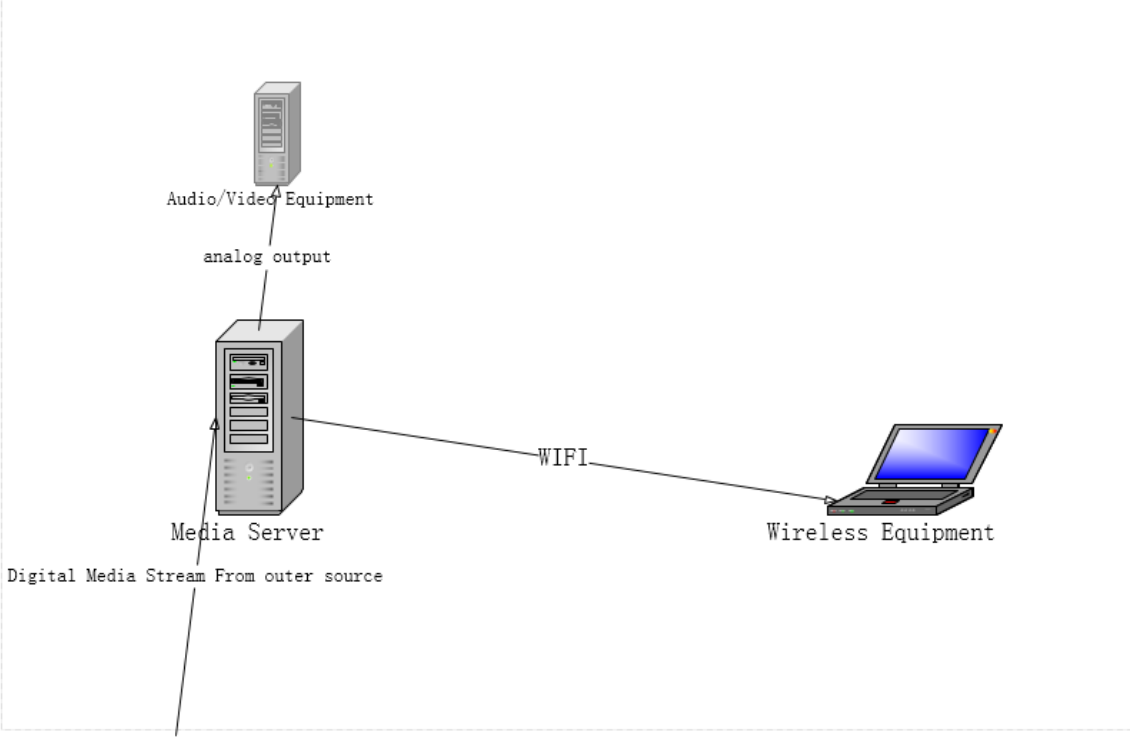
1.



Server:

```
main(){

      while(IsSwtichOn) {
            RawDataStream = PullFromOuterSource();
            MediaDataStream = Filter(RawDataStream);
            UpdateData(MediaDataStream);
      }



}

DataUpdateEventHandler(Object sender, Data d) {
      asyncAnalogOutPut(d);
}
```

```
WirelessClientRequestEventHandler() {
      asyncOutPut(requestedData);
}
```

2. A system is scalable when it is it can grow in one or more of the dimensions without an unacceptable loss of performance and features such as number of components, geographical size, number and size of administrative domains etc.

3. Scaling can be achieved through distribution, replication, and caching.

4. It depends on how the client is organized. It may be possible to divide the client-side code into smaller parts that can run as multiple thread. In this case, when one thread is waiting for the server to respond, other threads are still running. Alternatively, we may be able to rearrange the client so that it can do other work after having sent a request to the server. This last solution effectively replaces the synchronous client-server communication with asynchronous one-way communication. Like one fundamental of design is to never block UI.

5. Performance can be expected to be bad for large n. The problem is that each communication between two successive layers is, in principle, between two different machines. Consequently, the performance between P1 and P2 may also be determined by n-2 request-reply interactions between the other layers. Another problem is that if one machine in the chain performs badly or is even temporarily unreachable, then this will immediately degrade the performance at the highest level.

6. We need to take into account that the outgoing capacity of seeding nodes needs to be shared between clients. Let us assume that there are S seeders and N clients, and that each client randomly picks one of the seeders. The joint outgoing capacity of the seeders is S X Bout, giving each of the clients S X Bout / N immediate download capacity. In addition, if clients help each other, each one of them will be able to download chunks at a rate of Bout, assuming that Bin > Bout. Note that because of the tit-for-tat policy, the download capacity of a BitTorrent client is mainly dictated by its outgoing capacity. In conclusion, the total download capacity will be S x Bout / N + Bout.

7. Each layer must be independent of the other ones. The data passed from layer k +1 down to layer k contains both header and data, but layer k cannot tell which is which. Having a single big header that all the layers could read and write would destroy this transparency and make changes in the protocol of one layer visible to other layers. This is undesirable.

8. If call by reference is used, a pointer to i is passed to incr. all modifications to pointer are exactly modification to data on memory, so the i will be two after increase twice. However, with copy/restore, i will be passed by value twice, each value initially as 0. Both will be incremented, so both will now be 1. Now both will be copied back, although the second copy were overwriting the first one, the final value would still just be 1.

9. Both the client and the server create a socket, but only the server binds the socket to a local endpoint. The server can then subsequently do a blocking read call in which it waits for incoming data from any client. Likewise, after creating the socket, the client simply does a blocking call to write data to the server. There is no need to close a connection.

10. In principle, a reliable multicast service could easily be part of the transport.