



Step in packet sending process

1. The Application has previously created a socket (here, a UDPSocket) It calls `Socket::Send()`. Either real data or dummy data is passed at the API.
2. `Socket::Send` forwards to `UdpSocketImpl::DoSend()` and later to `UdpSocketImpl::DoSendTo()`. These functions set the proper source and destination addresses, handle socket calls, such as `bind()` and `connect()` and then the `UdpL4Protocol::Send()` function is called. As in a real implementation, the socket must query the `Ipv4` routing system to find the right source address to match the destination address.
3. **UdpL4Protocol** is where the socket-independent protocol logic for UDP is implemented. The `Send()` method adds the UDP header, initializes the checksum, and sends the packet to the `Ipv4` layer. The packet is not sent directly to the `Ipv4` layer but via a callback called `m_downTarget`. In this example, the `downtarget` is `Ipv4L3Protocol`, but it could be some other shim layer in general.
4. **Ipv4L3Protocol** adds the IP header and sends the packet to an appropriate `Ipv4Interface` instance, based on the route that was passed down from the UDP layer. In this example, the device is one that supports Arp.
5. **Ipv4Interface** looks up the MAC address if Arp is supported on this NetDevice technology, and if there is a cache hit, it sends the packet to the NetDevice, or else it first initiates an Arp request and waits for a reply.