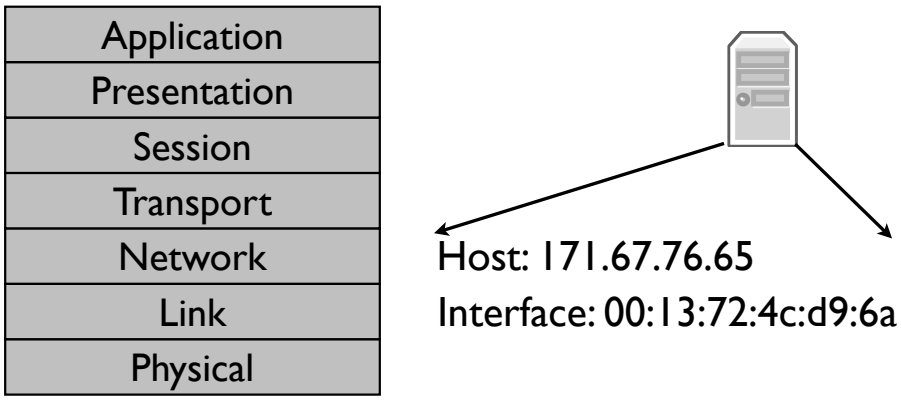
Address Resolution Protocol (ARP)

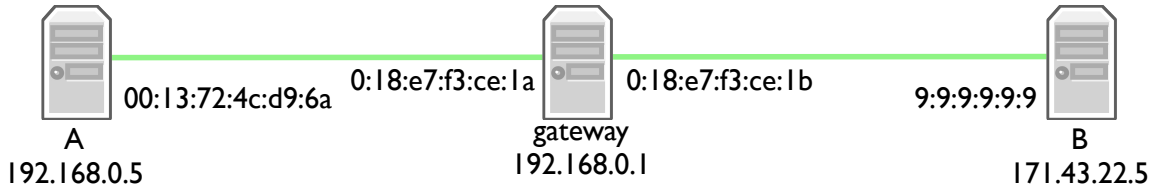
The Address Resolution Protocol, or ARP, is the mechanism by which the network layer can discover the link address associated with a network address it's directly connected to. Put another way, it's how a device gets an answer to the question: "I have an IP packet whose next hop is this address -- what link address should I send it to?"

Layers of Addresses



ARP is needed because each protocol layer has its own names and addresses. An IP address is a network-level address. It describes a host, a unique destination at the network layer. A link address, in contrast, describes a particular network card, a unique device that sends and receives link layer frames. Ethernet, for example, has 48 bit addresses. Whenever you buy an Ethernet card, it's been preconfigured with a unique Ethernet address. So an IP address says "this host", while an Ethernet address says "this Ethernet card."

Addressing Problem

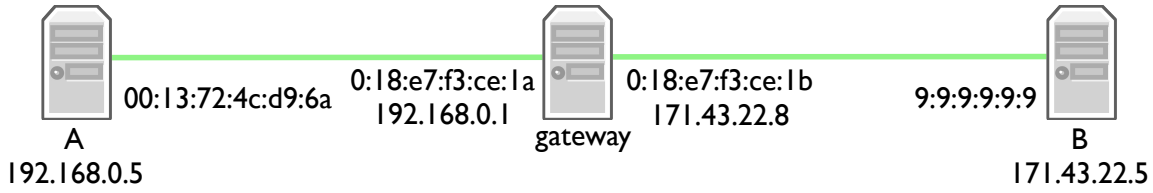


48-bit Ethernet addresses are usually written as a colon delimited set of 6 octets written in hexidecimal, such as 0:13:72:4c:d9:6a as in the source, or 9:9:9:9:9:9 as in the destination.

One thing that can be confusing is that while these link layer and network layer addresses are completely decoupled with respect to the protocol layers, in terms of assignment and management they might not be. For example, it's very common for a single host to have multiple IP addresses -- one for each of its interfaces. It needs to because of the concept of a netmask. For example, look at this hypothetical setup. The gateway, in the middle, has a single IP address: 192.168.0.1. It has two network cards, one connecting it to the destination 171.43.22.5, one connecting it to the source, 192.168.0.5.

The address 192.168.0.1 can really only be in one of these networks, the source network. The netmask needed for 192.168.0.1 to be in the same network as 171.43.22.5 is 128.0.0.0, or just one bit of netmask! But it can't be that all IP addresses whose first bit is 1 are in the same network as 171.43.22.5 -- 192.168.0.5, for example, needs to be reached through the gateway.

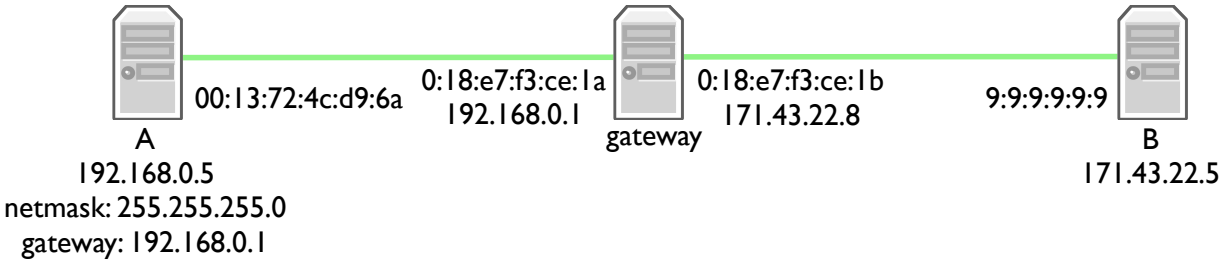
Example Addressing



So instead we often see setups like this, where the gateway or router has multiple interfaces, each with their own link layer address to identify the card, and also each with their own network layer address to identify the host within the network that card is part of. For the gateway, the left interface has IP address 192.168.0.1, while the right interface has IP address 171.43.22.8.

The fact that link layer and network layer addresses are decoupled logically but coupled in practice is in some ways a historical artifact. When the Internet started, there were many link layers, and it wanted to be able to run on top of all of them. These link layers weren't going to suddenly start using IP addresses instead of their own addressing scheme. Furthermore, there turns out to be a bunch of situations where having a separate link layer address is very valuable. For example, when I register a computer with Stanford's network, I register its link layer address -- the address of the network card.

Encapsulation

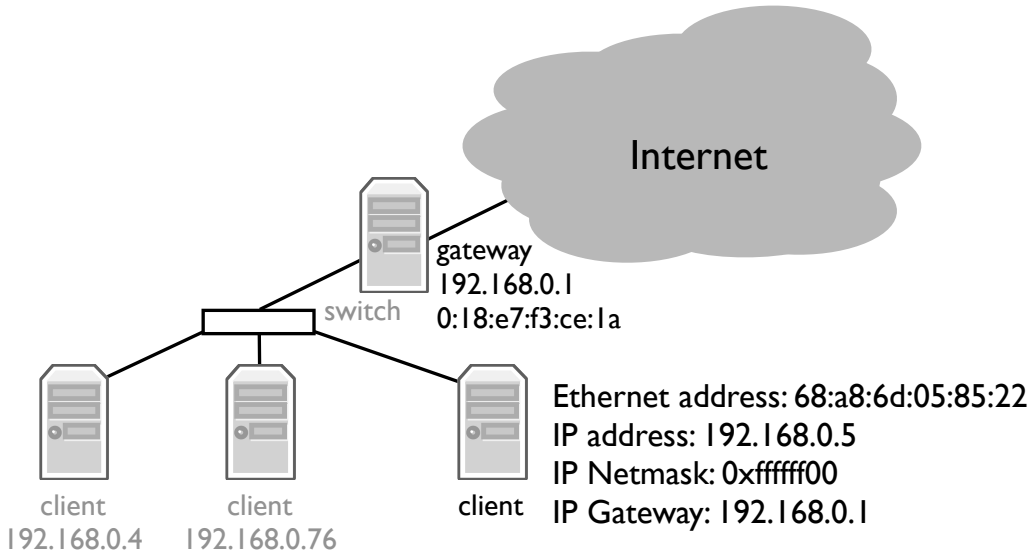


So what does this mean in practice? Let's say node A, on the left, wants to send a packet to node B, on the right. It's going to generate an IP packet with source address 192.168.0.5 and destination address 171.43.22.5.

Node A checks whether the destination address is in the same network. The netmask tells it that the destination address is in a different network: 255.255.255.0. This means node A needs to send the packet through the gateway, or 192.168.0.1. To do this, it sends a packet whose network-layer destination is 171.43.22.5 but whose link-layer destination is the link layer address of the gateway. So the packet has a network layer destination 171.43.22.5 and a link layer destination 0:18:e7:f3:ce:1a. The network layer source is 192.168.0.5 and the link layer source is 0:13:72:4c:d9:6a.

So we have an IP packet from A to B, encapsulated inside a link layer frame from A to the left gateway interface. When the packet reaches the gateway, the gateway looks up the next hop, decides it's node B, and puts the IP packet inside a link layer frame to B. So this second IP packet from A to B is inside a link layer from from the right gateway interface to B.

Example Problem



So here we get to the problem ARP solves. My client knows that it needs to send a packet through the gateway that has IP address 192.168.0.1. To do so, however, it needs to have the link layer address associated with 192.168.0.1. How does it get this address? We somehow need to be able to map a layer 3, network layer, address, to its corresponding layer 2, link layer, address. We do this with a protocol called ARP, or the Address Resolution Protocol.

Address Resolution Protocol

- Generates mappings between layer 2 and layer 3 addresses
  - Nodes cache mappings, cache entries expire
- Simple request-reply protocol
  - "Who has network address X?"
  - "I have network address X."
- Request sent to link layer broadcast address
- Reply sent to requesting address (not broadcast)
- Packet format includes redundant data
  - Request has sufficient information to generate a mapping
  - Makes debugging much simpler
- No "sharing" of state: bad state will die eventually

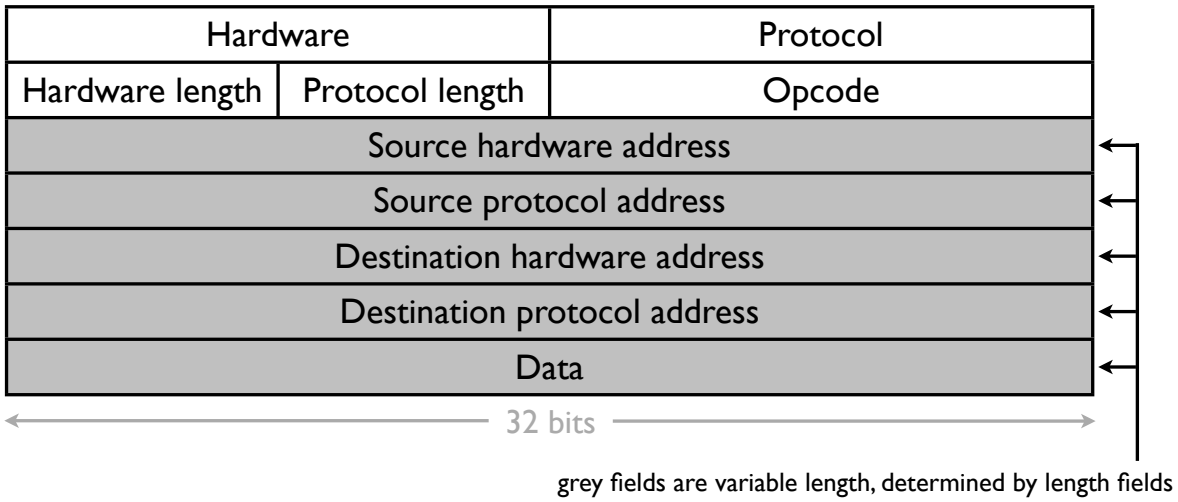
ARP is a simple request-reply protocol. Every node keeps a cache of mappings from IP addresses on its network to link layer addresses. If a node needs to send a packet to an IP address it doesn't have a mapping for, it sends a request: "Who has network address X?" The node that has that network address responds, saying "I have network address X." The response includes the link layer address. On receiving the response, the requester can generate the mapping and send the packet.

So that every node hears the request, a node sends requests to a link layer broadcast address. Every node in the network will hear the packet.

Furthermore, ARP is structured so that it contains redundant data. The request contains the network and link layer address of the requestor. That way, when nodes hear a request (since it's broadcast), they can insert or refresh a mapping in their cache. Nodes *only* respond to requests for themselves. This means, assuming nobody is generating packets incorrectly, the only way you can generate a mapping for another node is in response to a packet that node sends. So if that node crashes or disconnects, its state will inevitably leave the network when all of the cached mappings expire. This makes debugging and troubleshooting ARP much easier.

So how long do these dynamically discovered mappings last? It depends on the device: some versions of Mac OSX, for example, keep them around for 20 minutes, while some Cisco devices use timeouts of 4 hours. The assumption is that these mappings do not change very frequently.

ARP Packet Format (RFC826)



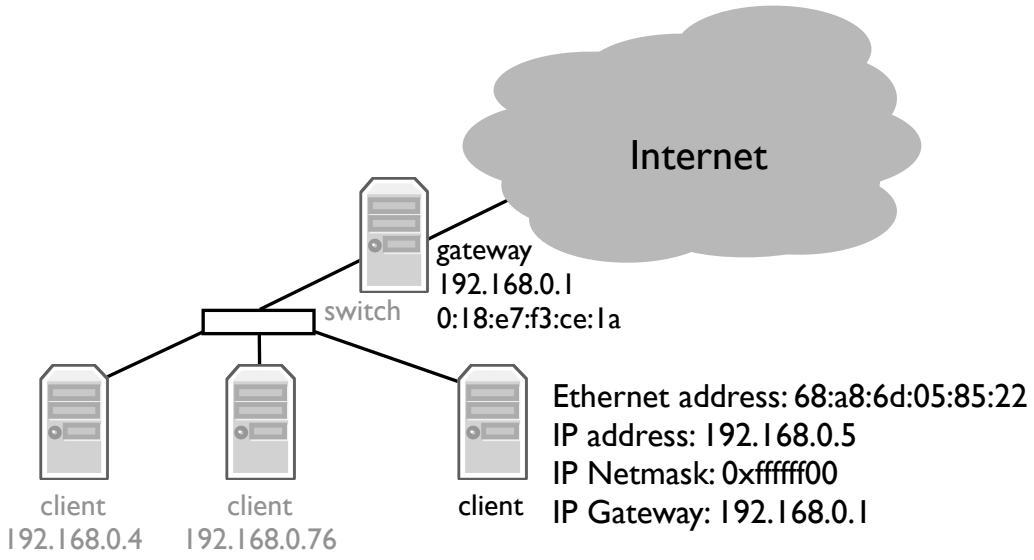
This is what an ARP packet actually looks like. It has 10 fields. The hardware field states what link layer this request or response is for. The protocol field states what network protocol this request or response is for. The length fields specify how many bytes long the link layer and network layer addresses are. The opcode specifies whether the packet is a request or response.

The four address fields are for requesting and specifying the mappings.

Remember, all of these fields are stored in network order, or big endian. So if I have an opcode of 15, it will be stored as 0x000f in the opcode field.

The full details of ARP are in IETF Request for Comments, RFC, 826. I'll just go over a simple request/response exchange.

ARP Request

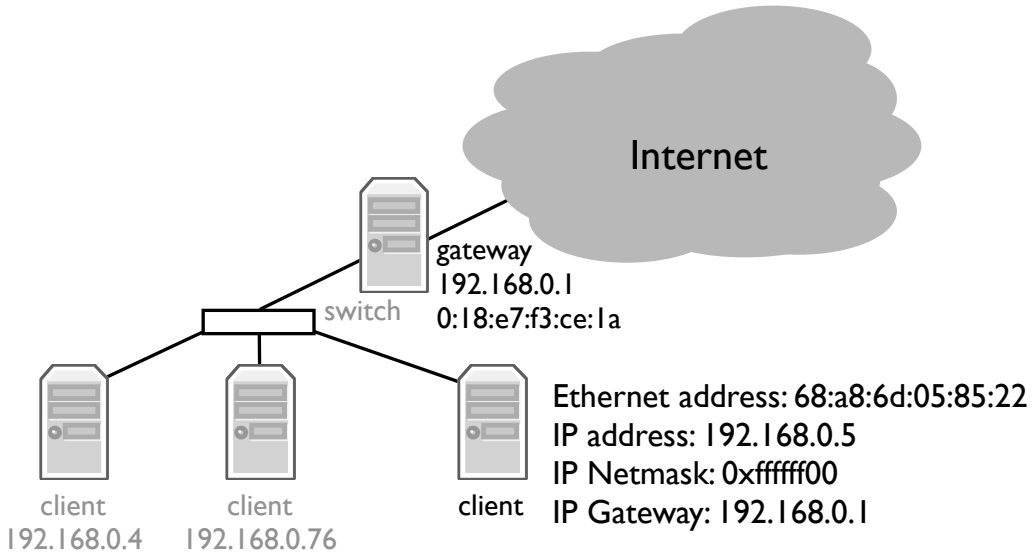


So let's say our client wants to send a packet to the broader Internet through its gateway. But it doesn't have the gateway's Ethernet address.

The client will generate an ARP request packet whose link layer source address is its address: 68:a8:6d:05:82:22. The destination link layer address is the broadcast address: ff:ff:ff:ff:ff:ff, all 1s.

The ARP request will specify that the hardware is Ethernet, which is value 1, the protocol is IP, which is value 0x0800, the hardware address length is 6, and the protocol length is 4. The opcode will be request, whose value is 1. The ARP source hardware field will be the requester's Ethernet address, 68:a8:6d:05:85:22. The source protocol field is the requester's IP address: 192.168.0.5. The destination hardware address can be set to anything -- it is what the packet is trying to discover. The destination protocol address is the address the client is trying to find a mapping for: 192.168.0.1. The client sends this frame on the Ethernet. Every node in the network receives it and refreshes its mapping between its link address, 68:a8:6d:05:85:22, and its network address, 192.168.0.5, or inserts a mapping if it doesn't have one.

ARP Reply



The gateway sees that the request is for its IP address and so generates a reply.

Like the request, the ARP reply will specify that the hardware is Ethernet, which is value 1, the protocol is IP, which is value 0x0800, the hardware address length is 6, and the protocol length is 4. The opcode will be reply, whose value is 2.
The ARP source hardware field will be the replier's Ethernet address, 0:18:e7:f3:ce:1a. The source protocol field is the answer: 192.168.0.1. The destination hardware address is the source hardware address of the request: 68:a8:6d:05:85:22. The destination protocol address is the source protocol address of the request: 192.168.0.5.

It's an open question what link layer address you send the response to. The original ARP specification stated that the replier should send it to the requester's link layer address, so unicast. It's common today to broadcast it, however, as doing so can more aggressively replace cache entries if the mapping needs to change. Nodes can also send what are called gratuitous ARP packets, requesting non-existent mappings, in order to advertise themselves on a network.

So we've seen how, in order to route packets, one needs to be able to map network layer addresses to link layer addresses. The Address Resolution Protocol, or ARP, provides this service through a simple request-reply exchange. If a node needs to send a packet to or through an IP address whose link layer address it does not have, it can request the address through ARP.