**CSC 435 Distributed Systems**

**Elijah Caluya**

**11/17/19**

**Final Study Log**

Total Word Count: 7745

**Twenty Concepts**

1. **Interface Definition Language - IDL**
   1. The main purpose of the IDL is to define the interface. THis is done by specifying the names of procedures/methods and the precise bit streams that travel through the interface in each direction. The IDL is very useful in establishing a client/server connection. We can configure the IDL to automatically generate a client stub and server skeleton/stub. The server skeleton is called by unmarshalling RPC OpSys subsystem on the remote server and this is called through the local server operating system after prompting the remote system.The following is how a remote call can be structured by IDL:
      1. Calling Routine (Parent Client node) uses client stub to connect to the server stub with both of the outlines being created by the IDL.
      2. The server stub then connects to the called subroutine (child server node)
      3. Then, that child server node will send make a call back to the server stub then that call goes back to the client stub.
      4. The final call is the client stub connecting back to the calling routine (Parent client node).
2. **Administration**
   1. Administration is categorized in this class as the most often overlooked scale-up problem. The main point of this scale up problem is that the design of the system would need to change as opposed to just increasing the size of the resources. Administration is not a traditionally algorithm-based structure so this is why it may be overlooked when discussing scaling up a system. Administration is an important factor when scaling up because it may require fundamental changes in the structure of the system design. Due to this fact, administration can often be considered the most critical bottleneck in scaling up.
3. **Blocking Calls**
   1. A blocking call is a system call that will pause the calling process until the event that the call was blocked for occurs and then the process will wake up and continue. Blocking calls can cause problems in distributed systems because the call can take a long time over the network or the calling process can wait for a very long time even though the CPU and memory resources are available. Another problem could be the geography connections because the farther they are, the more chances there are of failures in the connection. There is an important concept in blocking calls and non-blocking calls: Blocking calls are synchronous and non-blocking calls are asynchronous and usually use two processes. One compromise in using synchronous or asynchronous calls is the extra complexity needed for asynchronous calls due to having multiple processes. There are many types of asynchronous calls like send and forget (minimally) and waiting until the server application actually starts work on the request (maximally).
4. **Transactions**
   1. A transaction is the exchange of information and the related work needed for satisfying the request. A general rule for transactions is that the before and after state must have the same amount (Bank transaction example). A main topic related to transactions is the critical section, which is where you need to reserve resources for the duration of the transaction so that different processes cannot concurrently access the same data and produce incorrect results.Ensuring that no processes are accessing the critical section at the same time, we use semaphores that act as locks through Test and Set instruction. Transactions are much more complex in Distributed Systems because there is no hardware support for critical sections (No Test and Set), no shared memory, and the sending of messages over the network is unreliable.
5. **Process**
   1. A Process can be defined as a computer program that is being executed and it is possible that many processes can run a single program at the same time. This is true in local systems where all processes run on one CPU and in distributed systems where multiple processes can run on different network-connected CPU’s. For local systems, processes are able to communicate with each other through shared memory locations using Inter-processed communication (IPC). However, in distributed systems, the processes must send messages over the network because IPC is only available in local systems.
   2. The main topic for processes is context switching. This can be defined as saving the state of a process so that it can be resumed at a different time and allows for a single CPU to be shared by multiple processes. In context switching, processes are continuously swapped in and out of the CPU so that every process will have their turn to run for a while then it is swapped out for another process with its state saved. This concept was exemplified in our JokeServer project because we had to save the state of the client in some form on either the server, client, or both. It was discussed in class that the perfect solution for such problems in distributed shared memory systems is too hard to implement. This is because the implementation of a universal shared memory system as middleware is very complex as it involves all core memory is mapped to UUID’s and each machine would have its own range of UUID memory addresses mapped to local memory.
6. **Message Passing Discussion**
   1. In this paragraph, we will discuss the main trade offs associated with message passing. The main trade off is between reliability and innate efficiency of the message passing protocol. Take for example the “send and forget” message passing protocol. This method is the most efficient but least reliable when compared to other methods because it is cheap and quick but may cause more errors/failures when trying to send and receive data. On the other hand, “Guarantee delivery for every message” is the least efficient in terms of speed and cost but is the most reliable because it will make sure that the message will get correctly passed to the receiver. A general rule of thumb is if you don’t care about efficiency, use the most reliable protocol (i.e. for web use TCP/IP). And if you do care about efficiency, then implement only the necessary components because this will save space and time. The following are some problems that are inherent with messages:
      1. Receive buffer full
      2. Sender can’t wait, but send buffer is full
      3. Message gets lost
      4. Acknowledgement gets lost
7. **Event-Based Architecture**
   1. Event-based architecture involves processes subscribing to the event bus so that when another process publishes an event to the event bus, only the subscribed processes will receive the message. The middleware is responsible for the arrangement that only subscribed processes will receive that specific type of event that they are subscribed to. The processes do not need to know about each other since they are referentially decoupled. In short, processes publish and subscribe to the event bus so that they can send and receive a certain type of event. Shared data-space architecture is very similar to event-based architecture in the way processes publish and subscribe to events. Instead of an event bus though, the processes publish and receive data from a shared/persistent data space. The events must be stored in a persistent form because the events are time decoupled. Overall, both event-based and shared data-space architecture share strong similarities in the way processes are able to send and receive data.
8. **Discussion of Thin/Thick Client/Server**
   1. There are many pros and cons to having a lot of code on the client and vice versa with heavy implementation on the server. For a thin client thick server, there will be more packets, more exposure and a higher chance for profiling sender/receiver patterns. The biggest pro of having a thick server is the ability to not have to distribute updates to each client so it makes updating code much easier. This makes coding very simple because most of the components will be in one place. However, there will be a lot of inefficient traffic over the network, which will lead to slow interactive client response. In For a thick client, there will be much more code exposed on the client and a higher chance for out-of-date, insecure client code. This means that when you want to update code, you would need to make sure that each client will get the updated code since the code lives in many places at once and versions could be out of sync. However, having a heavy client can be beneficial by having less network traffic, quick interactive response, and be able to handle many exceptions locally and gracefully. In short, there are many things to consider when coding a server and client. Depending on your goal and how you plan on handling client update problems, you need to note all of the pros and cons that come from having a thick or thin client.
9. **Bit Torrent**
   1. Bit torrent is the topic of peer-to-peer connections being able to connect with each other to download and distribute files. Bit torrent is both structured and unstructured meaning the torrent files and trackers are the centralized or structured components, while the regular nodes are the decentralized or unstructured components. So how does Bit torrent work? Basically, torrent files are fixed locations that point to trackers (servers) that have a list of active nodes that have a certain file. Nodes are then able to download this file in the form of chunks from different nodes containing the file until the node downloads the full file. Another thing to note is that nodes can download and upload a file simultaneously and it is even possible for two nodes to trade pieces of a file. Seeders are servers that remain connected to the main swarm of nodes even when they are not actively downloading from it. This helps in being able to distribute the file since there are more connections and available nodes to download the file from. On the other hand, there are leechers who are servers that do not allow upload and only act to download files. Although this is locally optimal, this does not benefit the swarm since there are less nodes available to download the file from. The main idea that needs to be implemented in a peer-to-peer file sharing service is to reward good behavior, like being a seeder, so that it will encourage the use of the system and have a more beneficial community for distributing files.
10. **Migrating Code**
    1. The topic of migrating code is a very important topic when designing large scale systems as it can be costly or hard to implement. This involves one process moving from one machine to the other. There are two models for code migration: weak mobility and strong mobility. Weak mobility is when you move code to a new location, the process needs to start over. This means that the sender process executes the process and then in the receiver process, the same process is executed again. Strong mobility is the opposite where you are able to migrate code to a new location without restarting the process. This means that when the sender is initiated, the process is cloned and sent to the receiver and then the receiver is able to continue the process without restarting it because it is sent a clone. There are three ways in the way processes are binded to the resource: By dentifier, by value, and by type. Binding by identifier is the strongest form of process-to-resource binding because it is bound by the absolute name, thus the absolute thing. Binding by value is weaker than by identifier as it uses the standard libraries and copies of a library to bind the process to the resource. The weakest of the three is binding by type and this means that it uses a reference to bind the process to resource.
    2. This also brings up the question whether a resource can be moved in the first place. If the resource is unattached, like a data file specific to the application, then you are able to freely move it. Fastened resources can be theoretically moved, however it is not practical in terms of efficiency. A fixed resource cannot be moved whatsoever. There are multiple ways to handle migration and they can all be combined to make things easier. The method of pushing involves memory pages are pushed to the new machine and then later modified memory pages are resent during the migration process. Another way is to stop the current virtual machine, migrate the memory, and then start up the new virtual machine. The final method is to let the new virtual machine pull in the new pages when they need it. In other words, the processes start immediately on the new virtual machine and the memory pages are copied on demand.
11. **Server Socket**
    1. In our projects, we used a server socket for every single project because we needed to learn the ins and outs of a server client connection. In a server socket, the process blocks while listening for requests. Once it gets a request, a socket is returned and the server socket goes back to listening for more requests. What we implemented in our projects was a doorbell socket where we spawn worker threads whenever we get a connection and then return to listening. One thing to note is the queue size specified where simultaneous requests may be dropped if the buffer gets full.
12. **Marshalling/Unmarshalling and Serialization**
    1. Marshalling is the process of translating data or objects into a byte stream while unmarshalling is the process in which a byte stream is translated to an object. These processes were very useful in our Blockchain project because we needed to send objects back and forth between the server and client and in turn, we needed to unmarshal the data back to an object. A main discussion in the lectures were how it is possible to automate the process of marshalling/unmarshalling. It is generally considered hard and unsolvable to automate because it is impossible to create a lower-level middleware algorithm to marshal without knowing anything about the data. When using introspection, by knowing the object’s properties or type, this information would need to be marshalled as well. You also need to include when objects refer to other necessary objects as well. Java supports introspection through serialization. Serialization is the process of creating a string of bits to be processed one after another that can be accessed in any order.
13. **Remote Procedure Call**
    1. A crucial topic discussed in the lectures were the steps for a remote procedure call and the interaction between the client stub and server stub. One main topic to remember are the three files output by the IDL compiler. THese files are the header file, the client stub, and the server stub. The steps for a remote procedure call are as follows:
       1. Client stub builds message and then calls the local OS
       2. Client’s OS sends message to the remote OS
       3. Remote OS gives message to server stub
       4. Server stub unpacks parameters and then calls the server
       5. Server does work and then returns result to the stub
       6. Server stub packs it in a message and calls the local OS
       7. Server’s OS sends message to the client’s OS
       8. Client’s OS gives message to the client stub
       9. The stub unpacks the result and returns to client
    2. For the client, the exchange protocols for a remote procedure call are the request or the acknowledge reply while for the server there is only the reply protocol. If there are timeouts then this is related to a server binding problem.
14. **Symmetric Key Encryption and Public/Private Asymmetric Keys**
    1. Symmetric key encryption is an overall very efficient and common form of encryption since it is cheap, fast, and has very good encryption. In Symmetric key encryption, both parties share the same key and are able to access the encrypted data. However, there rises a problem in distributing the key since there is only one set and it is dangerous to share the key. With asymmetric keys, both keys are functionally the same but one is kept a secret (private) and one if bound to the stakeholder publicly (public). What makes this method more secure is that you need both of the keys to encrypt and decrypt. The public keys are published in some type of data structure like a phone book that is available to all of the trusted certified sites. Public and private keys are used in the following examples:
       1. If person A wants to send a secret message to person B, then person A will encrypt it using person B’s public key.
       2. Person B can only read this message because it can only be decrypted using person B’s private key that only person B knows.
       3. If person A wants a signed copy of a message from person B, then person A can request the person B encrypts the message using person B’s private key.
       4. When a document is signed by an author, this will irrefutably authenticate them as the author. This can be unauthenticated by the author claiming that they exposed their private key.
15. **Verifying Blocks and Adding to the Blockchain**
    1. The topic of the Blockchain was very complicated in this class but when breaking down the core concepts it becomes easier to understand. Firstly, for a block to be verified, work needs to be done in order to solve a puzzle for that block. Different peers will compete against each other to try and solve the puzzle first to verify the block. Once a peer is able to solve the puzzle, they get a reward and the block is verified. All of the other peers will then give up on solving it and wait for the next block to solve. A crucial topic for the blockchain is the shared ledger. This ledger is available to all peers so that they can prepend the verified blocks to it. This is encrypted in a way such that it cannot be counterfeited by any one process working alone.
    2. Inserting the newly verified block into the actual Blockchain is not as straightforward as you may think and requires some extra steps for security. First, the random string solution for the puzzle is inserted into the current block as well as the next sequential block sequence number of the blockchain. A SHA-256 hash will then be created from the current block with the random string solution inside it. This hash will be added to the header. The new block will now be added to the blockchain and a copy of the updated blockchain will be multicast to the other nodes so that they will receive the latest version.
16. **Oracles and Certification Sites**
    1. An oracle is some type of site or service that is to be trusted so that public keys will be able to be linked to stakeholders. In more detail, they are trusted entities that post their public key and will present to stakeholders the bindings of public keys. There are special certification sites where oracles are able to create other trusted certification sites. The most important element when using oracles is trust so it is crucial that it everyone trusts that the public key for the certification site is real and the secret key for the site has not been leaked. There are two different oracles, first-level oracles and full oracles. What defines a first-level oracle is if an oracle gets busy, it is able to give other sites the ability to authorize stakeholders. Full oracles, on the other hand, can authorize other sites to have the ability of authorizing other oracles when it gets busy.
    2. The importance of these certification sites is to guarantee the authenticity of public keys, AKA certify keys, and to also give out public and private key pairs. These sites are available through third party vendors and once a certification authority has been established, other sets of public/private keys can be certified in various ways by it. In terms of the stakeholder’s responsibilities, they are to make sure that their private key stays a secret. Since stakeholders can be bound to their public keys, once their public key has been certified, you can send them secret messages.
17. **Clock Synchronization**
    1. Clock synchronization algorithms are very important because we need our components to be synchronized in our synchronized system. I know that sounds pretty obvious but it is very crucial when coordinating many different processes at the same time since certain events need to be triggered or completed precisely. For example, when coordinating processes on different machines, each machine will have its own clock and it is crucial for the timings for each machine need to be precise or near perfect. An event that occurred after another event may nevertheless be assigned an earlier time when dealing with multiple machines and they have their own clock.
    2. Clock synchronization can be useful when using GPS because you can use the time you received a signal from the satellite to see how far away you are from a location. For example, if you know the location of the satellite as well as knowing what time you have received its signal, then you can determine how far away you are from it denoted by the radius of the plane. You can use two radii to know where you are on the plane while ignoring the upper intersections.
    3. The Berkeley algorithm is used when no machine has an accurate clock. You use this algorithm to make an agreement between machines since their clocks are not accurate. What happens is that the clock is set manually from time to time based on an external source. Overall, adjusting clocks can be problematic since time cannot run backwards and some clocks are known to be more accurate than others.
18. **MulticastTree from Distributed Hash Table**
    1. The basics of MulticastTree from DHT has many properties including:
       1. Any DHT node can accept multicast commands
       2. Nodes can come and go from the DHT at will
       3. Nodes can join up with and drop from the multicast group at will
       4. Multicast nodes can be DHT members, or linked from DHT members.
    2. Multicast groups are formed by generating a multicast ID, which is a random value that acts as the name of the multicast group. This name is then handled by the appropriate DHT node. To be multicasted, the build message is sent to any DHT node and this gets passed along to the right DHT node, which becomes the root of the multicast group where messages are processed or forwarded to other nodes. When sending multicast messages, they are passed up to the root for the multicast group and then root processes the request and then issues the multicast.
    3. There are essentially three types of DHT nodes: not (yet) participating, pure forwarders, and forwarder members. Nodes that want to join the multicast group are theoretically independent of the DHT. If a DHT node has never seen a request before, it becomes the parent of the new node in the multicast tree and it becomes the forwarder. If the DHT node has previously seen the request for a node join the multicast group, then it adds the previous node as its own child and doesn’t forward the request up toward the root since it has already done so. Essentially to change from a forwarder to member, you simply flip a bit and to change a member to a forwarder, you flip the bit back.
19. **DHCP and NAT**
    1. DHCP means Dynamic Host Configuration Protocol and it is a network management protocol for UDP/IP networks where IP addresses are allocated when they are needed without administration from a pool of valid, real IP addresses. The DHCP client broadcasts a query to the DHCP server when a user boots their machine and this process returns a valid IP address, a base length, subnet mask, and a default gateway. There are three methods of DHCP allocation:
       1. Dynamic: the client is given whatever is available from the pool at the time of the request. This method is easy to implement and needs no administration.
       2. Automatic: It is the same as dynamic but you must keep track of what you’ve allocated and always give that IP address to the client.
       3. Static: Keep the MAC address of the clients in a table.
    2. NAT means Network Address Translation and it allows more computers to use the internet than there are real IP addresses. NAT works by the local host sending a request packet to the internet via the NAT-enabled router. The router then saves this request in a table for IP/Ports. The IP/Port is then replaced with the router’s IP address and a virtual port that indexes the entry in the table. A router IP/virtual port is then sent to the destination and when the server replies, it uses that router IP/virtual port. Once the router receives the reply, it uses the port to lookup in the table and replace it with the new router IP/virtual port. This is then sent to Ethernet and the original sender retrieves from the local network. The entries are then stored for a while then discarded.
20. **Password Encryption**
    1. Everyone has entered their username and password for their account but have you ever thought how this password is saved and encrypted/decrypted while also keeping security within the site? Well as a general rule, you should never save plain text passwords on the database because we never need it. The user will register their password over SSL and it will encrypt a new string using the password. The result is then saved in the database. Salt is encrypted along with the string which basically makes the password much longer and harder decipher to the unknown eye. When the user logs in with their password over SSL, the password they entered is encrypted along with salt and is compared to the stored string. If they are the same then they are allowed entry.
    2. A dictionary attack is an attempt to steal private data from a computer system by using a dictionary head list to generate random passwords. This involves building a dictionary of words and word-ettes that are added to the second column of an array. Each word is then encrypted and the string returned is put into the second column of the array. The attacker would then sort on column one using binary search to try and find the correct password. Salt is useful in preventing the succession of this attack since it increases the length of the string and lowers the chance the attacker to get the right password.

**Forum Postings**

**Started thread “Use of other Java Libraries?”**

* Are we allowed to use other Java Libraries besides the starting ones given (Java.io and Java.net)? For example, what if we wanted to use something in Java.util?

**Replied to the thread “JokeServer Data Structures and Compromises”**

* Hello Mitchell,

I had a similar idea of utilizing a HashMap for checking the UUID from the client's state on the server side. However, I found difficulty in implementing this in my code so I decided on another implementation that Prof. Elliott suggested in the joke-state.html page. This implementation was to send over the State for the jokes and proverbs from the client to the server and supply the proper joke or proverb accordingly to what the mode the server was in and what the client was missing. The server then sends the joke or proverb back for the client to display and the appropriate state gets updated and sent back. I kept track of the states with a string for each state that I would modify and send back and forth with each connection with the client and server. I believe I have a similar implementation for the admin setting by creating a separate thread in the JokeServer class to strictly handle connections from the JokeClientAdmin on a different port. I found this implementation of not using a UUID personally easier for me to grasp as opposed as using a UUID to get the state of the client. It is quite interesting to read these other implementations for these programs.

**Replied to the thread “Multiple Server Checklist”**

* Hello Steven,

I believe this part of the checklist is referring to the arguments passed in on the command line for the secondary server utilization. For example, in the Joke Client you would need to specify a second IP address in the command line in order to enable secondary server functionality. This is just my assumption since this entry on the checklist is under the MULTIPLE JOKESERVERS section. If I got this wrong please let me know.

* Oh ok I think you're right. The second server should run in a separate command window and the joke client can toggle between primary and secondary. I think when you put the "secondary" argument into your jokeserver you start to run only the secondary server on a different port. So from my understanding you have one primary server running strictly running on one port in one window and one secondary server running strictly running on another port in another window. The client can then toggle between which server to connect to by switching which ports it connects with. Again, if my understanding is incorrect please let me know.

**Replied to the thread “JokeServer and ClientAdmin”**

* From my understanding, when you specify that you are running the secondary joke server, you will need to run the secondary admin as well. So if you are only using one server then the second admin server does not need to be running. I believe you need to specify an argument for JokeServer that you are running the secondary server so that the secondary admin can run as well. As for the JokeClientAdmin, it would start out by having the ability to connect to primary server and change the modes. If you enter something to switch to the secondary server then it will only communicate with the secondary server. I believe that it is similar to the JokeClient where by entering 's' you can toggle between primary and secondary server. Hope this helps.

**Study Log**

* **The following notes are topics that Prof. Elliott highlighted in the lectures like having to put items “in our toolbox” or material he went into great detail. I felt these were important topics and I note them down in the sequential notes. I mainly took notes from Prof. Elliott’s lectures and these reflect the notes I took.**
* **IDL: Interface Definition Language**
  + Specify the names of procedures or methods, and the precise bitstreams that go through the interface in each direction.
  + The main purpose of the IDL → define the interface.
  + There may be a large amount of score sets of interacting and possibly conflicting policies for large distributed systems.
    - Client stub for calling the server automatically generated
    - Server stub also generated automatically by IDL
  + Local call sequence is between parent and child, basically calling routine and subroutine.
  + In a remote call structured by IDL the calling routine communicates with the client stub which in turn communicates with the server stub and then to the called subroutine.
    - This then loops around back to the calling routine.
* **Characteristics of fully decentralized (distributed) algorithms which help with scalability**
  + No machine has complete info about the system state
  + Machines make decisions based only on local info
  + Failure of any one machine does not ruin the algorithm
  + THere is no implicit assumption that a global clock exists.
* **Administration is the most overlooked scale-up problem**
  + The design of systems needs to change rather than just the size of the resources.
  + Generally not algorithm-based structure so it is overlooked in scale-up discussions
  + It is often the most critical bottleneck because it may require fundamental changes.
* Synchronous calls use one process and use blocking calls to block and wait until the event happens.
* Asynchronous calls are non-blocking and use two processes so does not need to block.
* **Transactions**
  + Main thing is before and after state must have the same amount
    - Bank transaction example
  + Need to use locks to make sure no more than one process is in the critical section so data will not get messed up.
  + No shared memory in distributed systems so no support for critical sections.
    - Need to send messages using IPC (Inter-Processed Communication) but is unreliable because there are multiple problems with messages.
* **Process**
  + A computer program that is being executed
  + Can be true in both local and distributed systems
  + One program can run in many processes at the same time
* **Context Switching**
  + Processes are continuously swapped in and out of the CPU so that every process gets a chance to run for a while.
  + Switch by using IPC for the processes to talk to each other.
* **Test and Set Instruction**

1. Look at semaphore. If true, then return to A, else if False then set to true.
2. Complete Critical Section
3. Set semaphore back to false
4. Process X completes to B, then is switched out
5. Process Y loops to A continuously
6. Process X switched back in, completes CS and C
7. Process Y completes CS and C

* There is a perfect solution for universal shared memory in distributed systems but it is too hard to implement
* **Tradeoffs between reliability and innate efficiency**
  + Need to always think about tradeoffs between reliability and innate efficiency of the message passing protocol.
  + “Send and forget” is the most efficient and least reliable
  + “Guarantee delivery for every message” is possibly the least efficient and most reliable.
  + If efficiency is the least important → use the most reliable protocol
  + If efficiency is the most important, → implement only what you need, when you need it.
* **Problems with messages:**
  + Receive buffer full
  + Sender can’t wait, but send buffer is full
  + Message gets lost
  + Acknowledgement gets lost.
  + As you can see there are multiple things that can go wrong when sending messages in distributed systems. That’s why using a local system is optimal.
* **Problems using send/receive**
  + Sender and receiver have to be active and know each other’s address or endpoint.
  + Buffering concerns.
* **Asynchronous Calls**
  + Minimally → send and forget
  + Maximally → wait until the server application starts work on the request
* **Event-Based Architecture**
  + Components publish and subscribe from the event bus.
  + The middleware arranges only processes that are subscribed to the event bus for a certain type will receive the events published.
  + The processes are referentially decoupled so they do not need to know each other.
* Shared (Persistent) data space is similar to event-based architecture but events are published to the shared data space instead of the event bus and components can retrieve this data. However, processes are time decoupled so the events must be stored in some persistent form.
* **Thick Client**
  + Pro:
    - Less network traffic
    - Quick interactive response
    - Handle many exceptions locally and gracefully
  + Cons:
    - **Really hard to update code across clients**
    - Code lives in many places at once, versions out of sync
    - More difficult to administer
    - System errors must be handled by client
* **Thin Client**
  + Pros:
    - **Do not need to distribute updates**
    - Simple to code because everything is in one place
    - Simple to maintain and monitor
  + Cons:
    - Lots of inefficient traffic over the network
    - Slow interactive client response
    - Increase computational and data burden on server
* **Adding a Node to Distributed Hash Table**
  + Use RNG to produce some number in the correct key range
  + Look up the next highest numbered node, make successor
  + Insert new node in between predecessor and successor. Move data/processing code from successor as needed.
  + Removal of node is the reverse, including moving data/processing node
* DHT is very powerful and flexible
  + Deterministic scheme to map key of ata item to identifier of some node to store it.
  + Look up data by same key
  + Nodes can join and quit at will
  + Rapid communication between nodes.
* **Bit Torrent**
  + Peer-to-peer system for distributing files that is both structured and unstructured.
  + Nodes exist in swarms and for any file, at least one set of nodes must have a complete copy of the file.
  + The torrent files point to trackers (servers) and the trackers keep lists of active nodes with the file.
    - They download chunks of the file from other nodes
    - The nodes can upload and download the file simultaneously
  + Seeders remain connected to the swarm even when they are not downloading from it (good).
    - This is beneficial since more nodes are able to download the file from it, thus making the file more available.
  + Leechers don’t allow upload and only download (bad).
    - This is not locally optimal because leechers are not benefitting the swarm.
  + There needs to be a reward for good behavior built into the design
* **Code Migration**
  + Weak Mobility: When moving code to a new location, the process needs to start again.
  + Strong Mobility: When migrating code, the process does not need to restart
  + 3 ways to handle migration (can be combined)
    - Push: push memory pages to the new machine and resending the ones that are later modified during the migration process.
    - Stop: stop the current virtual machine, migrate the memory, and start the new virtual machine.
    - Pull: start on the new VM immediately and copy memory pages on demand.
* **MIME type**
  + Multipurpose Internet Mail Extension
  + Typically determined by a server from the file type of the data file
  + What we used in our project to send an HTML or text page over and display it properly in the browser
  + Used in MyWebserver project
* **UDP**
  + Universal Datagram Protocol
  + No acknowledgement, resend or guarantee
  + Cheap
  + Receive method returns address and port of sender as well as the data
* Doorbell socket is mainly what we used in our projects where we block while listening for requests. When request is received a socket is returned and the server socket goes back to listening.
* **Marshalling/Unmarshalling**
  + Marshalling is the process of transforming a representation of an object that was used for storage to a form that is executable. Unmarshalling is the reverse process.
  + Used in our block chain assignment to send data in different forms to the client and server for the xml page
* **Serialization**
  + The process of creating a string of bits to be processed one after another, from a collection of bits that can all be immediately accessed in any order.
  + When objects refer to other necessary objects, they have to be included in the serialization as well
* **Request Exchange Protocols**

1. Client Request
2. Server Reply
3. Client Acknowledge Reply

* **Steps of a Remote Procedure Call**

1. Client stub builds message, calls local OS
2. Client’s OS sends message to remote OS
3. REmote OS gives message to server stub
4. Server Stub unpacks parameters, calls server
5. Server does work, returns result to the stub
6. Server stub packs it in message, calls local OS
7. Server’s OS sends message to client;s OS
8. Client’s OS gives message to client stub
9. Stub unpacks result, returns to client

* **3 Files output by IDL compiler**
  + A header file
  + The client stub
  + The server stub
* **Bigger Buffer Reduces Jitter (Usually common way to solve problems)**
  + Constant average delivery is timely
  + The bigger the bufferm the better extremes are handled
  + More delay in startup to fill the buffer part way
  + Always a tradeoff and prediction is important
* **Symmetric Key Encryption**
  + Most common, Cheap, fast, and very good encryption
  + Workhorse of cryptography
  + The main problem is distributing your key
* **Asymmetric Key Encryption**
  + Use of Public and private keys where both keys are functionally identical but one is private and the other is public.
  + Use the receiver’s public key to send a secret message to them.
  + Use sender’s private key to sign a message
* **Verify Block**
  + Need to solve puzzle to verify block of data
  + Peers compete to verify the block first by solving the puzzle.
  + Once a node solves the puzzle, they get a reward and all the nodes stop working for it.
* **Shared Ledger**
  + All peers share the ledger
  + It is constructed so that it cannot be counterfeited by any one process working alone.
  + Verified blocks are prepended to the shared ledger so that other nodes can see the new updated ledger.
* **Adding to Blockchain**
  + Insert the random string solution into the current block along with the next sequential block number.
  + Create an SHA-256 hash of the current block and put into header
  + Add new block to the blockchain and multicast the updated blockchain to all other nodes
* **Oracles**
  + Trusted entities that post their trusted public key and post the bindings of public key to stakeholders
  + Mainly used to be trusted to link public keys to stakeholders
  + Everyone must trust that:
    - The public key for the certification site is real
    - The secret key for the site has not leaked
  + First Level Oracles
    - If an oracle gets busy, it can authorize other sites to authorize stakeholders
  + Full Oracles
    - If an oracle gets busy, it can authorize other sites to authorize other oracles
* **Clock Synchronization**
  + When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.
  + In short, time is very important in distributed systems
* **GPS**
  + If you know the location of the satellite and you know what time you’ve received its signal, then you know how far away you are from it (the radius).
  + Using two radii you know where you are on the plane
  + You can also use GPS signals to find the time.
* **Network Time Protocol (NTP)**
  + Know time of request and of receipt of response
  + Know processing time
  + Know difference of two = sum of delays
  + Average of the 2 network delays should be added to time at send
* **Berkeley Algorithm**
  + When no machine has an accurate clock, you can use an agreement amongst them.
  + Set the clock manually, from time to time based on an external source.
* **Application-Level Multicast and DHT**
  + Can cross router and network boundaries
  + May not be as efficient as network-level multicast
  + Must construct an overlay-network
  + Organized as a tree where there is one path to each node
  + Or as a mesh network where there are multiple paths between pairs of nodes
  + MulticastTree from DHT
    - Any DHT node can accept multicast commands
    - Nodes can come and go from the DHT at will
    - Nodes can join and drop from the multicast group anytime
    - Multicast nodes can be DHT members, or linked from DHT members
  + Forwarder: If an intermediary DHT node has not seen a request before, it becomes the parent of the new node in the multicast tree.
  + If the DHT node has already seen a request for a node to join the multicast group, it adds the previous node as its own child, and does not forward the request up toward the root.
  + DHT nodes are either not (yet) participating, pure forwarders, and forwarder members.
  + Sending Multicast Messages
    - Passed up to root for the multicast group
    - The root then processes the request and then issues the multicast
  + Change from forwarder to member → flip a bit
  + Change from member to forwarder → flip the bit back
  + DHT can support as many multicast groups as there are ID numbers
* **DHCP - Dynamic Host Configuration Protocol**
  + More efficient to allocate IP address only when they are needed from a pool of valid, real, IP addresses.
  + 3 Methods of DHCP Allocation:
    - Dynamic: give client whatever is available from the pool at time of request
    - Automatic: like dynamic but keep track of what you’ve allocated and always give that IP address to the client
    - Static: keep the MAC addresses of the clients in a table.
* **NAT - Network Address Translation**
  + Allows many more computers to use the internet than there are real IP addresses.
  + NAT assigns users “Fake” (local) IP addresses that cannot be used directly on the internet
  + NAT addresses are not unique
* **Mobile IP**
  + If server push is required, or others are using local resources that are mobile, then DHCP will not work.
  + IP addresses are subnet-based, for routing purposes, geographically fixed
  + Requires home agent and clients to be Mobile IP or have to route all packets twice.
* **IPSec (IP Security)**
  + Operates at a lower level than TSL, SSL, SSH, so applicable to a wide range of applications like TCP and UDP without needing to rewrite apps.
* **Bluetooth**
  + Low battery consumption.
  + Wake up and listen for a paging call, so long latency to establish contact.
  + Switches 1600 times/second between 79 sub-bands of 2.4 GHz public frequency for reduced interference.
  + Designed for quality of service required of audio
* **Master/Slave**
  + Pairs operate as master/slave where master allocating communication slots
  + Piconet: up to 7 slaves , one master, and up to 255 dormant slaves in low power mode waiting for a wakeup signal
  + All devices can be master or slave and all have 48-bit GUID’s but slaves assigned 1-7 to save space.
* **Passwords**
  + Never save plain text password in a database
  + User registers password over SSL → Encrypt some good string plus salt using password → save result in DB
  + User logs in with password over SSL → encrypt some good string plus salt using password → compare to stored string and if equal allow access.
* **Token Ring Algorithm**
  + Have an unordered group of processes on a network
  + A logical ring constructed in software
  + Mutual exclusivity makes it easy to implement
* **Election Algorithms (Bully Algorithm)**

1. P sends an election message to all processes with higher numbers
2. If no one responds, P wins the election and becomes the coordinator.
3. If one of the higher-ups answers, it takes over. P’s job is done

* **SuperPeer selection with DHT**
  + DHT addressing is via m-bit space so you use the leftmost 3 bits
  + When node-ID and 1110000 are not equal zero, then you are a SuperPeer
  + Selects 1 in 8 as a SuperPeer, but can also select 1 in 4, 16, 32.
* **Message Sending**
  + Weighted reference counting so you know about someone being “told” about you.
  + Messages have counters and sender ID and never respond to a message with a lesser ID.
  + Use a reliable message protocol
  + Do global, independent, garbage collection.
* **Data Stream**
  + A sequence of data units
  + Can be used for sending discrete items
  + Be continuous, such as for audio and video
* **Synchronous Transmission Mode**
  + Max end-to-end delay
  + No constraint on arriving early
  + I.e. Sensor data that is aggregated at intervals - must arrive in time, but no penalty if early.
* **Isochronous Transmission Mode**
  + Data units are transferred on time
  + Cannot arrive late or early because the buffer space is unknown
  + Requires bounds on jitter.
* **Properties of Streaming Data**
  + Store data streamed from files
  + Live data streamed from a real-time source
  + “Loop” delayed live streaming data
  + Compressed data and secure data
* **Multiplexing**
  + Need real-time distribution of synchronization specifications as synchronization gets more complex.
  + Multiplex all of the data streams into one and send the sync info too.
  + Can perform sync at sender if multiplexed, when receiver decodes, it is already synced.
* Asynchronous Transfer Mode (ATM): Fast packet switching with suitable quality of service for multimedia transmission.
* **Mobile Agents**
  + Processes, codelibraries, and state that move from one location on a network to another to perform work on a remote location’s behalf.
  + Interact with local resources, such as databases, other processes, and users.