu, microsoft.com, etc.) and 76,750,312 third level domains (such as cs.bu.edu and research.microsoft.com). Four country domains (Bouvet Island, St. Pierre And Miquelon, Svalbard And Jan Mayen Islands, and Somalia) had zero hosts in them ☺



The set of directory services used to figure out the IP address of any host name (and also the reverse lookup as in “what is the name of the host whose IP address is X?”) is called the Domain Name System or DNS.

The DNS service that each domain must have (in order to be able to resolve names under it in the DNS hierarchy) is assigned to port 53.

One can try this “domain name lookup” from any command prompt of a computer (e.g., using commands such as “nslookup” or “dig”) as well as from many web sites that allow lookups of various information about domains.



Now we can understand what needs to be done to get a simple web page such as “www.bu.edu” on one’s browser!

(1)Browser needs to find the IP address of www.bu.edu. To do so requires a lookup of .edu, and then of .bu.edu, and then of www.bu.edu.

For each one of these lookups

a. Browser must open a TCP connection with port 53 (the DNS port) of the host running the DNS service
b. Browser sends the name it wants to resolve
c. Browser gets the IP address

(2)Browser opens a TCP connection with port 80 of the host with IP address returned for www.bu.edu. The program associated with port 80 of www.bu.edu is nothing other than the web server program for BU.

(3)Browser sends a request using the HTTP vocabulary to get the Home page.

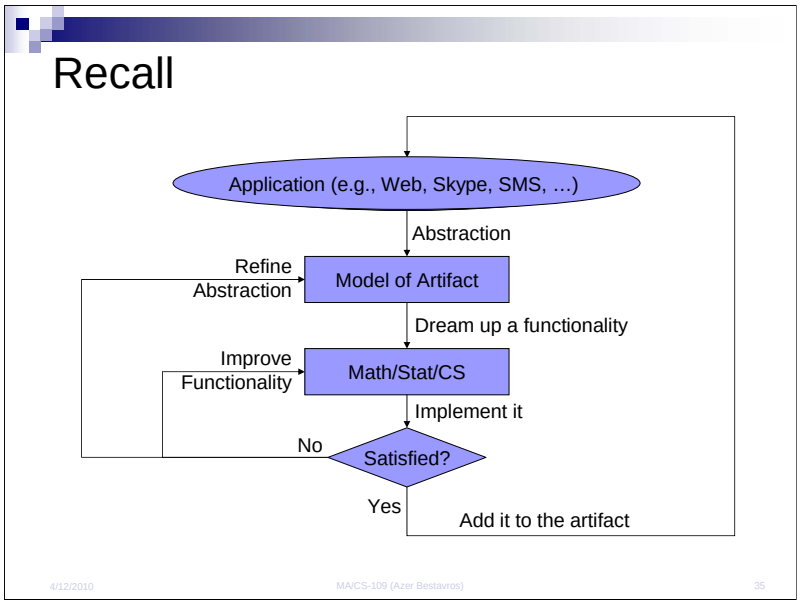
(4)Web server program responds by sending to the browser the web page (in a special markup language called “HTML”).

(5)Browser receive the web page and displays it according to the instructions in the returned HTML (these instructions may require the browser to go fetch other content (e.g., pictures) from another web server. And, to do so, this whole process has to be repeated!)

(6)Browser says “thank you” and closes the TCP connection with the server!

Notice that in all of the above we have the illusion of a program talking to another (just like the ladies using the “love phone” would do). In reality, of course, any time information is communicated from one program to another (namely steps 1.b, 1.c, 3, 4, and 5), that information must be broken down into packets (of 1.5K bytes each), each of which with the IP addresses (and port numbers) of the sender and receiver. These packets are routed through the Internet independently by having the routers forward them on to the next router along a path (e.g., as identified by Dijkstra’s algorithm). And, packets belonging to the same communication are assembled at the destination and delivered to the program at the specified port number!

Whew!

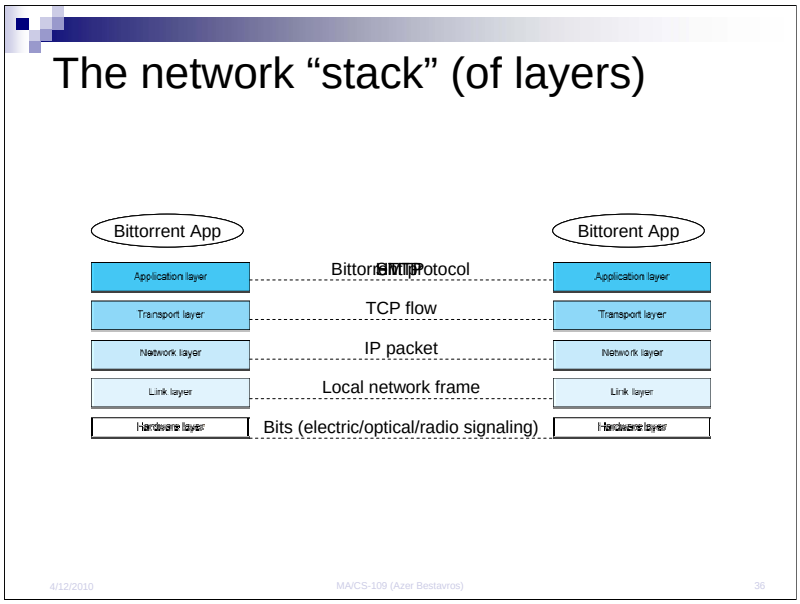


So, what have we done so far?

Well, we kept dreaming (and implementing) new functionalities, adding more layers on top of the IP (Internet Protocol) layer. Namely, we added a transport layer (TCP) as well as an application layer for running services such as name lookup (DNS), email (SMTP), web servers (HTTP).

The process of abstracting complex artifacts enabled us to add new functionality to it -- making it even more complex... Clearly, this cycle of "envisioning" a new functionality, implementing it, and adding it must be done with buy in from the community (or else it would be useless since nobody would really use it).

In a very nice article in the NYT entitled “How the Internet Got Its Rules” at http://www.nytimes.com/2009/04/07/opinion/07crocker.html, the RFC (Request for Comments) process that computer scientists followed to create the Internet's many functionalities (including all those we considered in this course) is described and its value emphasized.



And, for each one of these layers we are totally ambivalent to what happens on the layers below it.

When we have a web browser “talk” to a web server at the **application layer**, neither the browser nor the server knows anything about the layers below them; they simply “talk” to one another!

Of course in reality (as we saw), this “talking” is carried out over a TCP connection. And at that **transport layer**, we get the illusion of a communication channel that deliver information in the order it was sent, reliably, and without overloading the network.

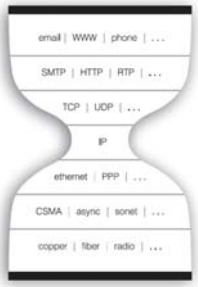
Of course in reality (as we saw), this communication channel is the result of individual packets sent from the sender to the receiver independently over the **network layer** by sending the packets over a path determined using an algorithm such as Dijkstra’s.

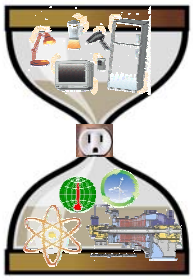
Of course in reality, each packet is forwarded from one router to the next over the local area network. At that **link layer** (as we saw), we get the illusion of a wire carrying the data.

Of course, in reality (as we saw) that wire is implemented using multiplexers and demultiplexers at the **hardware layer**.

Advantage of net abstractions

- Support multiple realizations/technologies (below) and multiple functionality/apps (above)





- IP is the equivalent of the electric plug/outlet

4/12/2010 MA/CS-109 (Azer Bestavros) 37

At each of the layers we considered so far, there might be multiple functionalities. For example, we may have multiple protocols such as HTTP and SMTP at the application layer, and various types of local area networks such as wireless and wired at the link layer.

Unlike all other layers, the network layer has only one protocol – the Internet Protocol (IP). Below the IP protocol we have protocols that support many different technologies and above the IP protocol we have protocols that support many different applications and uses.

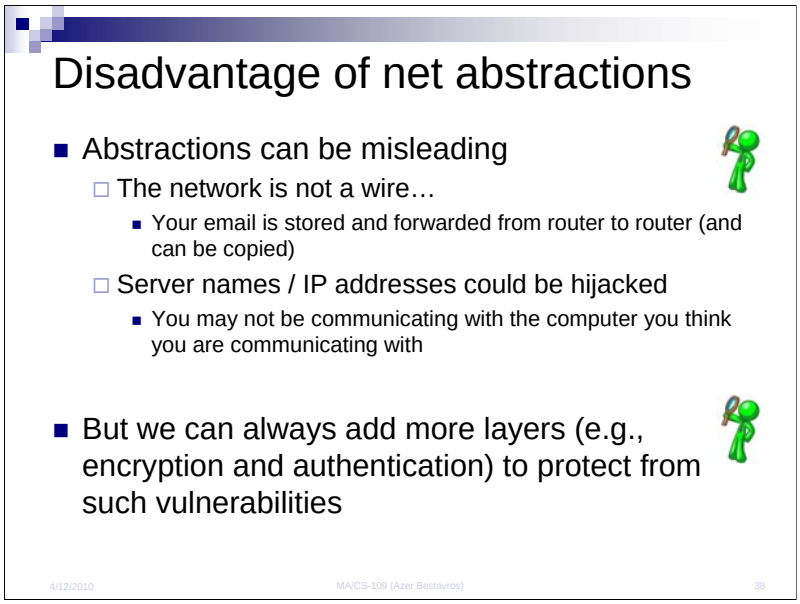
The IP protocol acts as the common “plug” that connects all these different applications (above it) to the different communication technologies (below it).

Applications above the IP layer are designed with the “illusion” of the functionality supported by IP. So, when new communication technologies emerge (e.g., when 802.11 wireless networks were introduced) these applications do not have to be redeveloped – they will simply work because the IP layer “hides” (or abstracts out) the details of that new technology from the application.

This is precisely why the Internet has emerged as the ***universal*** communication medium it is. It can carry text, images, voice, video, programs, … using wireless, optical, copper, satelite … communication hardware.

We don’t need to change applications when we introduce new hardware, and we don’t need to change the hardware when we introduce a new applications!

This is so, because of the “narrow waist” (or hourglass) shape of the Internet design, with a single protocol – the IP protocol – at this narrow waist.



So far, we have been singing the praises of abstractions… But, one has to also realize that these abstractions hide details, which may have consequences. For example, while we may want to think of the transport layer as providing us with a "wire" between two programs running on two different hosts, in reality the data transmitted over that abstract "wire" is stored and forwarded from one router to the next in the form of packets. Well, these packets can be copied and thus our communications may be intercepted!

But, the good news is that if we know what an abstraction hides (e.g., the fact that people can listen in on Internet communications), we can always add more functionalities (i.e., add more layers) to the Internet to deal with such limitations.

For example, as you recall from our discussion of cryptography and the use of public and private keys, since people can listen in on Internet communication, we can add a layer that encrypts the data we send over the Internet so that if anybody intercepts it, they cannot decipher it. This is precisely how sensitive information (e.g., a credit card number) is communicated safely over the Internet.