22C:118 Fall 2004 Week 1 Summary

1. On Monday, the course syllabus was distributed and some of the course plan and expectations explained. Then, we began covering some of the basic constructs of networking and its motivations.

   - The term *link* refers to the idea of a communication path between computers or other devices.
   - Most of the complexities of networking arise from scarcity: we can't have direct links from everyone to everyone else.
   - Links have different qualities (not all links are the same). Some links are between two endpoints, some (like satellites) transmit to many listeners.
   - Simplex, (full) Duplex, half-duplex links.
   - "Pipe Width" varies from link to link. Width of pipe constrains how many bits per second can be put into a link (technically called bandwidth).
   - "Velocity" varies from link to link. How long does it take for a bit to travel from one end to the other?
   - Reliability varies from link to link. Some links may occasionally lose or corrupt bits.
   - Some links are real, physical links, others are "virtual links" that look like real links to end-users, but are actually built out of a combination of physical links and software that connects the physical links in a path.

2. The TA office and office hours are added to the online syllabus.

3. Wednesday's lecture motivated two basic ideas, the sharing of links (introducting terms *communication channel* and *connection*), and the layered architecture of networks. Within layers, there are *network protocols*, and the TCP/IP suite of protocols provides many examples.

4. **Reading Assignment:** pages 3-8 of the TCP/IP Sockets book, please read this by 30 August, and concentrate on all the italicized terms in the text.

5. Friday's lecture elaborated on the TCP/IP layering (the protocol stack), introducing explicitly the MAC layer — Ethernet or WiFi usually sits at this layer — and the communication units at the different layers: signals, bits, frames, datagrams, packets, and byte streams.

   The Internet is an example of a packet switching network, meaning that sharing of links is essentially managed by addressing. There are different forms of addressing at the different layers, including frame addresses (typically 48 bits), IP addresses (currently 32 bits), port addresses (16 bits), and FQDN addresses such as www.google.com, used by applications. Packets have several parts, including the sender address, the destination address, and the "payload" (which has the actual useful data from the application user).

Informally, the process of adding address information so that a packet can be distinguished from other packets sharing the same communication channel is called multiplexing (actually it has a more formal meaning, given in textbooks); demultiplexing removes the address information after a packet arrives to its destination.

It is important to realize that each IP packet is forwarded through the ”internet cloud”, from link to link, on the basis of its destination address. Each router (or switch) in the middle of a route between sender and destination has to decide, for each packet, what to do with the packet: either keep the packet and deliver it to some local application, or forward it (resend it) along some link. Which link to choose for resending? This decision is controlled by a very simple algorithm: table lookup. That is, conceptually there exists a table, and each line in the table consists of an IP address and a link to use for resending along the path leading eventually to the host computer associated with that IP address. (Actually, the table is a bit more complicated, as we'll see later on.)

22C:118 Fall 2004 Week 2 Summary

1. On Monday, before starting the lecture, we looked at how to navigate the course web page. Here, we saw that the first homework assignment has been posted. The remainder of Monday was dedicated to a detailed presentation of the flow of data from one host to another, crossing four LANs. Host A sends a packet to host B, which forwards to C, which then forwards to D. The layers application, TCP, IP, Ethernet, NIC, and wire (in that order) are used at host A. At hosts B and C, the order is wire, NIC#1, Ethernet, IP, Ethernet, NIC#2, and wire. At host D, the order is wire, NIC, Ethernet, IP, TCP, and application. This detailed examination of the flow of a packet, when combined with the explanation of ARP in the homework's assigned reading of RFC 1180, should be enough for students to understand the various steps that go on in packet forwarding — except for routing decisions.

2. Wednesday started with a detailed look at IP addresses, how the numbers are divided up into a network, subnet, and host interface part. When LANs are configured, the usual technique is to give each LAN its own subnet. Also, each host on the LAN uses the same subnet for its IP address for the attached NIC. This implies, of course, that a host attached to two or more LANs (such as a router) would have IP addresses for the subnets corresponding to each LAN.

   Two special host interface addresses are reserved, and not to be used as real host addresses: if the host part has all zero bits, then the address is a "network" or "subnetwork" address. If the host part has all one bits, then we have a broadcast address for than subnet.

   Once we know about subnets and the convention to use the same subnet for all host interfaces attached to a LAN, it makes sense to talk about routing tables. The basic problem of routing tables is that they get to be too large and difficult to maintain if IP addresses are chaotically assigned to hosts. By using the subnet discipline, we can have "wildcard" entries in routing tables, so that a single table entry is all that's needed to refer to all the hosts of a LAN. What about IP addresses outside a LAN? For that, the routing table is extended to have gateways. A gateway is a host attached to more than one LAN, which can forward packets from one LAN to another. A LAN can actually have more than one gateway (but this was not shown in class). A line in the routing table can use wildcard matching to indicate that packets not intended for any host on the LAN should be sent to a gateway. When a datagram matches this type of table entry, the datagram is "embedded" in an Ethernet frame, much as a letter is put inside an envelope, such that the frame's destination MAC address is that of the gateway. Then the gateway will receive the frame and not throw it away, but rather, allow the IP layer of the gateway to decide what to do.

   By carefully arranging gateways in the right way, we can have "faith-based routing", meaning that each gateway that cannot directly deliver a datagram to a host on a LAN attached to it will find in its own routing table the IP address of another gateway. The first gateway "has faith" that the second gateway (the one in its routing table) has sufficient information either to directly forward the datagram to a host or to forward the datagram instead to yet another gateway. This process may repeat many times.

3. **Note:** when sending email to the professor or TA, please put the string [118] somewhere on the subject line. For example,

   ```
   Subject:  [118] question about homework 1
   ```

   Doing this will guide your email safely through the spam filters and reduce the chance it can be lost or delayed.

4. Posted: solutions to Homework 1, see the followup (reply) to the first homework article on the webpage.

5. New reading assignment: please read the first chapter of Sync (pages 11-39) in Week 3 (Monday is a holiday).

6. Friday's lecture was a high-level description of how FQDNs (Fully Qualified Domain Name, look it up at http://www.webopedia.com/TERM/F/FQDN.html, for example) map to IP numbers. TCP/IP works without domains or host names, so some protocol is needed to maintain and lookup IP numbers given the host and domain information. This is done by a distributed network of domain servers, using the DNS hierarchy (look that up in http://supportnet.merit.edu/m-spectop/t-dns/hierarchy.html). The DNS hierarchy roughly corresponds to a tree, with "." as the root, domains edu, com, gov, net, org, us, fr, ca, etc, as children. Each of these domains has its own children; the com subdomain has subdomains yahoo, google, amazon, and so on. The edu subdomain has uiowa, iastate, uiuc, osu, and so on. Within the uiowa domain, there are subdomains for cs, psychology, medicine, and so on. Within the cs subdomain of uiowa, we have hostnames, like www, l-lnx119, r-hpx200, and many others.

   The DNS protocol and system of servers is essential a distributed database; each domain has a DNS server which is aware of hosts in its domains and the IP numbers of parent DNS servers as well as child (subdomain) DNS servers — these are needed to forward requests. DNS servers also maintain caches that hold results from previous lookups, so that requests about remote hostnames can be quickly answered without having to send packets across the Internet.

   DNS databases are flexible enough to allow aliases (one IP number can have many names) and generic names to allow one name to refer to many IP numbers (this is a way to do load balancing; our computer science does this for the name linux, for example). Using the "host" command on our departmental machines, you can query the DNS databases, though the syntax is not simple for doing this for all types of queries. Try "host www.google.com" for example, and you will see that www.google.com is an alias for www.google.akadns.net, which has two IP addresses.

   Some services even provide the ability to create host names and IP number associations on-the-fly: a list of these can be seen in (google directory).

Friday's lecture also had a short presentation of how ICMP enables ping and traceroute to function. The secret is the TTL field of datagrams, which also prevents packets from being endlessly forwarded in case there are routing table errors. The traceroute command uses TTL values of increasing size, probing along a path to discover the identity of routers and determining how many hops are in a path.

22C:118 Fall 2004 Week 3 Summary

1. Monday was a holiday and the weather was nice.

2. Wednesday was the last lecture of the initial part of the course, introducing TCP/IP. The lecture motivated network address translation (NAT), firewalls, and proxies. Also, the topic of routing protocols was mentioned – these maintain routing tables in automatic ways to route around link failures and network congestion; BGP is a famous example of a routing protocol in the current Internet. It is perhaps a surprise that such protocols as BGP and DNS, both so crucial to the Internet, are actually at the application layer of the protocol stack. As a final part of the lecture, some of the basic terminology of experimental measurement was reviewed (frequency distributions, the mean, the mode, the median, the standard deviation).

3. Friday continued the topic of TCP/IP, but as demonstrated using Python programs (the course web page has an article on TCP/IP sockets, which adapts the second chapter of the TCP/IP Sockets book to Python. The main thing to be aware of in TCP is the lack of correspondence between read and write operations (at least in terms of the the exact number of bytes). This is typical of stream-oriented communication, including files and Unix pipes.

22C:118 Fall 2004 Week 4 Summary

1. This week introduces the Python language, culminating in the first programming homework. Monday and Wednesday lectures demonstrate the basic features of the language, with more about classes, objects, and Python libraries later in the week.
2. Please look under the "Help" on the course web site, which has a Python FAQ with pointers to resources.
3. Friday's lecture presented UDP in Python, basically repeating the subject of Chapter 4 of the TCP/IP Sockets book, but in Python. The lecture also motivated the use of concurrency in Python, typically programmed using one or the other of the various thread modules.

22C:118 Fall 2004 Week 5 Summary

1. Solutions to the second homework (Python programs) are available as a reply to the article assigning the homework.

2. The third homework is due Friday, posted on the course web site.

3. Monday's lecture went over a couple of solutions to the second homework and explained a little about the third homework.

   Students are reminded that the first exam is one week from Monday.

   The remainder of Monday's lecture discussed Python threading and locking features. An article posted to the course web site shows several examples.

4. Wednesday continued the explanation of Locks and threading, with a demonstration, also including the Queue object. Instead of using Threads, the select() function is an alternative way to manage a number of concurrently open sockets.

   Wednesday's lecture also started the topic of message content (from Chapter of the TCP/IP Sockets book) and protocol message formats. This was illustrated for HTTP by capturing what a browser transmits to a server — HTTP commands.

5. Scores for first and second homeworks have been posted to the web site (see under Help topic on course home page).

6. Friday's lecture included an Exam Review, showing the midterm exam from the previous semester.

   Then we looked a bit more in depth at HTTP and how Python supports common Internet protocols such as HTTP.

   The content of a message is often formalized using a standard definition language, such as SGML, HTML, or XML – Python also has modules for some of this.

   RSS is a new, emerging standard for syndication (news). Using RSS feeds, it is convenient to write a news aggregator. A Python RSS reader was demonstrated in class.

22C:118 Fall 2004 Week 6-7 Summary

1. The first exam was Monday 27 September.
2. Wednesday's lecture first went over the exam solutions.
3. Wednesday was also the first introduction to parallel programming, mainly about motivations for parallel programming and the relation to networking (it is an important application of networks).
4. Friday began the specific notions of Linda (a coordination language) and the tuplespace. Operations for the tuplespace were listed and illustrated with a few examples, including some online examples. To accompany these topics, the course web page offers an implementation of Linda in Python and reading material.
5. Skipped: Monday and Wednesday of Week 7.
6. Friday's lecture covered one of the approaches to parallel programming, the specialist approach. This was illustrated by a specialist approach to computing prime numbers (blackboard only), and then two specialist approaches to summing a vector of numbers. This will be the basis for the next homework.

22C:118 Fall 2004 Week 8 Summary

1. Monday's lecture illustrated the specialist approach to parallel programming with the example of matrix multiplication done by systolic arrays. Slides are posted on the course web site for this.

2. Wednesday's lecture covered barrier synchronization, and several possible solutions using Linda tuple operations were considered in the lecture.

3. Friday's lecture concentrated on agenda-style parallel programming, and this was illustrated by the simple example of summing items in an array. A more interesting example of image processing was also described; this is the basis of the fifth homework.

22C:118 Fall 2004 Week 9 Summary

1. Monday started a break from programming topics, to establish some basis for network measurement. The lecture's topic was basic probability and described the main probability distributions used in networking.
2. Wednesday's lecture presented a examples of computing expected values, both analytically and experimentally. Then the lecture covered the important topics of statistical confidence and sampling theory, culminating in an example using the Student t-Distribution to calculate confidence intervals.
3. Friday was the first lecture on simulation, starting with some motivating examples and categorizations: time-stepped vs event-driven, random vs deterministic, end-result only vs behavior studies. We will mainly look at event-driven, but the example of Lotka-Volterra simulation (which is time-stepped) was illustrated during the lecture.

22C:118 Fall 2004 Week 10 Summary

1. Monday introduced the basics of event-driven simulation, from philosophy to representation of objects, events, and the clock. A vague example based on customers arriving to an ATM machine hinted at how such a simulation may work.
2. Wednesday went over Python version of event-driven simulation in some detail, showing the example of customers arriving to a server. The lecture also began the topic of queueing theory.
3. The first part of Friday's class was answering questions about the homework. Then the lecture continued a little about queueing theory, until everyone almost fell asleep. Then the lecture turned again to event-driven simulation, this time introducing the notion of a distributed simulation, which requires messages between simulators at different hosts. An example showed this to be a non-trivial problem.

22C:118 Fall 2004 Week 11 Summary

1. Monday began the details of how distributed event-driven simulation works. The notion of event messages (different simulators send messages to each other when events are scheduled for objects located at other simulators) can possibly lead to causality errors.

   Causality errors can be prevented by arranging for each simulator to replace its event list by a collection of queues, and by satisfying two ”invariant” conditions. The first invariant condition specifies that all queues (actually the entire history of events ever put on any queue) are in FIFO, timestamp increasing order; the second invariant says that a simulator waits until all its input queues are nonempty, and then selects a queued event with minimum timestamp to fire.

   It was shown in class (a theorem) that such a distributed simulation will always produce a result equivalent to a sequential simulator of the entire simulation.

   Unfortunately, the second invariant can cause the distributed simulation to deadlock. One way to prevent a deadlock is by the use of null messages.

2. Wednesday's lecture continued the topic of distributed simulation, looking at the problem of how to make sure that the first invariant holds during the simulation. The idea proposed is to use output queues as well as input queues: rather than send an event message immediately, that message is buffered in an output queue. Such a message is only really sent to the corresponding input queue when the local simulator clock is large enough to ensure that no additional event will ever be generated with a smaller timestamp than the buffered message in question. This idea can be improved if the application being simulated provides ”lookahead”, meaning that it can be guaranteed that no scheduled event can have an earlier timestamp than (clock+L), where clock is the value of the local simtime, and L is the lookahead factor.

   Friday's lecture (not yet completed at the time of this writing), but some part of the lecture will review topics covered on Monday's exam.

22C:118 Fall 2004 Week 12 Summary

1. Monday was the second examination.

2. Wednesday's lecture examined the paradigm of Client-Server in more detail, by introducing the RPC abstraction (RMI is the same idea). Several non-trivial aspects of Remote Procedure Call include security, fault tolerance, heterogenous systems, and how to hide networking aspects so that applications programmers don't need to program all the details.

   Important distinctions for RPC are whether a specific Call (or Method) would be stateless or stateful, since the latter implies some session context, added overhead, and perhaps cookies or some other multiplexing technique. Methods can also be idempotent or non-idempotent, and the possiblity of nested RPC is a further complication, particularly for fault tolerance. In class it was shown why UDP can be preferred over TCP for some types of RPC.

3. Friday first reviewed the exam given on Monday. Then, continuing on with RPC, the lecture presented an overview of how the process of implementing RPC in a general way can be automated. Concepts such as brokers (Object Brokers are used in CORBA) or directory services can help locate servers; the semantics and types of arguments of Remote Call/Method are documented using an Interface Definition Language; to change a local call to a remote call, the method or routine is replaced by a stub, which manages networking, does marshaling and demarshaling of parameters, and data conversion to solve the heterogeneity problem. IDL specs for a method are input to a compiler that produces client and server stubs, header files or include files for the application programs, and possibly also the input needed for a directory service. All that an application programmer needs to do is write the application and describe methods using the IDL, and the rest (one hopes) is automatic.

22C:118 Fall 2004 Week 13 Summary

1. Monday: a lecture on databases, the basics of transactions, the ACID property and recovery, and the need for scheduling algorithms that guarantee serializability.

2. On Wednesday, the topic of two-phase locking (2PL) and how it guarantees serializability (but not recoverability, which motivates strict 2PL) was explained. Then, finally, we are ready for the main issue related to networking, how to manage transactions in distributed databases. The protocol for two-phase commit was introduced.

3. Friday's lecture examined the two-phase commit protocol and how transactions are coordinated so that ACID properties hold even in a distributed database. A special case of this, now seen in many commercial applications, is a replicated database, where deciding the order of transaction commits is related to barrier synchronization (that is, before the next transaction can be run, all sites have to finish the current transaction). In non-replicated distributed databases, there is opportunity for more parallelism because different transactions can operate at different sites, on unrelated parts of the database. The two-phase commit protocol requires that all the participants of a transaction be in a kind of uncertain state, between the time they are ready to commit and actually get a message from the coordinator to do the commit (or, they could get abort messages, which is why things are uncertain). During the period of uncertainty, updates to the database are pending, and the variables to be updated are locked. Thus, transactions potentially block other transactions during the period of uncertainty. This drawback can be addressed by using yet another phase, leading to a three-phase commit protocol — that protocol is described in the reading.

22C:118 Fall 2004 Week 15 Summary

1. Monday: Deadlocks in databases, prevention and detection, was the lecture topic. Broadly speaking, deadlocks can be communication deadlocks (we saw this with TCP applications and with parallel simulation) or resource deadlocks, as occur with locking for operating systems (even in Python threads) and database transactions.

   Deadlocks can be prevented in databases by aborting ”young” transactions whenever they block older ones, so that the older transactions never wait on locking. This idea requires some kind of timestamping mechanism for given each transaction a unique timestamp.

   Another way to prevent deadlock is to order all possible resources by some known ordering R and make sure that every transaction requests all its locks by some (sub-) order following R. Both these prevention ideas could also work in the case of distributed databases.

   Deadlocks are detected by having the transaction manager construct a wait-for graph (WFG). There exists a deadlock if and only if the WFG contains a directed cycle. The WFG is a dynamic object, changing over time as transactions lock and unlock items.

   In a distributed database, there can be deadlocks even though no site's transaction manager has a WFG with a cycle. That is because a particular transaction can have subtransactions running at the different sites. To solve the detection problem, one can use a new type of server called a ”deadlock detector”. As edges are added and removed from the local WFG at each site, messages are sent to the deadlock detector. The idea is for the deadlock detector to construct a ”global” WFG. However, this is tricky to implement correctly: the deadlock detector is not synchronized exactly with each site's transaction manager, and therefore we have the danger of ”phantom deadlock” detection.

2. Wednesday began the new topic of Peer-to-Peer networks, which use the concept of Overlay Networks to join network users. An overlay networks is essentially a virtual network on top of the Internet in which nodes are programs and links are TCP connections (they could be UDP as well). In a peer to peer network, all peers are both clients and servers.

   The lecture's motivation is file-sharing. We looked at classical file sharing systems such as Napster and Gnutella. Napster requires a server that acts as a directory of all peers and their content. If you're looking for a file, ask the server and find out who has it. The actual copying of the file goes directly between peers. Napster suffers because the server is a single point of failure. By contrast, Gnutella has no server. Each request for a file simply gets broadcast to all peers currently connected to the Gnutella (overlay) network. But the Gnutella idea becomes inefficient when the network gets large.

3. Friday covered the Chord style of peer-to-peer networking. It is a special-purpose network based on a circle of identifiers and ”fingers” that are pointers within the circle to make find operations more efficient.

22C:118 Fall 2004 Week 16 Summary

1. Monday: briefly covered was the way that applications like BitTorrent and eDonkey split files into chunks. This enables parallel downloading of files, which is done by a server that keeps track of which chunks are available and where they are. Parallel downloading could make freeloading problems worse, and BitTorrent solves this by regulating download rates. Only users that upload chunks are allowed to use higher speed, parallel downloading of chunks. This mechanism enforces "fair play" among the clients.

   Trust is a significant issue with peer-to-peer networks, as well as other network applications. While security techniques (passwords, encryption) can solve some problems of how we trust the network hardware, we need other ideas for how to trust the quality of a web search, or the response to a P2P file search. One idea for trust is algorithmic: the pagerank algorithm used by Google, for instance, attempts to find the most valuable (trusted) links that match a search. Pagerank is a graph-theoretic measure of how many URL-links refer to a page; the more links, the (hoped-for) more valued is the page. Another example of an algorithmic technique is the way that spam-filters identify email as untrusted. Beyond algorithms, another idea is to use existing "social networks" that model the way humans trust each other. The lecture looked at `http://trust.mindswap.org/trustProject.shtml` where there is a presentation about how email could rank incoming mail by a trust metric. The trust metric could be set by instant message contacts or other ways to identify which email senders are trusted. Once each user has a database of trusted contacts, a networked application could actually calculate the trust of unknown parties ("friend of a friend"), which is the theme of several social network software applications.

2. Wednesday continued the theme of trust, looking again at the topic of reputation networks. The study of reputation networks and how to use them for improving network software is an emerging area; the lecture looked at one example, `buddybuzz.org`, which proposes ranking reading recommendations based on the structure and measures in a social network.

   Wednesday also mentioned another new area in networking: ubiquitous computing, pervasive computing, and proactive computing (a search engine will turn up many interesting resources for these terms). What will happen with the number of computers vastly outnumber humans? How will they be managed and networked? The new generation of such computers may be very tiny, even to the size of a grain of dust, using wireless links for communication, designed for a specific purpose, yet capable of supporting more general applications.