1.

The properties that make the TCP protocol important are:

- Connection-Oriented: TCP ensures that the two devices establish a connection between them before they exchange data; and this will probably prevent potential data loss.
- Reliable: TCP is reliable because it checks the integrity of the data that has been sent and received to ensure that the data gets to its destination. And data retransmission is done if the need arise.
- Bidirectional: If a TCP connection is established, then the devices on the connection can send and receive data, regardless of which of them has started the connection.
- Stream-Oriented: Most low-layer protocols are designed, as stream-Oriented so higher-layer protocols must send data in blocks. But using TCP allows applications to send a continuous stream of data for transmission.
- Acknowledged: What makes TCP reliable is that transmissions are acknowledged. The receiver must acknowledge the sender for each piece of data that is transmitted.
- Multiple connection and End point Identification:
  Connections of the TCP are identified by the sockets of the two devices that are connected so this allows each device to have multiple connections opened and handle them independently without conflicts.

2.

My application protocol is like: it sends and receives the data as buffer array, which is limited to 10000 characters because I assume that data exchange between the server and the client can't be more than 10000.

3.

As far as machine architecture is concerned the CPUs store data in either big or small endian format depending on the processors architecture.

Big endian is the byte order where the most significant byte is stored first while the

little endian is the byte order where the least significant byte is stored first.
Since it is impossible to predict the type of the byte order for both systems that are exchanging data, network protocols must define a standard byte order so that both end points can convert to in order to exchange data smoothly. And that is the Network byte order (big endian). Big endian byte ordering has been chosen as the "neutral" or standard for network data exchange.
My program will work if I compile and run the server with a big-endian architecture while my client is on IFI's Linux-machines although ifi machines does not support big-endian, and the reason is that I have used methods that convert from and to network byte order respectively and they are: ntohl() and htonl() for ports.

Big endian byte order

Byte 0	Byte +1	Byte +2	Byte+3
01	02	03	04

Small endian byte order

Byte 0	Byte +1	Byte +2	Byte+3
04	03	02	01

4.
If my program uses connection-oriented communication will avoid the possibility to happen one of these:

- Link failure
- Non-sequenced data.
- Delay
- Errors
- Duplicate data
- No acknowledgement
- Lost Data

5.

Linux and Berkeley socket API offers polling method opportunity to develop servers that can create multiple connections simultaneously. We can use select (), poll () and epoll () to implement network applications with non-blocking socket I/O. But choosing one of them depends on the application requirements.

Basically the polling methods have the same functionality but they differ in details. *Poll* () is a newer polling method, which probably is much better designed and doesn't suffer from most of the problems which *select* has.

In my program I have chosen to use select () because of these reasons:

- Select () existed for some time before poll (), so it is sure that every platform that has network support and non-blocking socket will be working on select () but it might not work on poll ().
- Select () handles the timeout within one nanosecond precision while poll() and epoll() can only handle one millisecond precision. Although this is not concerned on a desktop or server system but it may be necessary on a real time embedded platform.