PREM KRISHNA CHETTRI

Distributed System Assignment 1 Submission Date: 05st Oct '15

## Solution 1>

### Advantages of Overlay Networks.

1> Overlay networks allows the visualization of each of the peer over the existing system and so it is scalable and easily transformable to working network.
2> It is virtualization of a system which means, we have are no restriction to the number of nodes the network can handle unlike traditional networks.
3> Due to the virtualization, we can overlap an IP and the node running in a particular IP can further extended to multiple level even within itself.
4> The routing algorithm can be redesigned and all restriction can be changed or modified as and when we need without considering the underlying physical network structure.
5> It allows the scalable model where a peer / node is independent of its physical location and limitation.

### Disadvantage of Overlay Network.

1> It is not always possible to treat a virtual entity treated same as the physical entity. Sometime, different physical entity can have different limitations, controls and policies, which overlays overlooks.

2> Although overlays virtually controls the entire network assuming to be similar units, in reality, each functional unit may have a different challenge when the network actually grows in its full scale. Overlays hide out the details of the network location, which cost in number of serious issues like network delay, latency, bandwidth and the processing capability of one unit from other.

3> Theoretically overlays conceptualize that the network of system is highly scalable but in practical, with each node addition the we need to consider the bandwidth cost which grows along with the network and managing traffics alongside.

## Solution 2 >

Finger table for Node 15

| | |
|---|---|
| 1 | 26 |
| 2 | 26 |
| 3 | 26 |

| | |
|---|---|
| 4 | 26 |
| 5 | 47 |
| 6 | 47 |
| 7 | 75 |

Finger table for Node 106

| | |
|---|---|
| 1 | 110 |
| 2 | 110 |
| 3 | 110 |
| 4 | 127 |
| 5 | 127 |
| 6 | 15 |
| 7 | 47 |

Finger table for Node 127

| | |
|---|---|
| 1 | 15 |
| 2 | 15 |
| 3 | 15 |
| 4 | 15 |
| 5 | 15 |
| 6 | 47 |
| 7 | 69 |

**Solution 3 >**

We need additional finger table for the following hoping as it is as given below.

Finger table for Node 26

| | |
|---|---|
| 1 | 47 |
| 2 | 47 |
| 3 | 47 |
| 4 | 47 |
| 5 | 47 |
| 6 | 69 |
| 7 | 106 |

Finger table for Node 75

| | |
|---|---|
| 1 | 106 |
| 2 | 106 |
| 3 | 106 |
| 4 | 106 |
| 5 | 106 |
| 6 | 110 |
| 7 | 15 |

Finger table for Node 110

| | |
|---|---|
| 1 | 127 |
| 2 | 127 |

| | |
|---|---|
| **3** | **127** |
| **4** | **127** |
| **5** | **127** |
| **6** | **15** |
| **7** | **47** |

For from the above finger tables,

For hoping from Node 15 for key 12. Following hope will take place

15 -> 75 -> 110 -> 127

For hoping from Node 26 for key 3. Following hope will take place

26 -> 106 -> 127 -> 15

**Solution 4 >** With given condition, each node is capable of receive the maximum incoming bandwidth of B(in). As, at any given instant, only one node is allowed to connect with one seed, the maximum download for a node say N1 is possible, when the only connected node say, N1 to N0, which is seeding and is using its full capacity. Even if B(out) > B(in), the node , N1 can have a maximum download value as B(in) which certainly is its maximum incoming bandwidth.

**Solution 5 >** Yes it will work. We can put the native code in all the bytes of the first word not only in the first byte of the first work. So no matter which byte is been checked, the code will always be in there.

Another way to deal this situation is create a byte of 0. When a system sends byte 0 it will arrive as 0 and when the system decode that it's a byte 0, it can safely assume that it is a first byte of the first word and now the code can be retrieve easily.

**Solution 6 >**

As system in a networks can be located at different geographical physical location, the delay in each communication will be substantially higher and increase as two calling parties geographical location increase. Because synchronous communication requires that the caller to blocked until its message is received, it may take a long time before a caller can continue when the receiver is far away.

So, to deal with such situation, we can design an application which does other work while the communication progresses which eventually leading to the form of asynchronous communication.

**Solution 7**> By creating a synchronizing RPC , what we are achieving is the network transmission efficiency. What does that means is, say in our given problem we have, each way latency = 20 ms, marshaling and unmarshaling = 1 ms each , takes 150 ms for each process and and can handle only 1 process at a time.

The time to transmit a response for a client is 1 marshalling + network latency = 21 ms. Now after receiveing on server end, server does again the same thing. So server time for transmitting a response is 1 marshaling + network latency = 21 ms ( **Assuming that we don't consider unmarshalling done at server end and sever transmits as soon as it receives the request** ).

So for 1 process request it takes 21 + 21 + 150 (processing time) = 192ms.

a> For 2 back to back process, it will take = 2 * 192 =384 ms

b> In Async call , we can't reduce this delay either for one particular process request. However, what we can achieve is while one process is getting executed, we can send another process request. This will reduce the amount of time, the second , third process can execute its requests.

So, for async call, client waits for the first response and keep working on it, till it receives the server response. For the time take will be same for a process is same 192. But what interesting happens , it client can issue the next request immediately, without waiting for the server to reply for the first request. So, while server is processing the request 1, client initiates the next request 2 as soon as it has the server response to its reply to request 1.

So, we will have the following total time for 2 back to back request.
Total time = 21(P1) + 21(P2) + 150(P1) + 150(P2) = 342 ms. Instead of 384 ms in Synchronous call

Here, we received request of P2 while we were executing process P1 and we responded the result of P1 while we execute the process P2.

**Solution 8>** Pointer as we know is basically deals with the addresses and references of object in a particular system. It is highly vulnerable that an address space of one system to lay exactly the same with that of other and passing such address space of one system to other across the network is something we should always avoid. It is because of this address space of one system cannot be replicated to another system, we cannot handle pointers in RPC mechanism.

The one way we can overcome this issue is by object serialization. Rather then

passing out the pointer, we can actually pass a value, which the pointer is pointing to. So, we basically can create a stream / file of pointee and pass as a data stream and when the caller receives, it can desterilize data and create this object back to its original data.

### Solution 9>

a> This is "**at-most-once"** case as the server tracks all its previous executed logs and finds the one, if the request has already been served.

b>This is **"at-least-once"** case as the server may re-execute the request if the request comes after more than 1 hours durations. The logs check mechanism fails to find the one that was executed before 1 hour.

c>This again is **"at-least-once"** case as the server will execute all the request that client sends.

d>This falls under **"at-most-once"** case, as the server won't execute any request at all.

e>This again is **"at-most-once"** case, as the server has crashed and not able to execute any request at all.