Group:

Ahsan Naveed (anaveed)
Amandeep Sindhar (asindhar)

Method	Ease of Programming	Clock Sync. Accuracy	Adaptability
Socket: UDP	**5**	**6**	**5**
Socket: TCP	**4**	**5**	**6**
ZeroMQ: ReqRep	**1**	**3**	**3**
ZeroMQ: PubSub	**2**	**4**	**2**
RPC or RMI	**6**	**7**	**7**
REST	**3**	**1**	**1**
SOAP	**7**	**2**	**4**

Assignment 03 – Part 03:

Note: All the rankings below have been assessed in the context of our clock synchronization scheme for a different context e.g. chat application rankings might be different.

Sockets:
A socket is a communication end point to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read. A socket forms an abstraction over the actual port that is used by the local operating system for a specific transport protocol. The standard socket-based approach toward transient messaging is very basic and as such, rather brittle: a mistake is easily made. Furthermore, sockets essentially support only TCP or UDP, meaning that any extra facility for messaging needs to be implemented separately by an application programmer.

UDP:
UDP is a simple transport-layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is further encapsulated in an IP datagram, which is sent to the destination.

Ease of Programming (Rank 5)
Reasoning: From a developer's perspective the amount of setup involved for sending a datagram is not appreciated.

1. For sending a packet via UDP, we need a message to send, its length, IP address of destination, port of destination
2. Once we have all these 4 things, we can create the socket object for carrying the data packets b/w client and server
3. We invoke **send/receive** call for actually sending/receiving packets on both client and server sides

4. The extract the data from the received packet using **getData** method

Clock Synchronization Accuracy (Rank 6)
Reasoning: Marshaling and un-marshaling of UDP datagram packets might introduce some unwanted inaccuracies or packets could be lost altogether.

Adaptability (Rank 5)
Reasoning: Compared to TCP there is less teardown. But special attention has been paid to standardizing the socket interface to allow programmers to make use of its entire suite of (messaging) protocols through a simple set of operations. Also, standard interfaces make it easier to port an application to a different machine.

TCP:
TCP is the more sophisticated of the two protocols, providing reliable delivery. First, TCP ensures that the receiving computer is ready to accept data. It uses a three-packet handshake in which both the sender and receiver agree that they are ready to communicate. Second, TCP makes sure that data gets to its destination.

Ease of Programming (Rank 4)
Reasoning: Not much setup information is needed when establishing a connection.

1. Setup TCP server which unlike UDP does not need to know the client
2. Server accepts incoming client connection request using **accept** method
3. Client establishes connection by providing Server IP address and port number

Clock Synchronization Accuracy (Rank 5)
Reasoning: Three-way hand-shake might introduce some undesired delays.

Adaptability (Rank 6)
Reasoning: Significant teardown is involved i.e. breaking and establishing connections is a lot of work.

ZeroMQ:
ZeroMQ establishes a higher level of abstraction in socket-based communication by pairing sockets: Actual message transmission generally takes place over TCP connections, and like TCP, all communication is essentially connection-oriented. However, setting up, and maintaining connections is kept mostly under the hood: an application programmer need not bother with those issues. ZeroMQ sockets support many-to-one communication instead of just one-to-one communication as is the case with standard sockets. ZeroMQ sockets also support one-to-many communication, i.e., multicasting. Unlike TCP, ZeroMQ uses messages instead of bytestreams as its underlying model of communication.

ReqRep:

The request-reply pattern is used in traditional client-server communication, like the ones normally used for remote procedure calls.

Ease of Programming (Rank 1)
Reasoning: The request-reply pattern simplifies matters for developers by avoiding the need to call the **listen** operation, as well as the **accept** operation. Moreover, when a server receives a message, a subsequent call to **send** is automatically targeted toward the original sender. Likewise, when a client calls the **recv** operation (for receiving a message) after having sent a message, ZeroMQ assumes the client is waiting for a response from the original recipient.

Clock Synchronization Accuracy (Rank 3)
Reasoning: Since ZMQ uses TCP under the hood similar reasons for unwanted inaccuracies apply here.

Adaptability (Rank 3)
Reasoning: ZMQ is cross-platform (Linux, Unix, Windows etc.) and supports multiple programming languages. Reconfiguring the server would be a significant amount of work.

PubSub:
In the case of a publish-subscribe pattern, clients subscribe to specific messages that are published by servers. In its simplest form, this pattern establishes multicasting messages from a server to several clients.

Ease of Programming (Rank 2)
Reasoning: The server is assumed to use a socket of type **PUB**, while each client must use **SUB** type sockets. Each client socket is connected to the socket of the server. By default, a client subscribes to no specific message. This means that as long as no explicit subscription is provided, a client will not receive a message published by the server.

Clock Synchronization Accuracy (Rank 4)
Reasoning: Similar to ReqRep.

Adaptability (Rank 2)
Reasoning: Similar to ReqRep.

RPC or RMI:
An RPC is essentially a standardized way to let a client on machine **A** make an ordinary call to a procedure that is implemented on another machine **B**. While the basic idea sounds simple and elegant, subtle problems exist.

Ease of Programming (Rank 6)
Reasoning: RPC is fully embedded is Java, where it is referred to as a remote method invocation (RMI). In essence, a client being executed by its own (Java) virtual machine can

invoke a method of an object managed by another virtual machine. By simply reading an application's source code, it may be hard or even impossible to see whether a method invocation is to a local or to a remote object.

Clock Synchronization Accuracy (Rank 7)
Reasoning: To start with, because the calling and called procedures run on different machines, they execute in different address spaces, which causes complications. Parameters and results also have to be passed, which can be complicated, especially if the machines are not identical. Finally, either or both machines can crash and each of the possible failures causes different problems. Still, most of these can be dealt with, and RPC is a widely-used technique that underlies many distributed systems.

Adaptability (Rank 7)
Reasoning: RPC has language-based support e.g. RMI in Java. Need to make sure we spin up a JVM for each new client.

REST:
A resource-based architecture.

1. Resources are identified through a single naming scheme
2. All services offer the same interface, consisting of at most four operations
3. Messages sent to or from a service are fully self-described
4. After executing an operation at a service, that component forgets everything about the caller

Ease of Programming (Rank 3)
Reasoning: From developer's perspective only one endpoint is needed, and data format can be in either XML, HTML or JSON.

Error checking needs to be deferred until runtime and with generic operations, changes are much easier to accommodate, as they would generally involve changing the layout of strings that encode what is actually required.

Clock Synchronization Accuracy (Rank 1)
Reasoning: Due to error checking on runtime plus unpredictable network traffic.

Adaptability (Rank 1)
Reasoning: Platform and language agnostic.

SOAP:
An object-based and service-oriented architecture.

Ease of Programming (Rank 7)
Reasoning: Only allows for XML data exchange format.

Many syntactical errors can often already be caught during compile time. Also, one can argue that specifying the semantics of an operation is much easier with specific interfaces than with ones that offer only generic operations.

Clock Synchronization Accuracy (Rank 2)
Reasoning: Unpredictable network traffic.

Adaptability (Rank 4)
Reasoning: Platform and language agnostic. Each new system joining this architecture will incur administrative delays.

References:

https://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html
https://en.wikipedia.org/wiki/SOAP
http://www.hpcs.cs.tsukuba.ac.jp/~tatebe/lecture/h23/dsys/dsd-tutorial.html