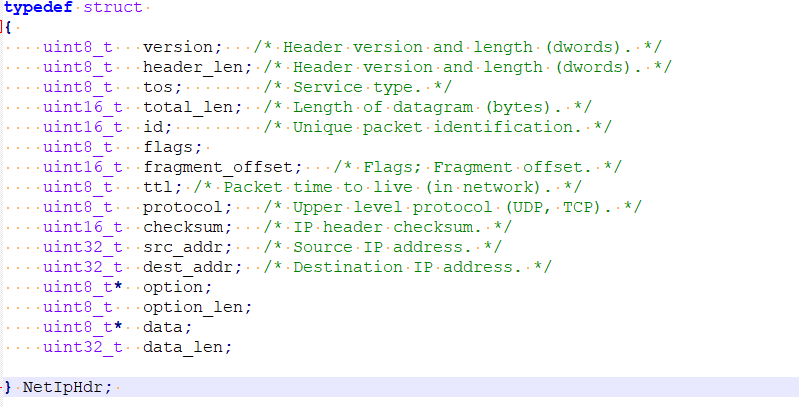
The document Explains about the Extract the Data received from the AIRPCAP and how to decode it.

1. **Below is the code snippet of the IPV4 header structure and each parameter description is below:**



The NetIpHdr represent an IP header and its associated fields.

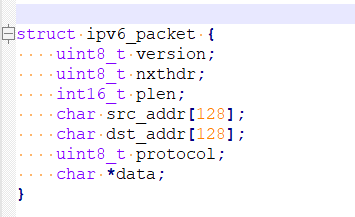
The NetIpHdr struct has the following members:

1. **version**: An 8-bit unsigned integer (uint8\_t) representing the version of the IP header.
2. **header\_len**: An 8-bit unsigned integer (uint8\_t) representing the length of the IP header in dwords.
3. **tos**: An 8-bit unsigned integer (uint8\_t) representing the type of service.
4. **total\_len**: A 16-bit unsigned integer (uint16\_t) representing the total length of the datagram in bytes.
5. **id**: A 16-bit unsigned integer (uint16\_t) representing the unique packet identification.
6. **flags**: An 8-bit unsigned integer (uint8\_t) representing the flags field of the IP header.
7. **fragment\_offset**: A 16-bit unsigned integer (uint16\_t) representing the fragment offset field of the IP header.
8. **ttl:** An 8-bit unsigned integer (uint8\_t) representing the time to live of the packet.
9. **protocol**: An 8-bit unsigned integer (uint8\_t) representing the upper-level protocol (e.g., UDP, TCP).
10. **checksum**: A 16-bit unsigned integer (uint16\_t) representing the IP header checksum.
11. **src\_addr**: A 32-bit unsigned integer (uint32\_t) representing the source IP address.
12. **dest\_addr**: A 32-bit unsigned integer (uint32\_t) representing the destination IP address.
13. **option**: A pointer to an 8-bit unsigned integer (uint8\_t) representing the optional data in the IP header.
14. **option\_len**: An 8-bit unsigned integer (uint8\_t) representing the length of the optional data.
15. **data**: A pointer to an 8-bit unsigned integer (uint8\_t) representing the payload data of the IP packet.
16. **data\_len**: A 32-bit unsigned integer (uint32\_t) representing the length of the payload data.

**The decoding process involves extracting specific fields from the raw packet data and populating the struct members accordingly. Here is the breakdown of the decoding steps:**

1. Create an instance of the NetIpHdr struct named ip.
2. Extract the version field from the first byte of the packet data by masking the most significant nibble (4 bits) and shifting it 4 positions to the right. Store the result in ip.version.
3. Extract the header length field from the first byte of the packet data by masking the least significant nibble (4 bits). Store the result in ip.header\_len.
4. Extract the type of service field from the second byte of the packet data and store it in ip.tos.
5. Extract the total length field from bytes 3 and 4 of the packet data. Convert the network byte order to host byte order using the ntos function. Store the result in ip.total\_len.
6. Extract the unique packet identification field from bytes 5 and 6 of the packet data. Convert the network byte order to host byte order using the ntos function. Store the result in ip.id.
7. Extract the flags field from the seventh byte of the packet data by masking the most significant 3 bits and shifting them 5 positions to the right. Store the result in ip.flags.
8. Extract the fragment offset field from bytes 7 and 8 of the packet data by masking the least significant 5 bits. Store the result in ip.fragment\_offset.
9. Extract the time to live field from the ninth byte of the packet data and store it in ip.ttl.
10. Extract the upper-level protocol field from the tenth byte of the packet data and store it in ip.protocol.
11. Extract the IP header checksum field from bytes 11 and 12 of the packet data. Convert the network byte order to host byte order using the ntos function. Store the result in ip.checksum.
12. Extract the source IP address field from bytes 13 to 16 of the packet data. Convert the network byte order to host byte order using the inet\_ntohs function. Store the result in ip.src\_addr.
13. Extract the destination IP address field from bytes 17 to 20 of the packet data. Convert the network byte order to host byte order using the inet\_ntohs function. Store the result in ip.dst\_addr.
14. If the header length is greater than 5 (indicating the presence of optional data), allocate memory for ip.option to store the optional data. Calculate the length of the option data and store it in ip.option\_len. Copy the option data from the packet into the allocated memory.
15. If the header length is not greater than 5 (no optional data), set ip.option to NULL and ip.option\_len to 0.
16. Allocate memory for ip.data to store the payload data. Calculate the length of the payload data and store it in ip.data\_len. Copy the payload data from the packet into the allocated memory.
17. Return the ip struct, containing the decoded IP header information.
18. Note: The code assumes the presence of certain helper functions (ntos, inet\_ntohs) for converting network byte order to host byte order and string representations of IP addresses. These functions are not explicitly shown in the provided code.

**B. Code snippet of the IPV6 PACKET Structure:**



The struct "ipv6\_packet" has the following members:

1. **version:** An 8-bit unsigned integer (uint8\_t) representing the version of the IPv6 protocol.
2. **nxthdr :** An 8-bit unsigned integer (uint8\_t) representing the next header field in the IPv6 packet.
3. **plen :** A 16-bit signed integer (int16\_t) representing the payload length of the IPv6 packet.
4. **src\_addr:** A character array of size 128 representing the source address of the IPv6 packet.
5. **dst\_addr:** A character array of size 128 representing the destination address of the IPv6 packet.
6. **protocol:** An 8-bit unsigned integer (uint8\_t) representing the protocol field in the IPv6 packet.
7. **data :** A pointer to a character (char) representing the data payload of the IPv6 packet.

**The function "Decode\_Ipv6\_Packet" takes a parameter "data" which is a pointer to a character array representing the raw binary data of the IPv6 packet. It decodes the packet and returns an instance of the "ipv6\_packet" struct.**

**The decoding process involves extracting specific fields from the raw packet data and populating the struct members accordingly. Here is the breakdown of the decoding steps:**

1. Create an instance of the "ipv6\_packet" struct named "packet".
2. Extract the version field from the first byte of the packet data. It is obtained by masking the most significant nibble (4 bits) and shifting it 4 positions to the right. Store the result in "packet.version".
3. Extract the nxthdr field from the 7th byte of the packet data and store it in "packet.nxthdr".
4. Extract the plen field from bytes 5 and 6 of the packet data. Since it is a 16-bit value, it needs to be converted from network byte order to host byte order using the ntohs() function. Store the result in "packet.plen".
5. Convert the source address from binary format to human-readable format using the inet\_ntop() function with AF\_INET6 parameter. The source address starts from byte 8 of the packet data and is 128 bits long. Store the result in "packet.src\_addr".
6. Convert the destination address from binary format to human-readable format using the inet\_ntop() function with AF\_INET6 parameter. The destination address is also located at byte 8 of the packet data and is 128 bits long. Store the result in "packet.dst\_addr".
7. Extract the protocol field from the 7th byte of the packet data and store it in "packet.protocol".
8. Set the "data" member of the "packet" struct to the address of the packet data plus an offset of 40 bytes. This assumes that the data payload starts at byte 40 of the packet data.
9. Return the populated "packet" struct