**Homework #2** Student ID**: 2018280070,** Name**: Peter Garamvoelgyi**

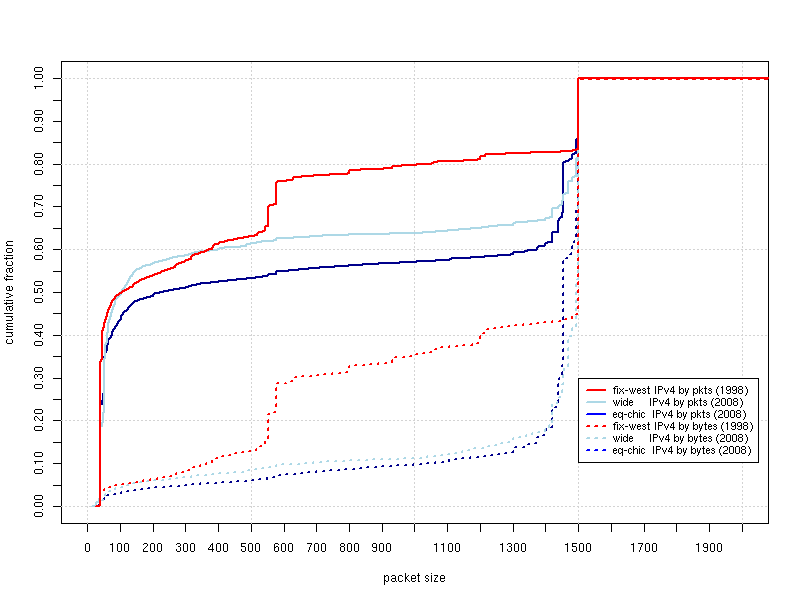
1. **Calculate/estimate the header overhead of the TCP/IP communication.**

A minimal TCP and IP header is 20 bytes each, i.e. 40B in total. The payload of an IP packet can range from 8B to 64 KB, but a more common practical upper bound is 1500B, due to Ethernet MTU. Using these, we can arrive at the following plot:

We can see that

1. for small packets, the 40B header can be a serious overhead, constituting up to 80%+ of the whole packet,
2. for large packets, the header constitutes about 3% or less of the overall packet.

For getting a more definite understanding of packet size distribution, we can take a look at Caida’s plots of IPv4 packet distribution in 1998 and 2008[[1]](#footnote-1) (measured in California and Chicago):

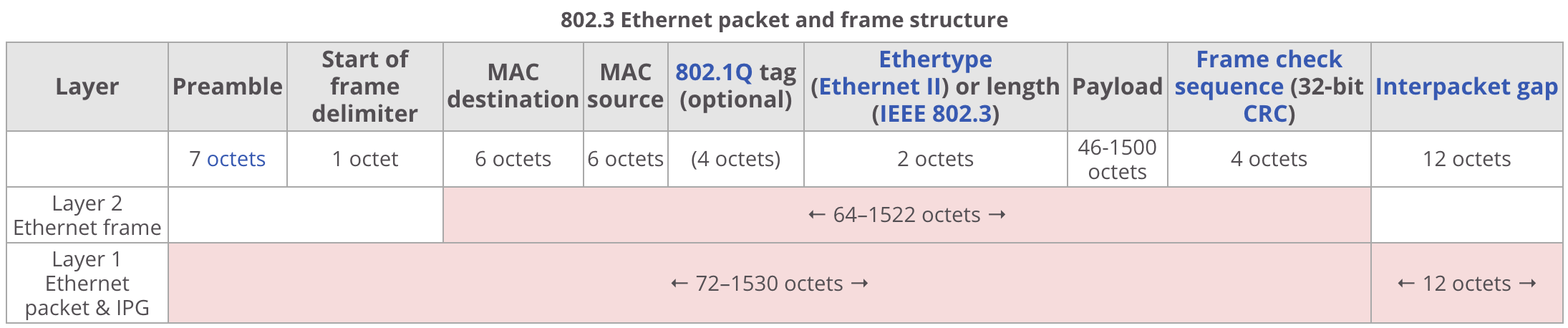


This plot suggests that 50% of the packets are smaller than 100B, while 20-40% are nearing 1500B.

The header overhead for a 100B packet is around 40%. If these stats can be generalized, that means that IP header overhead takes up a significant portion of network bandwidth.

1. **Some say the maximum packet size is 1518 bytes, others use other numbers, such as 1500 or even 1540, why?**

Let us take a look at the Ethernet packet/frame structure[[2]](#footnote-2):



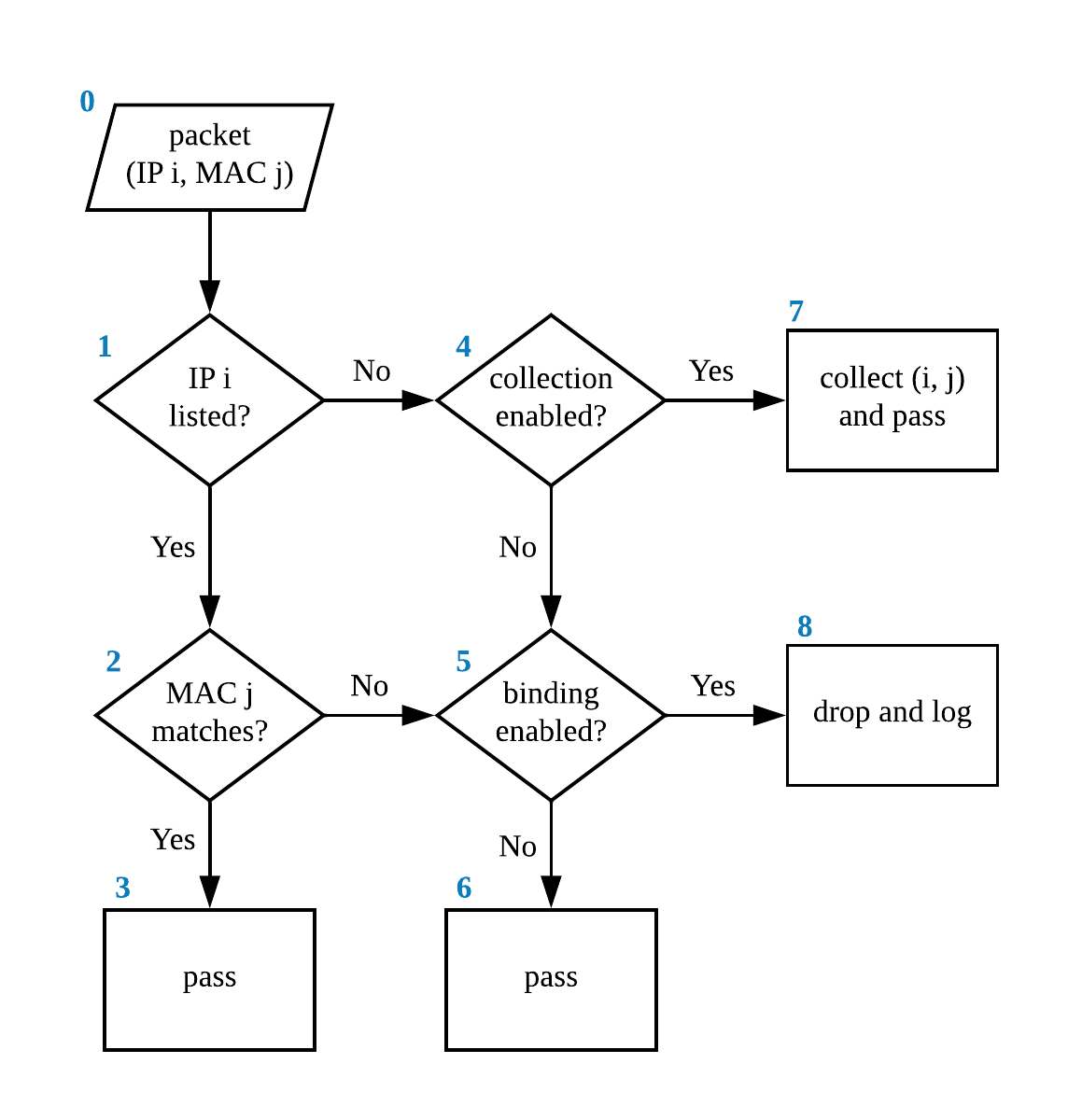
This shows that:

* The Ethernet **payload MTU is 1500 bytes**.
* Ethernet packets have a 14B header (destination MAC, source MAC, type/length) and a 4B CRC, **adding up to 1518B per packet**. (This becomes 1522B if we include the optional 802.1Q tag.)
* If we count the preamble, delimiter, and interpacket gap necessary for physical transmission, this adds an additional 20B, **adding up to 1538/1542B**.

To summarize: The reason for the inconsistencies in discussions of Ethernet maximum packet size is that different people mean different things when they talk about a packet. Some only consider the actual payload, some consider the headers and CRC, while others include the whole L1 frame.

1. **Design a firewall MAC-IP binding implementation with optional automatic MAC-IP pair collection and binding. A high level programming flowchart and description of the implementation will be fine. No actual coding is needed.**

**High level programming flowchart:**



**Discussion:**

In my design, a packet can pass in three different scenarios:

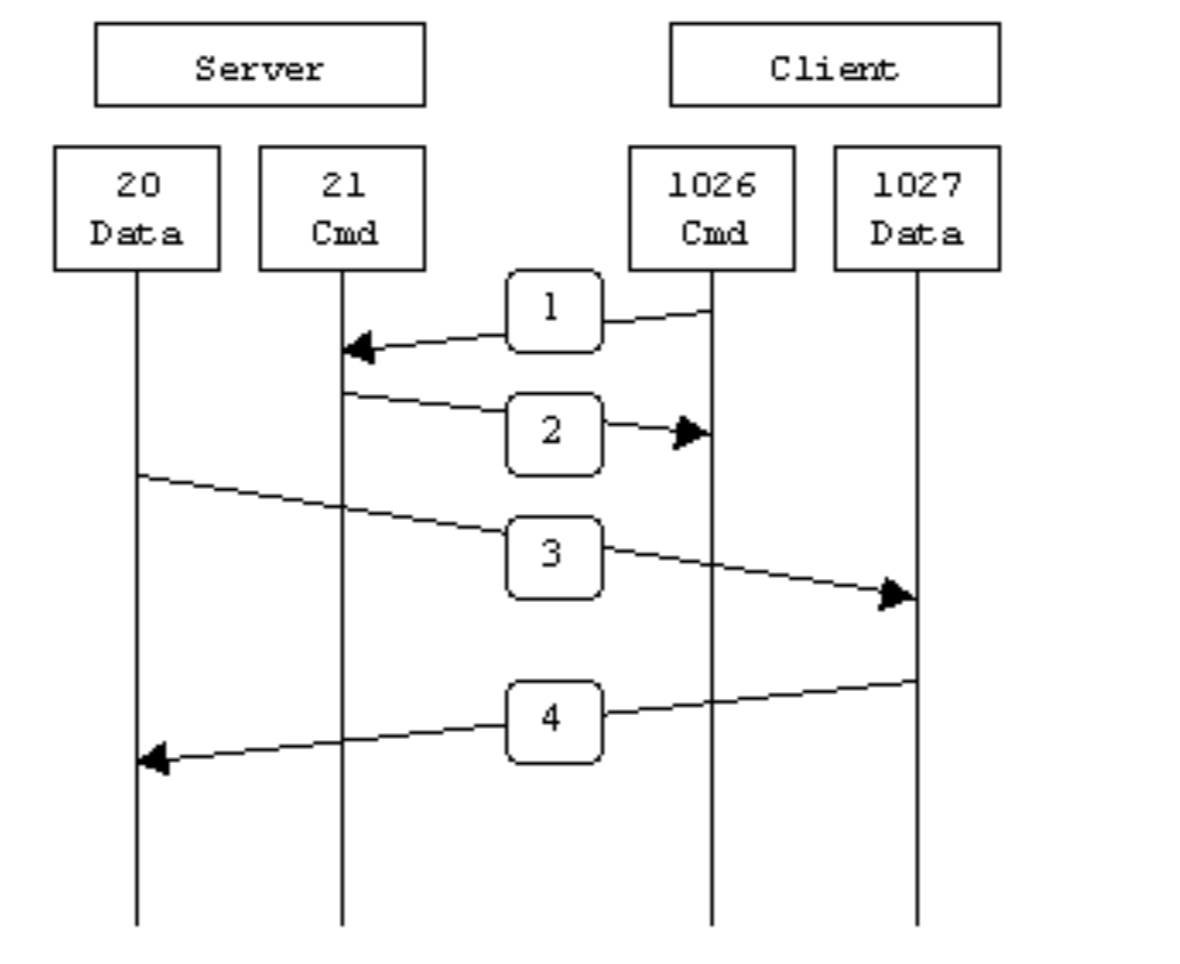
1. The IP is listed and the MAC matches the corresponding one in our database. (**0-1-2-3**)
2. The IP is not listed but we enabled collection. In this case, we simply store the new IP-MAC pair and pass. (**0-1-4-7**)
3. The IP is not listed but MAC-IP binding is not enabled. (**0-1-4-5-6**)

The two drop scenarios are as follows:

1. We enabled MAC-IP binding and we encountered a packet with listed IP but non-matching MAC. (**0-1-2-5-8**)
2. We enabled IP-MAC binding and we encountered a packet with an unlisted IP, and we also disabled collection. (**0-1-4-5-8**)

The implementation basically relies on two boolean switches/flags which can preferably set through a UI. Moreover, we need a way of storage and efficient lookup of IP-MAC pairs. (As such lookups are costly, our design could be improved by bypassing this when both collection and binding are disabled.) A typical use for this system would be to enable collection at the initialization phase, and then disable it and enable binding afterwards.

1. **(Bonus) When there is client side NAT and NPAT, how do you suggest we handle “dynamic protocols” such as Active FTP? Describe your method in details but no coding necessary.**

During active FTP, the client first connects to the server’s FTP command port (21) from any port N. Then, the client sends a command to the server specifying it’s data port (M). After this, the server proceeds to send the data from its FTP data port (20) to M.[[3]](#footnote-3)

Let us assume that the client is behind a NAT-enabled gateway. This way, the client’s command packet is translated this way:

[client IP] : N --> [gateway IP] : [random port K]

When the server sends replies to the client, it sends these to [gateway IP]:K and the gateway forwards these to [client IP]:N. However, when the server tries to send to [gateway IP]:M (M being specified in the command packet), the gateway does not know where to forward this packet, as there have been no corresponding outbound packets.

**Solutions:**

* The most robust solution is deep packet inspection. An FTP-aware NAT gateway, upon receiving an outbound packet to port 21, could inspect the payload and rewrite the proposed port to L, then add 2 PAT entries:

[client IP] : N --> [gateway IP] : K

[client IP] : M --> [gateway IP] : L

* While in theory M could be any port, typically M = N + 1. An FTP-aware NAT gateway can use this information. When the gateway receives an outbound packet to port 21, it saves two entries:

[client IP] : N --> [gateway IP] : K

[client IP] : (N+1) --> [gateway IP] : (N+1)

Of course, this requires that port N+1 on the gateway is not assigned yet.

* Alternatively, we can use the above solution with FTP-aware forwarding. If we receive an inbound packet from port 20 to port L, we check the entry for K=(L-1) and forward the packet to the corresponding host (port N+1). We only need one entry:

[client IP] : N --> [gateway IP] : K

1. source: <https://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml> [↑](#footnote-ref-1)
2. source: <https://en.wikipedia.org/wiki/Ethernet_frame#Structure> [↑](#footnote-ref-2)
3. image source: lecture slides [↑](#footnote-ref-3)