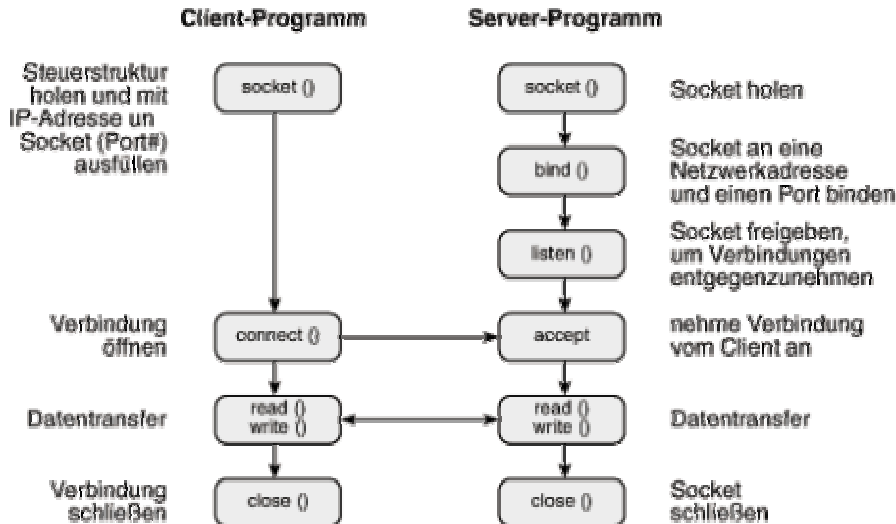


3. Anmerkungen zu Sockets

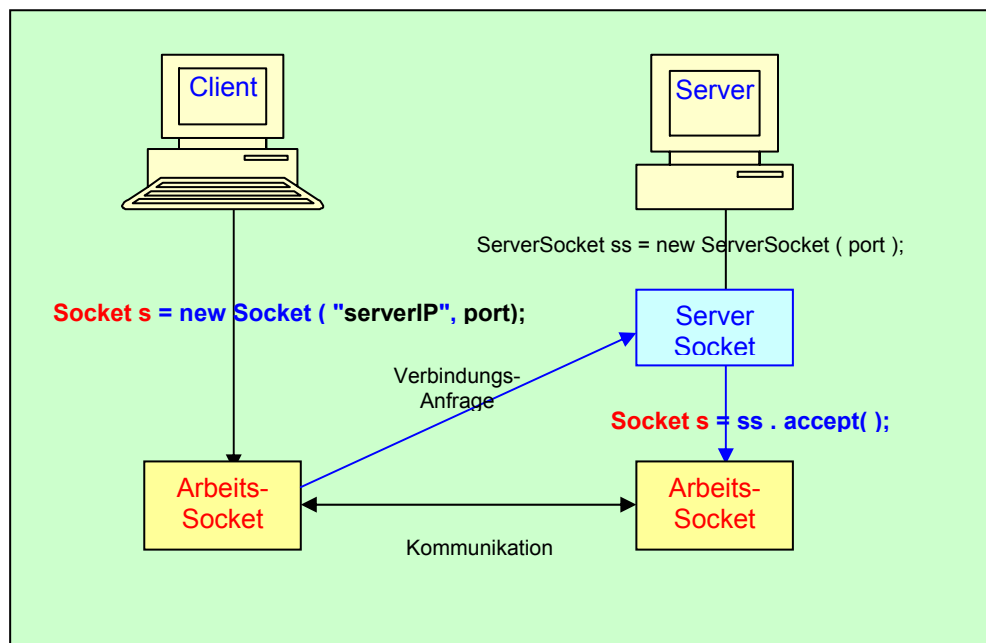
Mit dem Socket-API (z. B. Winsock unter Windows oder Berkely Sockets unter UNIX) ist es möglich, die verschiedensten Anwendungsfälle innerhalb Ihrer Software zu realisieren. Das API bietet die gesamte Funktionalität zum Transport der Daten über das Netzwerk.



Aufgrund der unterschiedlichen Rollenverteilung sind Client- und Serverprogramm unterschiedlich strukturiert – es gibt zwei verschiedene Arten von Sockets, die beim Eröffnen der Netzwerkverbindung eine Rolle spielen.

Zwei Arten von Sockets

- Der Client hat nur einen **Arbeits-Socket**, der sowohl die Verbindung anfordert als auch für den anschließenden Datenaustausch sorgt.
- Der Server benötigt hingegen zwei verschiedene Sockets,
 - einen **Server-Socket**, der auf eingehende Verbindungsanfragen von Clients horcht
 - und einen **Arbeits-Socket**, der für den Datenaustausch zwischen den beteiligten Rechnern sorgt.

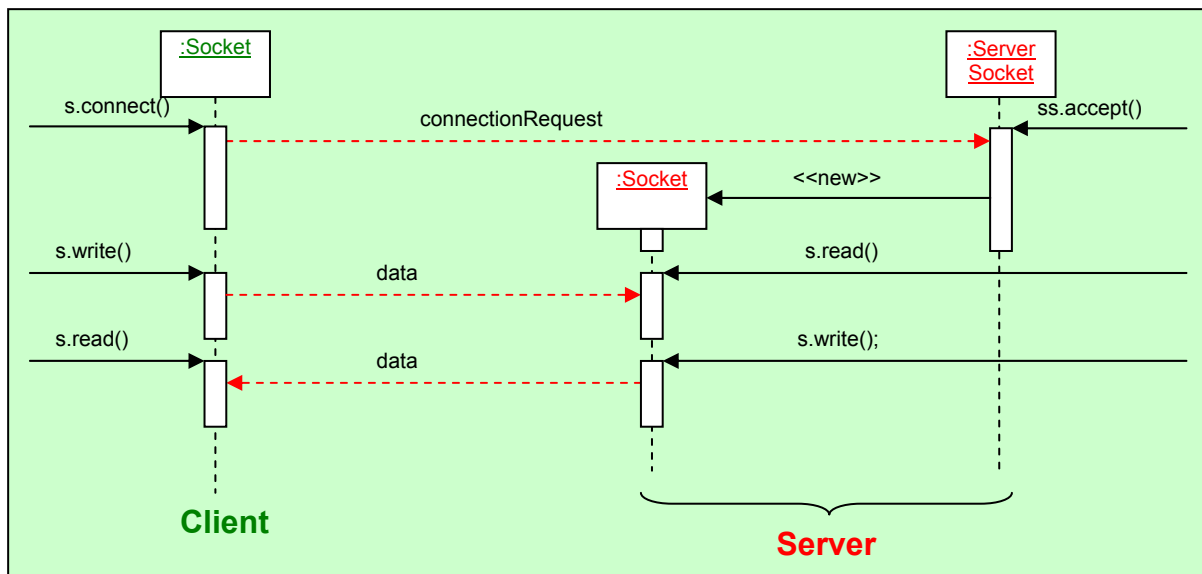


Diese beiden Socketarten haben ganz verschiedene Aufgaben. Der **Server-Socket** ist nur zur Annahme eingehender Verbindungsanfragen zuständig. Er ist nicht aktiv an der Netzwerk-Kommunikation beteiligt und er hat keine Ein- und Ausgabeströme, die von einer Anwendung sinnvoll genutzt werden können. Der Server-Socket wartet (blockierend) in der `accept()`-Methode auf eingehende Client-Verbindungen.

Beim Eintreffen einer Clientanfrage erzeugt der Server (in der zurückkehrenden `accept()`-Methode) einen **Arbeitssocket**, der von nun an für die gesamte weitere Kommunikation mit dem Client zuständig ist.

Verlauf eines Verbindungsaufbaus mit Sockets:

1. Das **Server**-Programm erstellt einen Server-Socket an einem bestimmten Port. Der Server-Socket ist passiv und wartet nach Aufruf seiner Methode `accept()` auf eintreffende Client-Anfragen.
2. Das **Client**-Programm erstellt einen Arbeits-Socket und sendet eine Verbindungsanfrage an den Server.
3. Wenn eine Client-Anfrage eintrifft, dann erzeugt der Server einen Arbeits-Socket, der die weitere Kommunikation mit dem Client übernimmt. Wenn der Server mehrere Client-Anfragen gleichzeitig bearbeitet soll, dann müssen **Threads** erzeugt werden.
4. Die **Kommunikation** erfolgt über die `read()` und `write()`-Methoden der beiden Arbeitssockets.



Sequenzdiagramm Socket-Verbindung