**SIRI’S CLIENT SERVER ARCHITECTURE.**

Siri is a virtual assistant who listens to your requests and performs actions accordingly. It is composed of a “thin” OS –wide client interface in charge of;

Sampling and compressing voice with the Speex audio codec; audio compression codec specifically tuned for the reproduction of human speech.

Sending data chunks to a remote computing facility;

Routing the responses to the OS.

Unlike Google Voice Search which performs the voice recognition for each query on the client side, Siri processes these request on the server side. When you speak something to Siri, it collects your voice and converts it into a data file then your devices passes that information to a remote server which processes the request and then returns an answer in pieces back to your device. Here, internet connection is required. In the apple servers the spoken words undergo different flowchart branches to arrive at a possible solution. The servers compares your speech against a statistical model to estimate, based on the sounds you spoke and the order in which you spoke them, what letters might constitute it. The highest probability estimates get the go ahead. These servers already have a huge database of questions and their probable answers, so there is no problem in getting answers to common questions.

The upside to this method is that it enables server side learning so that the system becomes smarter the more you use it. The more you work with it and make corrections when you’re misinterpreted, the better it gets an understanding of what you mean. Also, because most of Siri’s brain resides on remote servers accessed by many people, the more it is supposed to learn from everyone else.

The downside is that when the queries are long, there can be a lot of back and forth http requests to get an answer.

**HOW SIRI DEALS WITH TRAFFIC: BASIC TRAFFIC ANALYSIS OF SIRI.**

Now to ensure a high degree of responsiveness, the device continuously communicates with the back office, leading to traffic intense behavior. This becomes more critical when using Universal Mobile Telecommunication System or evolutionary loops which can have time varying bandwidth and delays and battery or data billing issues. Nevertheless,

When in the presence of a vast volume of users, the produced load can be highly fragmented, thus forcing the network operator to deploy proper quality of service.

To avoid latencies, as soon as the required processing is completed, the server sends data back to the device.

Delays between consecutive packets are very bounded and constant and therefore the application is not affected by jitter. On the average, the Interpacket time is of 8.12 and 4.73ms from phone to server and vice versa. Larger values are usually due to hesitations or pauses performed by the user. Therefore with enough bandwidth, the hardware and remote computational facilities can ensure excellent responsiveness.

Data produced by phones is mostly composed by voice, producing PDUs (protocol data units) of 800-1500 bytes. The resulting payload is large enough to avoid packetisation delays without generating too small packets. On the contrary though, PDUs carrying server responses are mainly 100-600 bytes, reflecting their textual nature. Further aggregation has to be avoided because collides with the constraints of pushing back information as fast as possible even at the cost of producing tiny packets. This prevents latencies or a poor quality experience but increases the bandwidth usage, mainly due to HTTP/TCP/IP headers.