What's the difference between sockets (stream) vs sockets (datagrams)? Why use one over the other?

http://stackoverflow.com/questions/4688855/whats-the-difference-between-streams-and-datagrams-in-network-programming#4688899

A stream socket is like a phone call -- one side places the call, the other answers, you say hello to each other (SYN/ACK in TCP), and then you exchange information. Once you are done, you say goodbye (FIN/ACK in TCP). If one side doesn't hear a goodbye, they will usually call the other back since this is an unexpected event; usually the client will reconnect to the server. There is a guarantee that data will not arrive in a different order than you sent it, and there is a reasonable guarantee that data will not be damaged.

A datagram socket is like passing a note in class. Consider the case where you are not directly next to the person you are passing the note to; the note will travel from person to person. It may not reach its destination, and it may be modified by the time it gets there. If you pass two notes to the same person, they may arrive in an order you didn't intend, since the route the notes take through the classroom may not be the same, one person might not pass a note as fast as another, etc.

So you use a stream socket when having information in order and intact is important. File transfer protocols are a good example here. You don't want to download some file with its contents randomly shuffled around and damaged!

You'd use a datagram socket when order is less important than timely delivery (think VoIP or game protocols), when you don't want the higher overhead of a stream (this is why DNS is primarily a datagram protocol, so that servers can respond to many, many requests at once very quickly), or when you don't care too much if the data ever reaches its destination.

To expand on the VoIP/game case, such protocols include their own data-ordering mechanism. But if one packet is damaged or lost, you don't want to wait on the stream protocol (usually TCP) to issue a re-send request -- you need to recover quickly. TCP can take up to some number of minutes to recover, and for realtime protocols like gaming or VoIP even three seconds may be unacceptable! Using a datagram protocol like UDP allows the software to recover from such an event extremely quickly, by simply ignoring the lost data or re-requesting it sooner than TCP would.

VoIP is a good candidate for simply ignoring the lost data -- one party would just hear a short gap, similar to what happens when talking to someone on a cell phone when they have poor reception. Gaming protocols are often a little more complex, but the actions taken will usually be to either ignore the missing data (if subsequently-received data supercedes the data that was lost), re-request the missing data, or request a complete state update to ensure that the client's state is in sync with the server's.