Question 1:

Let us consider a simple DC topology with one web server and 200 identical backend servers. The web server scatters requests independently to all the backend servers in a batch in parallel and gathers the responses before sending the aggregated response to the client. The backend servers respond in 5 milliseconds for 99% requests and 200 milliseconds for the rest.

a) What is the probability of waiting for more than 200 milliseconds before the web server has responses from all the backend servers?
   Solution: 0% - if it was formulated “correctly”:
   \[ 1 - 0.99^{200} \approx 0.87 \]  

b) Say, the response time is 5 milliseconds for 99.9% of the requests. What is the probability now?
   Solution: 0% - if it was formulated “correctly”:
   \[ 1 - 0.999^{200} \approx 0.18 \]

Question 2:

In the same setting as Q1 above, say each backend server prepares the response, generates a random number \( x \) between 0 and 1000, waits for \( x \) microseconds and then sends the response back to the web server, instead of sending it immediately.

a) What does this scheme want to achieve?
   Solution: The problem is named TCP incast. In the scatter-gather pattern, it is common to observe many responses arriving at the same time. A high number of synchronized flows to the same destination will cause congestion in the network. In order to prevent this scenario, one possible solution could be to try to desynchronize flows by introducing a random delay. Is there any other solution? The root cause of this problem is actually the standard TCP congestion control algorithm. Look at DCTCP.
b) What is the main drawback of this method?

**Solution:** This scheme could potentially increase the latency. Assuming that there are not so many responses and that we have a network capable to support the generated load, we will introduce an unnecessary delay.

**Question 3:**

To deploy a cloud application that occupies multiple machines, we need to make a decision where to place it - to use the least possible number of racks ("rack local") or to spread it evenly across the data center. What are the arguments for/against rack local deployments?

**Solution:** In a data center we want to exploit data locality to achieve maximum performance (minimum latency), but this is not always possible. Sometimes problems are “bigger” than a rack. Furthermore, when each rack is a fault domain, data has to be replicated across multiple racks for reliability reasons. This allows applications that are not “rack local” to continue the execution even under failure.

**Question 4:**

Does low link utilization always imply less processing at the switch? At a fixed line-rate and utilization, what is the impact of different packet size distributions on processing load?

**Solution:** No, processing at the switch can be related to the link utilization, but not necessarily. A processing overhead per packet is fixed regardless of size (e.g. read the header, figure out where to forward the packet). Therefore, in case of many small packets we can possibly have very high load on the switch, even at low throughput. On the other hand, big packets have small overhead relative to the size of the packet.

**We are happy to give individual feedback in person on request.**