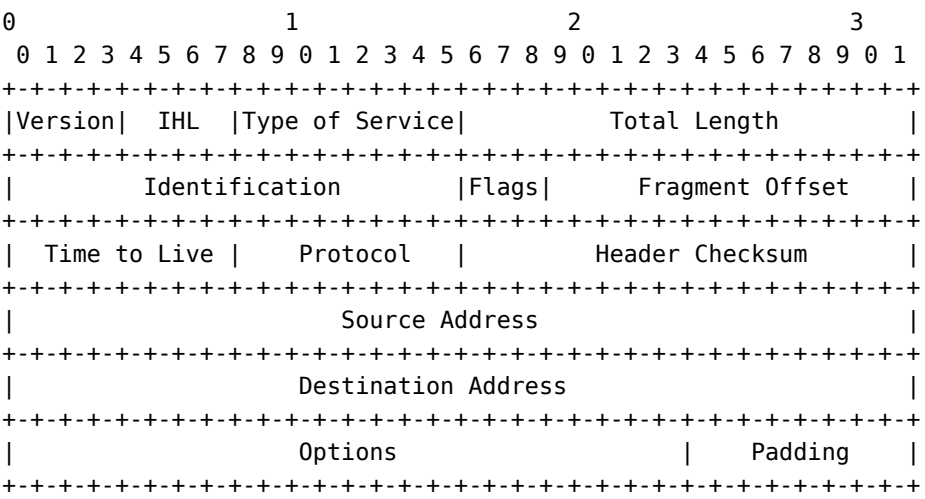
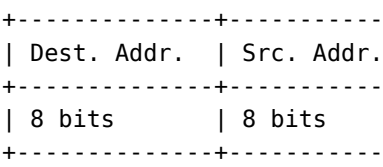
Oblig 1 IN4230 - kristfsk@uio.no: Design document

The MIP Protocol vs IPv4

To see the differences between the MIP protocol and ipv4 we should put their headers side by side.



ipv4 header [1]



MIP header [2]

The first difference we can see is the size. The MIP header is significantly smaller, with a size of only 32 bits = 4 bytes. The IPv4 header is at minimum 20 bytes long, and the IHL field can extend it up to 60 bytes [3]. There are some similar fields, like src address and dst address, TTL and length fields. IPv4 has a lot of fields which MIP do not, like the checksum field, which can be used to check for transmission errors. Performance wise MIP could be better in some situations. If you are in a situation where you only need to send small packets between a small set of computers and dont care about the extra features MIP could be more preferable because the smaller header. Since the MIP addresses are so small the number of possible nodes in the network is very limited (only $2^8 = 256$ machines!). IPv4 has a much longer address and therefore it is possible to have bigger networks $2^{32} = 4294967296$. The TTL of IPv4 is 8 bits, which makes it possible to do $2^8 = 256$ hops. TTL in MIP is 4 bits, which lets you do $2^4 = 16$ hops. When it comes to types, MIP is also more limited than IPv4, with only $2^3 = 8$ possible types. IPv4 has type of service, options and flags as well as the IHL. This makes IPv4 a more flexible protocol than MIP.

Why is MIP-ARP a special case?

When sending a MIP packet, we first need to know the mac address of the recipient and also what interface on our node to send the packet on. To find this out we send out a *broadcast* message. This is a message sent to every MAC address (where the broadcast address is FF:FF:FF:FF:FF:FF). To make sure that every node checks the MIP-ARP request, we specify the special address 0xFF as an address which every node has to accept. This way, nodes do not “throw away” the packet because it is not addressed to them.

A cleaner way to do this could be to have each node in the network broadcast out to its neighbors which MIP address it has. Another option would be to have the network administrator be the one in

charge of the mac to MIP mapping. Since MIP networks only can have 256, it is also totally doable for each node to have a complete list of all hosts.

Execution Flow chart

I wasnt totally sure what was meant by flow chart, but i think this is a good way of explaining:

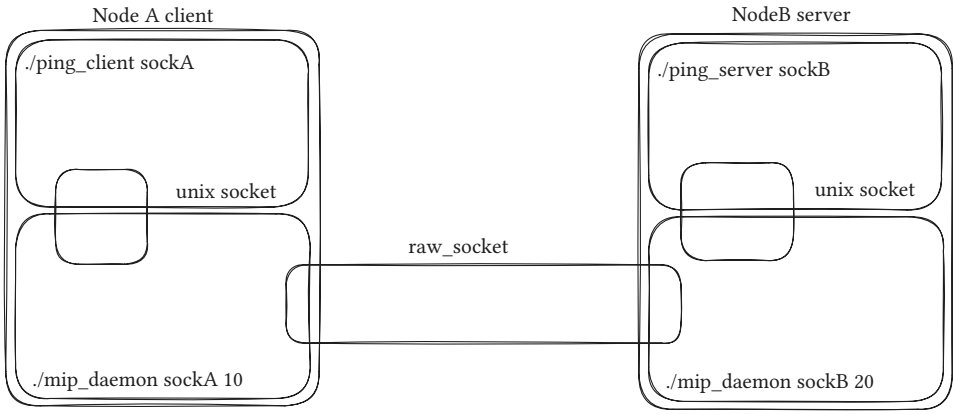


Figure 1: structure of MIP client and server

Case 1

The client initiates a ping request to MIP address 20 with an empty arp cache.

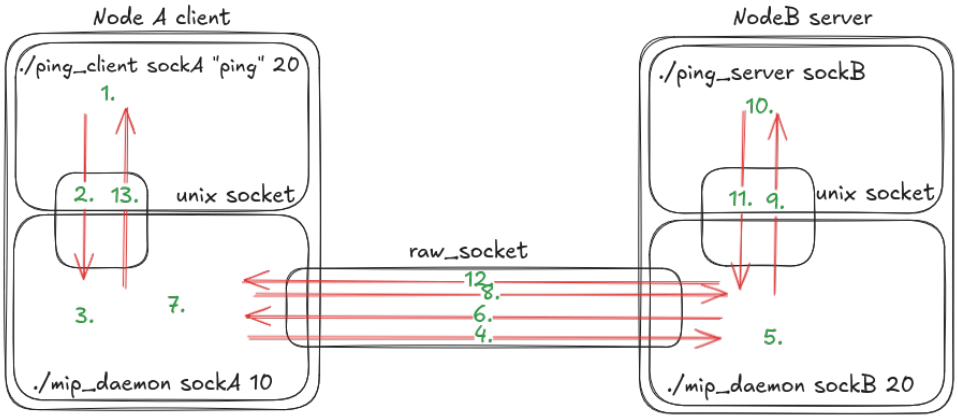


Figure 2: sending a MIP ping without arp cache

1. Ping client wants to send a ping message to MIP address 20.
2. The ping message and destination address is sent over the unix socket to the MIP daemon.
3. The MIP daemon checks its arp cache. cache miss.
4. Since it does not know where to send the MIP packet, it sends out a MIP-ARP request on all outgoing interfaces (in this case just one).
5. The MIP daemon receives the MIP-ARP request looking for the owner of MIP address 20. It saves the src of the MIP-ARP request in its own arp cache.
6. MIP-ARP response is sent back to node A
7. Node A now knows where to send the ping message for address 20
8. Ping message is sent over raw socket to node B
9. The daemon forwards the ping message over unix socket to the server.
10. The server copies the ping message, and changes out ping for pong. Sets the src of the ping to the dest of the pong (MIP address 10).
11. The server sends the pong message to the daemon over unix socket.

12. The MIP deamon checks if it knows where to send MIP address 10. arp request in step 5. and the pong is sent over raw socket to node A.
13. The MIP daemon forwards the packet over unix socket to the client, which can now print the message, and the time deifference from when the ping message was sent.

Case 2

In this case, node A already has address 20 in its arp cache.

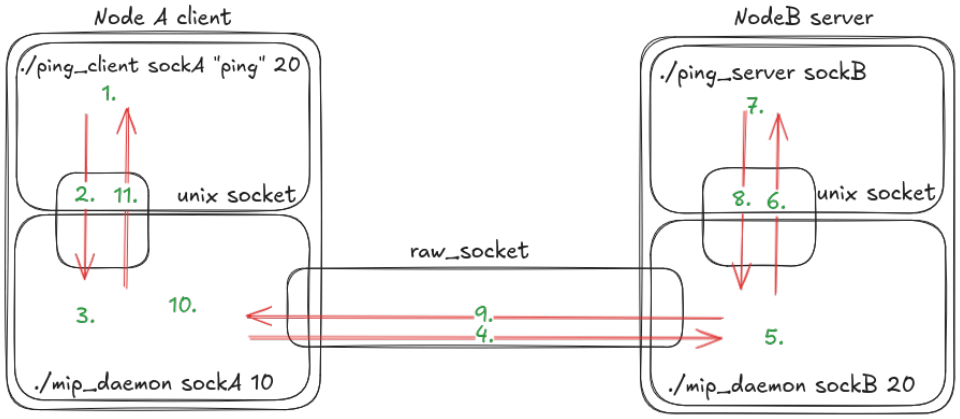


Figure 3: sending a MIP ping without arp cache

1. Ping client wants to send a ping message to MIP address 20.
2. The ping message and destination address is sent over the unix socket to the MIP daemon.
3. The MIP daemon checks its arp cache. cache hit.
4. Since it does not know where to send the MIP packet, it sends out a MIP-ARP request on all outgoing interfaces (in this case just one).
5. The ping is read from the raw socket (node B can use this packet to put address 10 in its arp cache).
6. The ping is forwarded to the server over unix socket.
7. The server copies the ping message, and changes out ping for pong. Sets the src of the ping to the dest of the pong (MIP address 10).
8. Pong message is sent over unix socket to MIP daemon.
9. Pong message is sent over raw socket to address 10.
10. Message is read from raw socket.
11. Message is forwarded to client over unix socket.

In this case we saved 1 RTT, since we did not

Case 3

In this case, node A tries to send a ping message to a address it is not connected with.

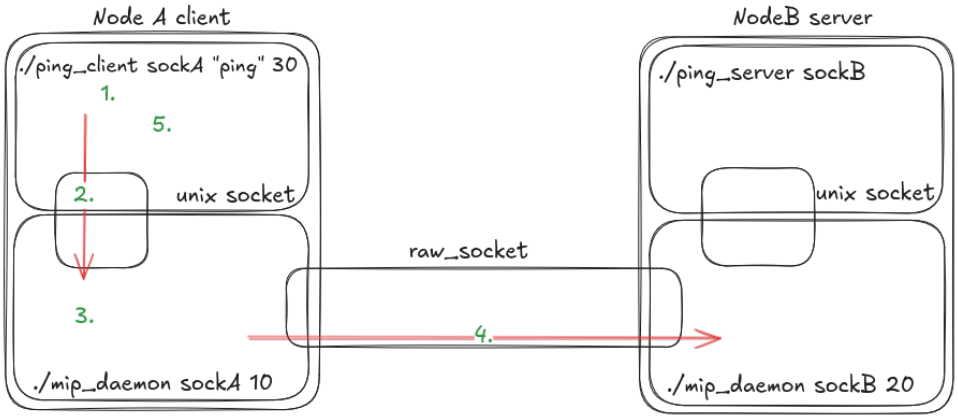


Figure 4: sending a MIP ping without arp cache

1. Ping client wants to send a ping message to MIP address 30.
2. The ping message and destination address is sent over the unix socket to the MIP daemon.
3. The MIP daemon checks its MIP cach for address 30. cache miss
4. Since it does not know where to send the MIP packet, it sends out a MIP-ARP request on all outgoing interfaces (in this case just one).
5. Client does not get a response from the MIP daemon and therefore times out.

Another flow chart

Another way of visualizing it is like this:

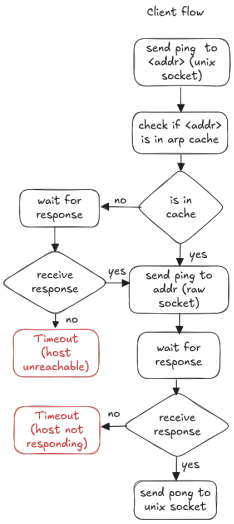


Figure 5: sending a MIP ping without arp cache

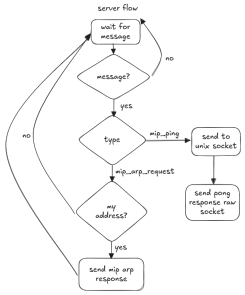


Figure 6: sending a MIP ping without arp cache

AI Disclaimer

While writing the programs for this assignment i have used AI as a “coding buddy”. The only AI tool I have used is the used the gpt.uio.no chat. I have not used any AI based coding tools such as copilot. My main approach to these kinds of tasks is asking gpt for suggestions of how to do something, like setting up an epoll_wait loop listening to two file descriptors. I always want to understand everything I write, and therefore never blindly copy code from the AI chat to my editor. All code generated by AI is always read by me, then I see what is applicable for my situation and manually. All code has understood and written by me into the text editor. When i have encountered bugs I have used AI chat to investigate the root cause by narrowing down as much as I can on my own, then giving the AI the minimal required context to my problem, then suggesting where the issue lies.

No AI was used while writing this document, except to generate bibtex references from links.

Bibliography

[1] J. Postel, “Internet Protocol.” Sep. 1981.

[2] M. Welzl, K. Hiorth, and K. Ciko, “IN3230/4230: Networks.” Aug. 2020.

[3] W. contributors, “IPv4.” 2023.