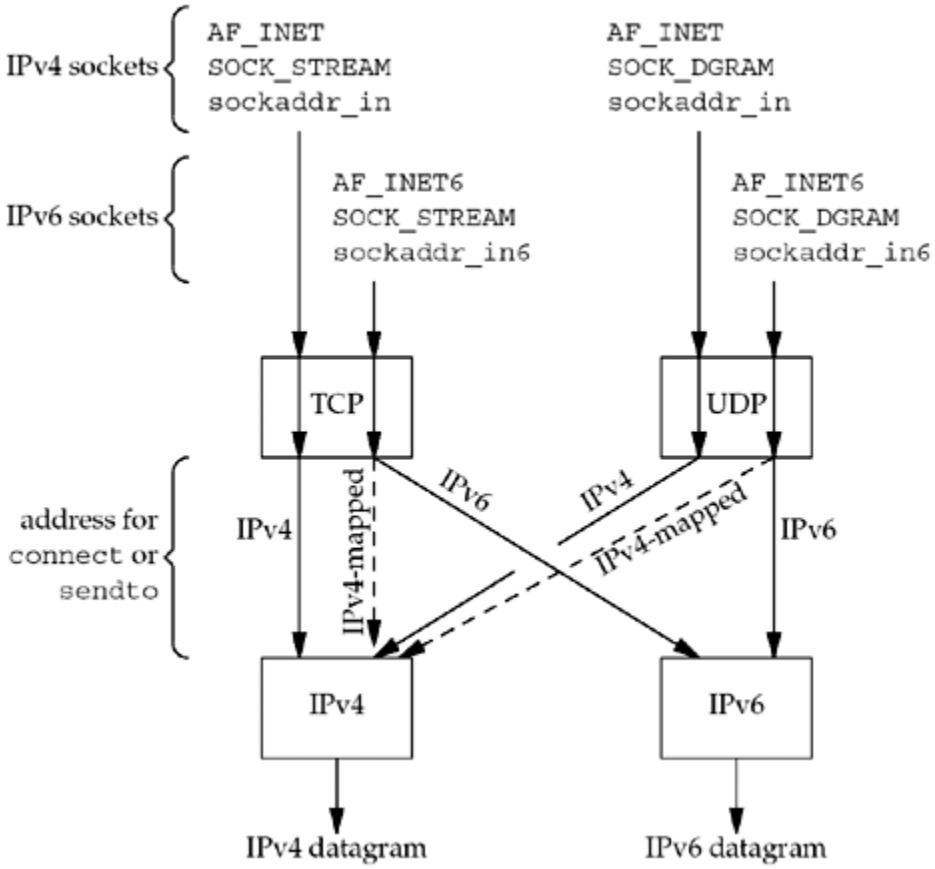
## 12.3 IPv6 Client, IPv4 Server

We now swap the protocols used by the client and server from the example in the previous section. First consider an IPv6 TCP client running on a dual-stack host.

1. An IPv4 server starts on an IPv4-only host and creates an IPv4 listening socket.
2. The IPv6 client starts and calls `getaddrinfo` asking for only IPv6 addresses (it requests the `AF_INET6` address family and sets the `AI_V4MAPPED` flag in its *hints* structure). Since the IPv4-only server host has only A records, we see from Figure 11.8 that an IPV4-mapped IPv6 address is returned to the client.
3. The IPv6 client calls `connect` with the IPv4-mapped IPv6 address in the IPv6 socket address structure. The kernel detects the mapped address and automatically sends an IPv4 SYN to the server.
4. The server responds with an IPv4 SYN/ACK, and the connection is established using IPv4 datagrams.

We can summarize this scenario in Figure 12.4.

**Figure 12.4. Processing of client requests, depending on address type and socket type.**




- If an IPv4 TCP client calls `connect` specifying an IPv4 address, or if an IPv4 UDP client calls `sendto` specifying an IPv4 address, nothing special is done. These are the two arrows labeled "IPv4" in the figure.
- If an IPv6 TCP client calls `connect` specifying an IPv6 address, or if an IPv6 UDP client calls `sendto` specifying an IPv6 address, nothing special is done. These are the two arrows labeled "IPv6" in the figure.
- If an IPv6 TCP client specifies an IPv4-mapped IPv6 address to `connect` or if an IPv6 UDP client specifies an IPv4-mapped IPv6 address to `sendto`, the kernel detects the mapped address and causes an IPv4 datagram to be sent instead of an IPv6 datagram. These are the two dashed arrows in the figure.
- An IPv4 client cannot specify an IPv6 address to either `connect` or `sendto` because a 16-byte IPv6 address does not fit in the 4-byte `in_addr` structure within the IPv4 `sockaddr_in` structure. Therefore, there are no arrows from the IPv4 sockets to the IPv6 protocol box in the figure.

In the previous section (an IPv4 datagram arriving for an IPv6 server socket), the conversion of the received address to the IPv4-mapped IPv6 address is done by the kernel and returned transparently to the application by `accept` or `recvfrom`. In this section (an IPv4 datagram needing to be sent on an IPv6 socket), the conversion of the IPv4 address to the IPv4-mapped IPv6 address is done by the resolver according to the rules in Figure 11.8, and the mapped address is then passed transparently by the application to `connect` or `sendto`.

## Summary of Interoperability

Figure 12.5 summarizes this section and the previous section, plus the combinations of clients and servers.

**Figure 12.5. Summary of interoperability between IPv4 and IPv6 clients and servers.**

| | IPv4 server IPv4-only host (A only) | IPv6 server IPv6-only host (AAAA only) | IPv4 server dual-stack host (A and AAAA) | IPv6 server dual-stack host (A and AAAA) |
|---|---|---|---|---|
| IPv4 client, IPv4-only host | IPv4 | (no) | IPv4 | IPv4 |
| IPv6 client, IPv6-only host | (no) | IPv6 | (no) | IPv6 |
| IPv4 client, dual-stack host | IPv4 | (no) | IPv4 | IPv4 |
| IPv6 client, dual-stack host | IPv4 | IPv6 | (no*) | IPv6 |

Each box contains "IPv4" or "IPv6" if the combination is okay, indicating which protocol is used, or "(no)" if the combination is invalid. The third column on the final row is marked with an asterisk because interoperability depends on the address chosen by the client. Choosing the AAAA record and sending an IPv6 datagram will not work. But choosing the A record, which is returned to the client as an IPv4-mapped IPv6 address, causes an IPv4 datagram to be sent, which will work. By looping through all adresses that `getaddrinfo` returns, as shown in Figure 11.4, we can ensure that we will (perhaps after some timeouts) try the IPv4-mapped IPv6 address.

Although it appears that five entries in the table will not interoperate, in the real world for the foreseeable future, most implementations of IPv6 will be on dual-stack hosts and will not be IPv6-only implementations. If we therefore remove the second row and the second column, all of the "(no)" entries disappear and the only problem is the entry with the asterisk.