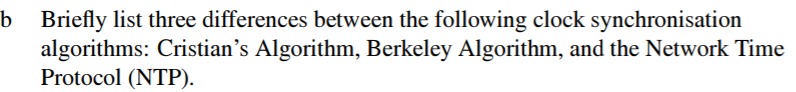


* No need for Interface Definition Language, as the interface is defined using Java (single language system)
* Automated garbage collection for remote objects that are no longer referenced
* No need for specific server stub (skeleton) - server can use a generic skeleton and all required method specific information can be inferred from the client call
* Ability to dynamically load code, e.g. client will dynamically load client stub from the server
* Serialising and marshalling are abstracted away from the user and handled by the RMI.



Cristian’s algorithm is used to sync a single client with another party (server) which has access to UTC. In Cristians’s algorithm the client queries the server (possibly multiple times) and estimates how much it has to adjust its clock based on the time provided by the server.

Berkley algorithm on the other hand applies to synchronization to a common time in a group of machines. Here, a centralized time server is assumed (which may not necessarily receive UTC) which has the role of a master. It polls all clients and calculates best estimate local time (e.g. by rejecting outliers and averaging) and sends to each client by how much that client’s clock needs to be adjusted.

Network time protocol, on the other hand, is used for larger networks ries (top tier) have access to UTC time and essentially propagate it downwards through the hierarchy thro



Not sure about the while(true){}



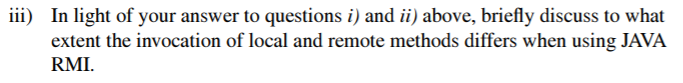


public class TimeLeader extends UnicastRemoteObject implements TimeLeaderI {  
 // Timekeeper list  
 private Vector<TimeKeeper> timekeepers;  
  
 public TimeLeader() throws RemoteException {  
 super();  
 }  
  
 void add(TimeKeeperInterface t) throws RemoteException {  
 timekeepers.add(t);  
 }

void remove(TimeKeeper t) throws RemoteException{  
 timekeepers.remove(t);  
 }  
  
 public long polledAverage() {  
 long sum = 0;  
 for (TimeKeeper tk : timekeepers) {  
 sum = sum + tk.getTime();  
 }  
 return sum/timekeepers.size();  
 }  
  
 private void pushAverage(long average) {  
 for (TimeKeeper tk: timekeepers)  
 tk.adjustTime(average);  
 }  
 }  
  
 public static void main(String[] args) {  
 TimeLeader TL;  
 while (true) {

try {  
 long average = TL.polledAverage();  
 TL.pushAverage(average);

} catch (RemoteException e) {  
 System.*out*.println(e.getMessage());  
 }  
 TL.sleep(5000);  
 }  
 }  
}



Local and remote methods are invoked using the same syntax. The difference is that when making a remote invocation the object making the remote invocation needs to handle remote exceptions.

Parameter passing: when type of a parameter is defined as remote interface, the corresponding argument is always passed as remote object reference à Clients always refer to remote object via remote interface type not implementation class type (This has not been taken into above)

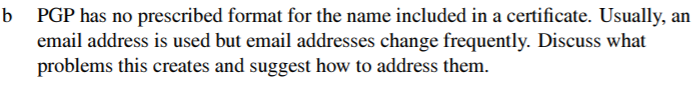
Also the above implementation, may need Registry and SecurityManager



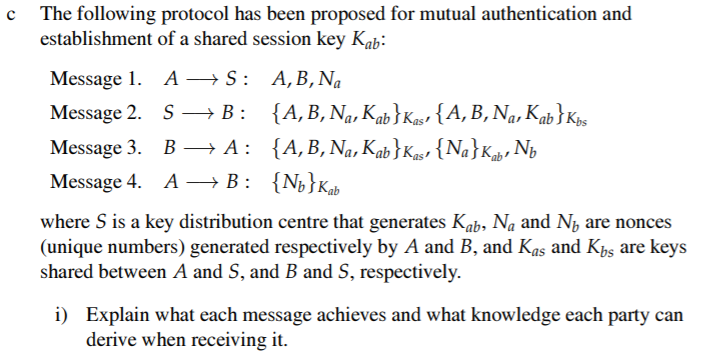
Kerberos is a centralized server which stores sensitive data like server private keys and encrypted user passwords. This system is used to secure access to a number of servers on the same network. The Kerberos server essentially acts as a Key Distribution Center for symmetric (private and session) within the system, authenticating users and generating session keys.

PGP on the other hand offers a distributed way to establish authentication and trust through public key cryptography (and without a central authority). In this scheme, parties can sign each other public keys generating certificates. In this way, Alice can estimate trust in a specific public key by looking at who has generated (signed) the certificate (which can be a number of parties), and aggregating the trust Alice has in these parties.

So these schemes are fundamentally different as Kerberos is (1) usually used for symmetric key distribution and (2) is a centralized system, while PGP is (1) usually used for public key cryptography and (2) is a fully decentralized system.



Since PGP certificate bind a public key to a name, if the name of Alice (e.g. the email address) changes, this would effectively invalidate the certificate. Alice could try to send a notification to other parties prior to the change in the email address in order to inform other parties of intended change of name. This would be signed by Alice’s private key, authenticating the message. Another approach altogether, could be to eliminate this problem entirely by using userID as the name, however this creates a problem of ensuring these are unique and also this convey little information about the actual user (unlike an email which for example includes a domain name).

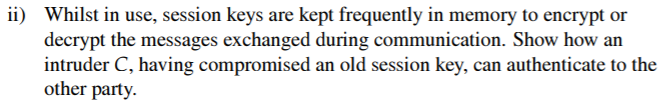


Message 1: Alice notifies KDC that she, Alice (using A), wants to talk to Bob (using B) and provides its nonce which will be used to prove freshness of communication later

Message 2: KDC then generates a session key Kab and sends two parts to Bob. First is the Alice and Bob identifiers, Alice’s nonce and the session key encrypted using Kas (private key between KDC and Alice). Second part is the same but encrypted using Bob’s key Kbs. Hence Bob can now decrypt the second part of the message and understand that Alice is attempting to contact him. Notably he will also know the new session key Kab and Alice’s nonce to prove the freshness of communication to Alice.

Message 3: Bob forwards the first part of msg2 to Alice. Alice, who knows Kas, can decrypt and discover the session key. She can also verify that this key was generated based on her request using her nonce Na. Finally, she also notices Bob’s nonce.

Message 4: Finally Alice confirms to Bob that the session key has been received and sends back his nonce encrypted with the session key to prove freshness of communication.

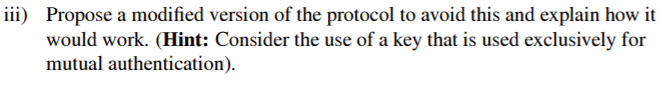


**???**

Intruder C can deploy a rc, by resending Message 3 with the old session key as if it was a fresh key. A would have thought it is communicating with B even though it is in fact C. C would also have Kab to decrypt upcoming communication with A.

Message 3. C --> A: {A, B, Na, Kab}Kas, {Na}Kab, Nb(c)

Message 4. A --> C: {Nb}Kab



B’s nonce is not encrypted and Eve could get that by listening to the communication. To protect the nonce we could change Message 3, and add the nonce to the session key decryption.

B à A: {A,B,Na,Kab}Kas, {Na, Nb}Kab

Message 1: A --> S: A, B, Na

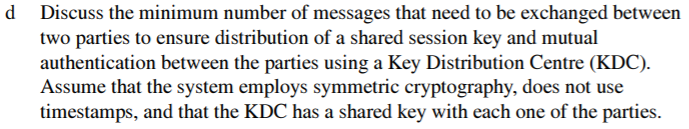
Message 2: S --> B: {A, B, Na, Kab, Ks}Kas, {A, B, Na, Kab, Ks}Kbs

Message 3: B --> A: {A, B, Na, Kab}Kas, {Na}Kab, Nb

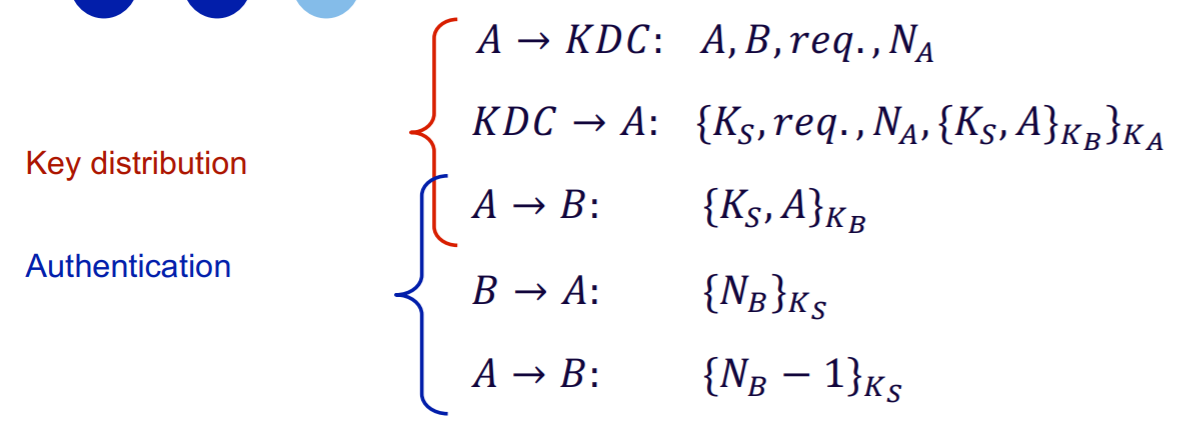
Message 4: A --> B: {Nb}Kab

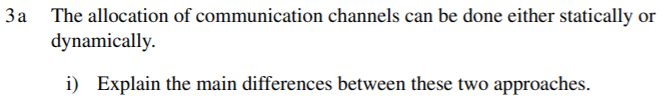
Ks here is another session key that’s used for sending messages back and forth between A and B. This way Kab is only used for authenticating A and B to each other, so if Ks is compromised, C can’t use it to convince A that they are B... doesn’t really help with the whole now can read all their messages thing, but think it’s what they were asking

Here is another proposal: add timestamp



I think this is just using a standard NS algorithm?

Might need to add autentication: https://en.wikipedia.org/wiki/Needham%E2%80%93Schroeder\_protocol



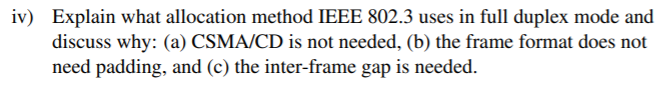
In static allocation, total channel bandwidth is divided among predefined number of users, while in dynamic the medium is used on demand. While static allocation guarantees a bounded bandwidth to all its users, the bandwidth may be underutilized when any user is silent. On the other hand dynamic allocation can lead to more efficient bandwidth usage, but does not guarantee bandwidth to all users. Dynamic allocation also needs strategies to avoid collision, whereas with static binding collisions can’t occur. Dynamic allocation has unbounded latency, static allocation bounded latency.



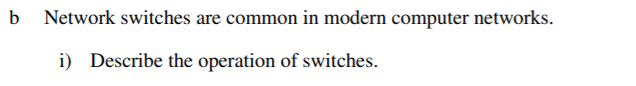
In both TDM and FDM, the users get a defined portion of the medium (either time slot or frequency range respectively). When any user is not sending or receiving data, their piece of the medium is not utilized at all leading to inefficiencies.



Both TDM and FDM rely on synchronization (or cooperation) of all users so that each user is aware of what time slot/frequency range they can use. If the number of users changes, so will the allocate portion of medium (e.g. more users lead to smaller time slot). Hence all users will have to agree and adhere to these new allocations. If users change frequently, this allocation will have to happen frequently as well. In addition, with FDM, you would need to communicate the new frequencies to the communication receivers so that they know what frequency to listen to.



For a full duplex mode, since incoming and outgoing traffic use separate medium, the collisions would not occur. Hence, mechanism to mitigate collisions, such as CSMA/CD (using sensing for collision detection) and from padding (ensuring they are long enough to effectively detect collisions). However, inter-frame gaps are still needed as other sublayers and physical medium are designed to expect gap between Ethernet frames / need time to recover in between frames.



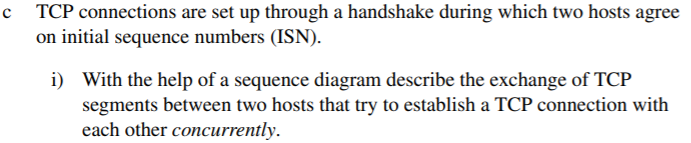
Switches (That’s router, isn’t it? Switches are for connecting devices on a single network, and routers for connecting one network to another) are connected to multiple networks through a number of ports and operate at a data link layer. Their role is to relay traffic from one network (device?) to the next, but only for traffic that is addressed to a recipient (e.g. Ethernet port) in the other network. They operate on the data link layer and use MAC addresses for forwarding. Switches perform a lookup in their internal table to detemine which network a specific MAC is. If a specific MAC address is unknown, they can attempt to find it (e.g. by flooding).

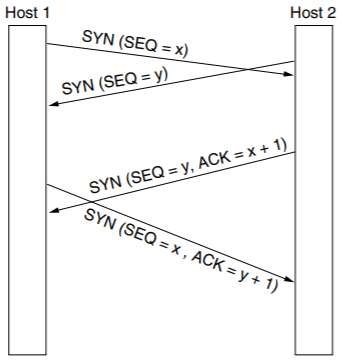
Because switches don’t relay traffic to another network unless the destination MAC is located on another network (or switch does not know where the destination MAC is and hence floods), they can be used to isolate traffic on a specific network from traffic on neighboring networks or to subdivide existing network. This is useful as it limits channel contention (competition for the channel) since a message in one network generally will not cause a collision in another (unless the switch decides to replay it).

Switches may also be used to segment a specific network (LAN) into a number of networks, referred to as Virtual LANs (VLANs). These can be reconfigured in software, based on the needs of the organization which may change, by configuring the switch’s port/VLAN tables.



Switches add 3 types of delay. Firstly, is the delay associated with looking up the destination MAC in the internal table, which is usually a short delay (processing delay). Secondly, switches must accommodate for differing speeds in attached LANs. In order to support this, switches use buffering, which also causes delays. The last type of a delay may or may not occur based on how the switch is configured. Switches may inspect the frame to ensure that the checksum is correct and no errors occurred, causing another delay. This last type of delay is often referred to as processing delay.

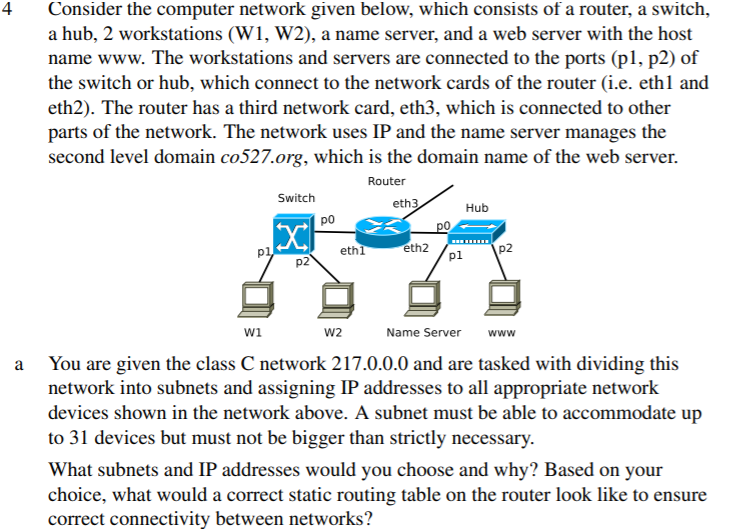




As a result, only a single duplex connection is established.



The purpose of ISNs is to ensure that residual duplicate messages in the network are rejected. Hence, choosing the same ISN for multiple connections will mean that two different packets with the same ISN are present, and the receiver will treat whichever packet is receives later as duplicate. In practice, ISNs are usually derived from clocks, ensuring that this does not happen.



To accommodate 31 devices, we need 6 bit host ID (because 5 bits allows 32 addresses, but 2 of them, all 1s and all 0s, are reserved). We have a type C network 217.0.0.0/24, so we have 8 bits to allocate for subnet and host ID. We have a single router dividing the network into 2 subnets. Hence, we choose 6 bits for host ID and 2 bits for subnet ID (resulting in 4 available subnets, out of which we will only use 2). Let’s use two subnets:

Subnet A: 217.0.0.64/26

Subnet B: 217.0.0.0/26

The assigned IP addresses are:

W1: 217.0.0.1/26 (on subnet B)

W2: 217.0.0.2/26 (on subnet B)

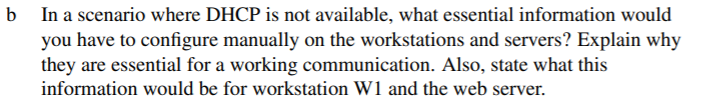
Eth1: 217.0.0.62/26 (on subnet B)

Name Server: 217.0.0.65/26 (on subnet A)

WWW: 217.0.0.66/26 (on subnet A)

Eth2: 217.0.0.126/26 (on subnet A)

|  |  |
| --- | --- |
| **Destination** | **Interface** |
| W1 (217.0.0.1) | eth1 |
| W2 (217.0.0.2) | eth1 |
| Name Server (217.0.0.65) | eth2 |
| Web Server (217.0.0.66) | eth2 |
| Default | eth3 |



DHCP, or Dynamic Host Configuration Protocol, is used to configure machine’s IP address once it is connected to a network. DHCP assumes that each network has a DHCP server which assigns IPs to individual machines. To get an IP address assigned, once a machine connects to a network, it’s NIC (network card) will issue a DHCP Discover packet onto the network which should be received by the DHCP server, which will respond with the allocated IP address (and time to live etc.).

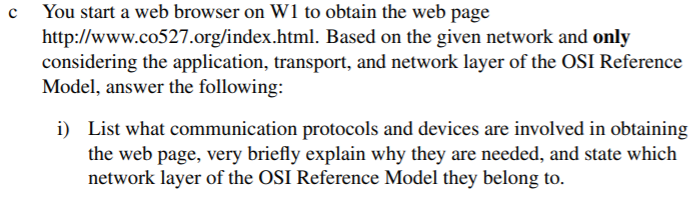
Having a valid IP address is essential to send packets using IP as each packet should specify a source IP address to establish proper communication.

Without DHCP available, IP addresses of individual machine must be assigned and configured manually. This may be acceptable for stationary machines, such as a work station or a webserver. In the case of the example above, W1 workstation will have to configure its IP address to 217.0.0.1 and the webserver will have to do the same using 217.0.0.66.

DHCPs are also used by ISPs to set the parameters of devices over the Internet access link so that customers do not need to call their ISPs to get this information. These include:

* Network Mask
* IP address of the default gateway
* IP address of DNS and time servers

à No need to configure these independently for each host



***This answer is combined for both this and the next part***

First, W1 must identify the IP address of the webserver running [www.co527.org](http://www.co527.org). This is done by the application (the web browser) calling a library procedure resolver and passing the desired host name www.co527.org to it. This procedure sends a name resolution query to the local name server (it is assumed that W1 has the IP address of local NS) using its IP address (127.0.0.65 in this case). This is done using UDP (transport layer), which identifies the standard source port (on W1) and destination port (on local NS). Once UDP datagram is constructed, it is passed down to the network layer which inserts it into an IP frame using the W1 and localNS IP addresses.

Local NS receives this IP packet, extracts the UDP datagram and uses the received host name [*www.co527.org*](http://www.co527.org)to find a corresponding IP address in its internal tables. Since web server is on the same local network as the local NS, it should have the mapping between the name and the IP address and not need to contact any other NS servers. It then copies the entry in its DNS table corresponding to [*www.co527.org*](http://www.co527.org)into a UDP datagram, which is then transferred using an IP packet back to W1. Now W1 knows which IP address.

W1 will now issue a HTTP request to the obtained IP address using a GET command (on the application layer). This would initiate a TCP connection to be set up between W1 and web server (on transport layer).

First, the web browser is bound to a socket on W1, essentially binding the process to a specific port. The TCP connection is set up by W1 transport layer sending an initiation message with the Syn flag and a random Seq number *x*, also detailing its source port and the standard port of the web server. This is sent over the network in an IP packet using the W1 and webserver IP addresses as source and destination (network layer).Webserver, then responds with a TCP packet acknowledging W1 Seq number (Ack = x + 1) and its own Seq number *y*. Finally W1 receives this, and finishes the three way handshake by incrementing its Seq number and acknowledging webserver’s Seq number (Seq = x + 1, Ack = y + 1). The TCP connection is set up and ready to be used.

W1 can now send the HTTP request over the TCP connection. We assume a HTTP 1.1 which uses persistent TCP connection, rather than setting it up for each request. In the header of the request, W1 specifies that this is a GET method and sends the URI of the resource requested (*index.html*). Webserver then receives the request (over TCP connection), fetches the desired resource and sends it back as an HTTP response.

Another thing to mention is how IP packets are routed between W1 and the web server. The IP packet sent by W1 arrives at the router at the IP address of eth1. Router then performs a lookup for the MAC address of the entity corresponding to the destination IP address in the packet (the IP address of the webserver). It is likely that this is already in cache, but if it is not, the router may used Address Resolution Protocol to locate the MAC of the webserver. Once this is done, the router sends the IP packet along that local link. The process happens in reverse for data from the webserver to W1.

