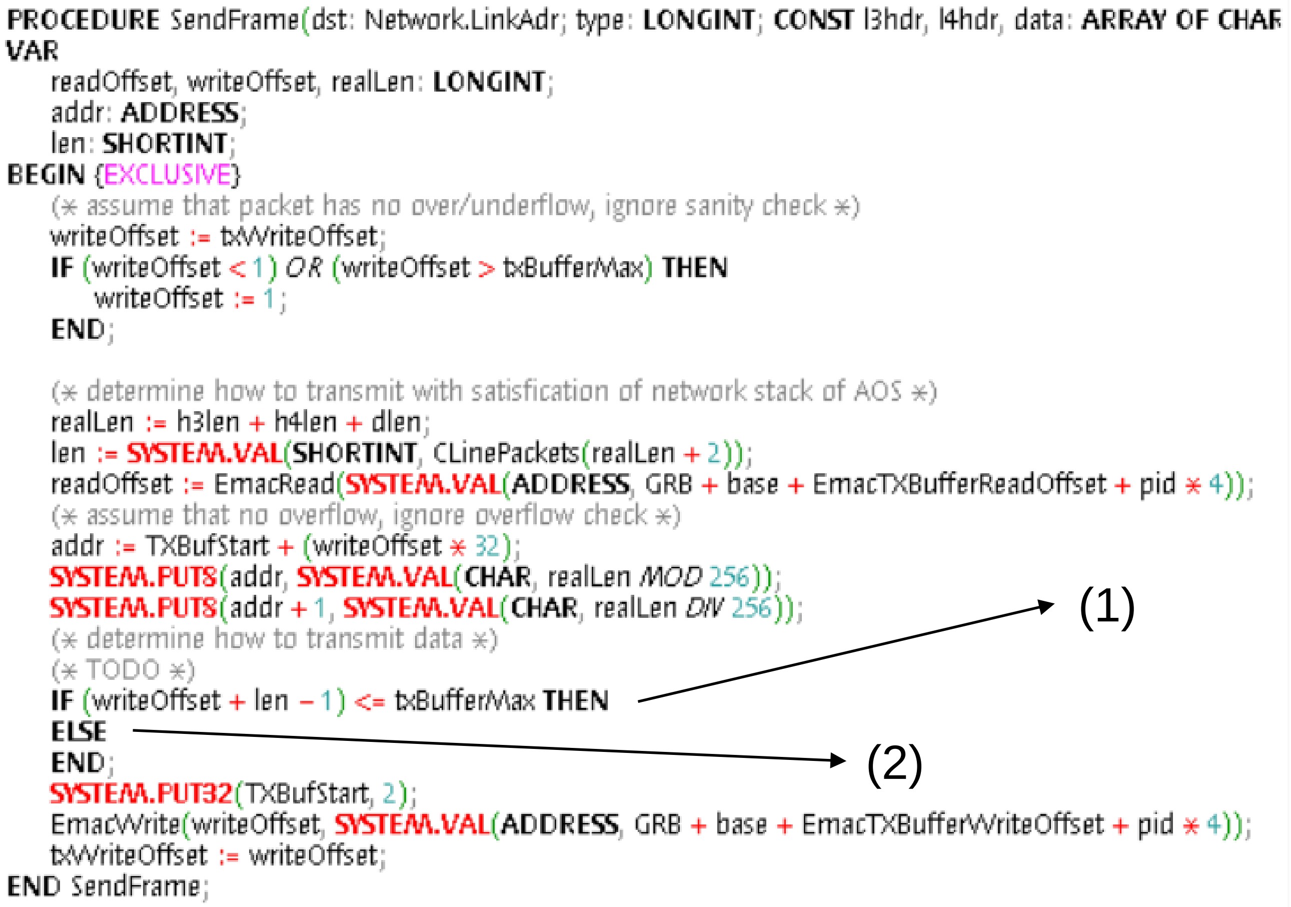
Implementing Packet Transceiver

Jae Min Choi

SNUCSE

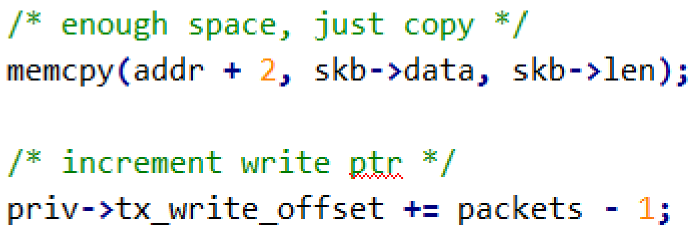
SendFrame (emac_tx)



```
PROCEDURE SendFrame(dst: Network.LinkAdr; type: LONGINT; CONST l3hdr, l4hdr, data: ARRAY OF CHAR
VAR
    readOffset, writeOffset, realLen: LONGINT;
    addr: ADDRESS;
    len: SHORTINT;
BEGIN {EXCLUSIVE}
    (* assume that packet has no over/underflow, ignore sanity check *)
    writeOffset := txWriteOffset;
    IF (writeOffset < 1) OR (writeOffset > txBufferMax) THEN
        writeOffset := 1;
    END;

    (* determine how to transmit with satisfication of network stack of AOS *)
    realLen := h3len + h4len + dlen;
    len := SYSTEM.VAL(SHORTINT, CLinePackets(realLen + 2));
    readOffset := EmacRead(SYSTEM.VAL(ADDRESS, GRB + base + EmacTXBufferReadOffset + pid * 4));
    (* assume that no overflow, ignore overflow check *)
    addr := TXBufStart + (writeOffset * 32);
    SYSTEM.PUT8(addr, SYSTEM.VAL(CHAR, realLen MOD 256));
    SYSTEM.PUT8(addr + 1, SYSTEM.VAL(CHAR, realLen DIV 256));
    (* determine how to transmit data *)
    (* TODO *)
    IF (writeOffset + len - 1) <= txBufferMax THEN
    ELSE
    END;
    SYSTEM.PUT32(TXBufStart, 2);
    EmacWrite(writeOffset, SYSTEM.VAL(ADDRESS, GRB + base + EmacTXBufferWriteOffset + pid * 4));
    txWriteOffset := writeOffset;
END SendFrame;
```

SendFrame (emac_tx)

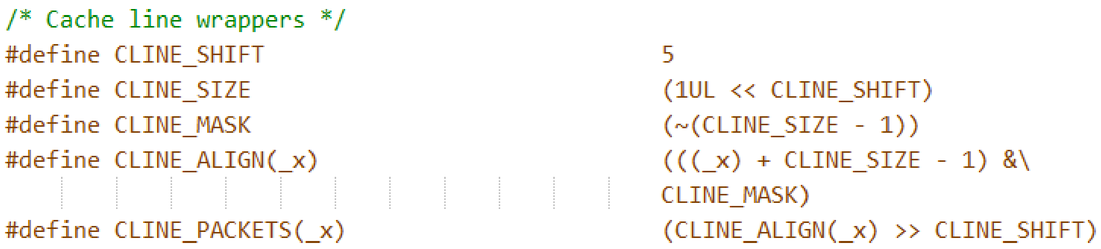
- (1) Enough space in buffer
  - memcpy: use a while loop of SYSTEM.PUT of varying sizes



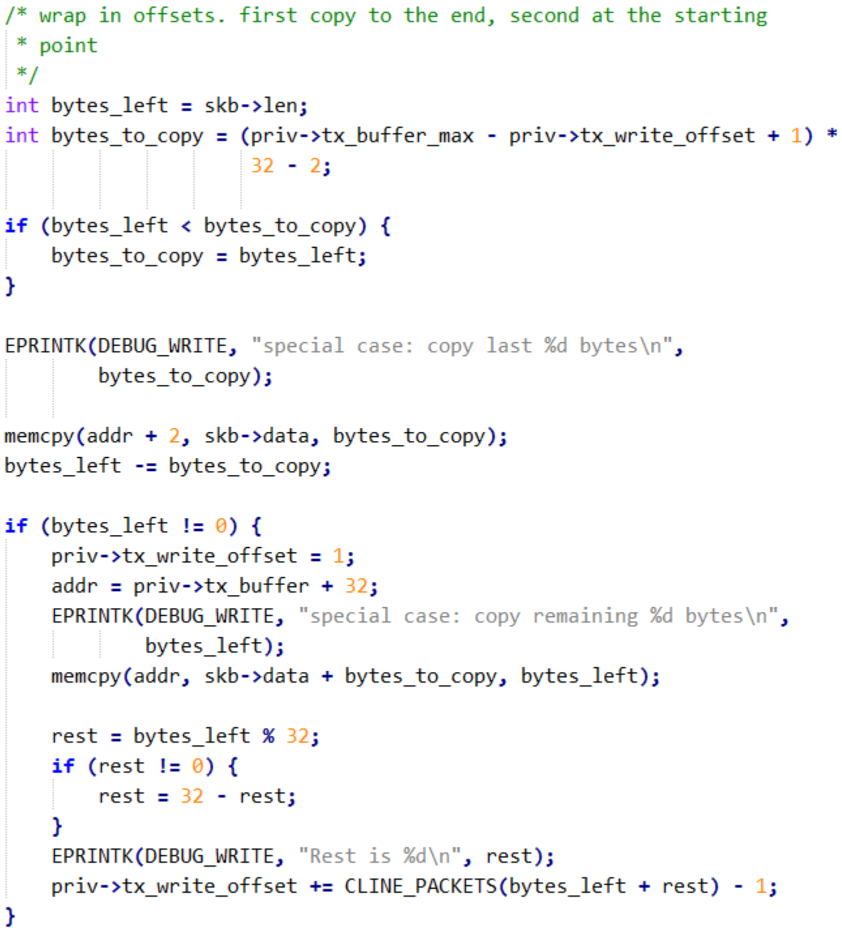
```
/* enough space, just copy */
memcpy(addr + 2, skb->data, skb->len);

/* increment write ptr */
priv->tx_write_offset += packets - 1;
```

SendFrame (emac_tx)

- (2) Not enough space in buffer, wrap to front
  - memcpy: same as (1)
  - CLINE_PACKETS: need to define macro?

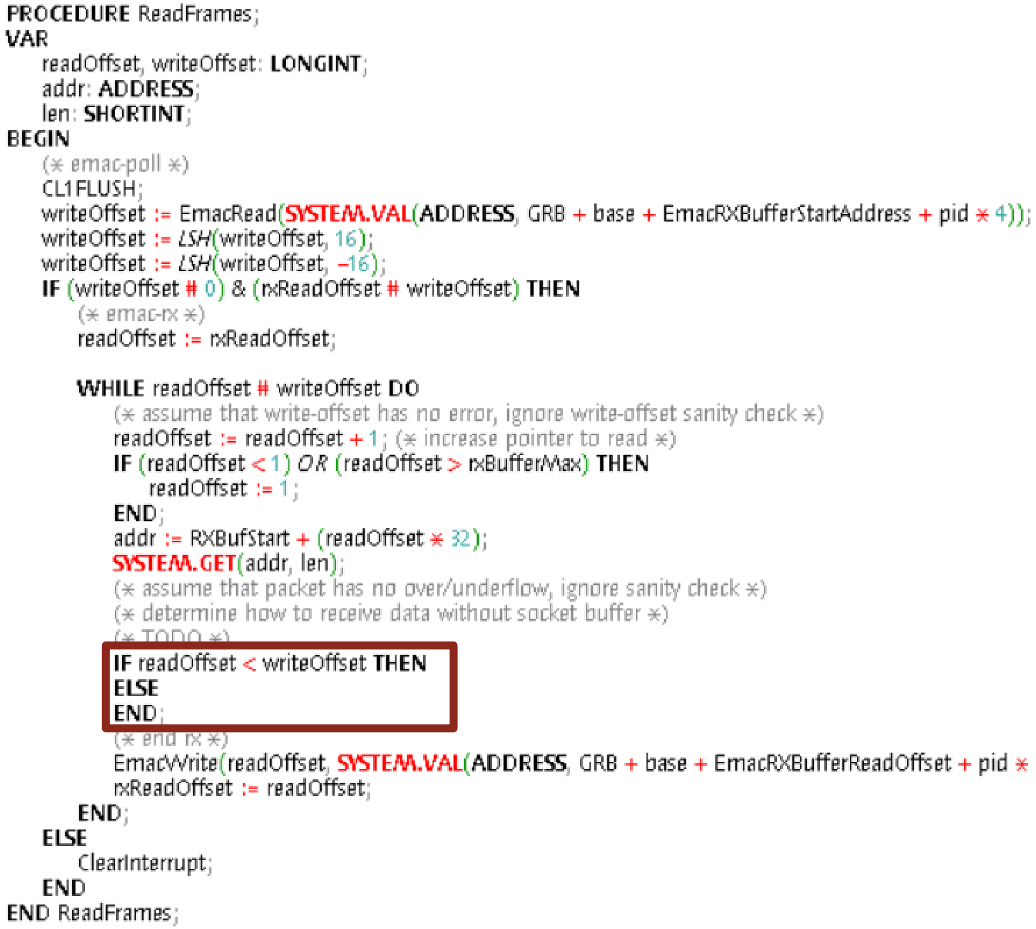


```
/* Cache line wrappers */
#define CLINE_SHIFT                             5
#define CLINE_SIZE                              (1UL << CLINE_SHIFT)
#define CLINE_MASK                              (~(CLINE_SIZE - 1))
#define CLINE_ALIGN(_x)                         (((_x) + CLINE_SIZE - 1) &\
                                                CLINE_MASK)
#define CLINE_PACKETS(_x)                       (CLINE_ALIGN(_x) >> CLINE_SHIFT)
```



```
/* wrap in offsets. first copy to the end, second at the starting
 * point
 */
int bytes_left = skb->len;
int bytes_to_copy = (priv->tx_buffer_max - priv->tx_write_offset + 1) *
                    32 - 2;

if (bytes_left < bytes_to_copy) {
    bytes_to_copy = bytes_left;
}

EPRINTK(DEBUG_WRITE, "special case: copy last %d bytes\n",
        bytes_to_copy);

memcpy(addr + 2, skb->data, bytes_to_copy);
bytes_left -= bytes_to_copy;

if (bytes_left != 0) {
    priv->tx_write_offset = 1;
    addr = priv->tx_buffer + 32;
    EPRINTK(DEBUG_WRITE, "special case: copy remaining %d bytes\n",
            bytes_left);
    memcpy(addr, skb->data + bytes_to_copy, bytes_left);

    rest = bytes_left % 32;
    if (rest != 0) {
        rest = 32 - rest;
    }
    EPRINTK(DEBUG_WRITE, "Rest is %d\n", rest);
    priv->tx_write_offset += CLINE_PACKETS(bytes_left + rest) - 1;
}
```

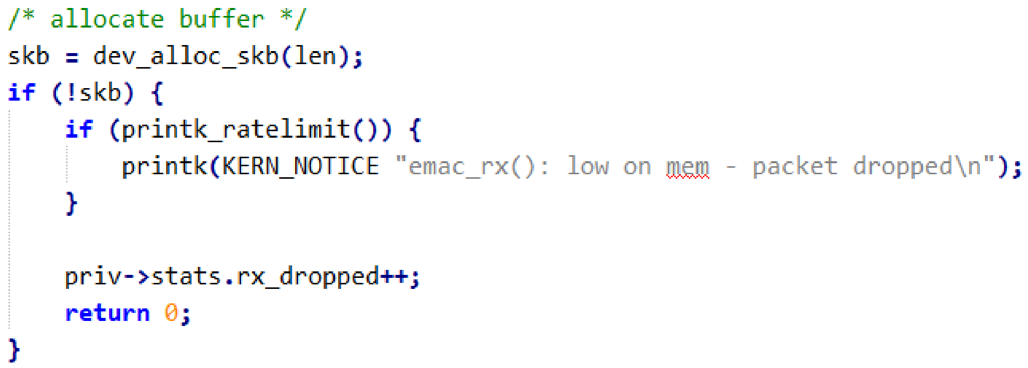
ReadFrames (emac_rx)



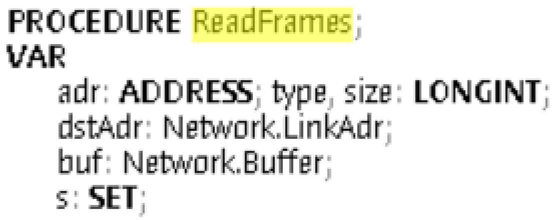
```
PROCEDURE ReadFrames;
VAR
    readOffset, writeOffset: LONGINT;
    addr: ADDRESS;
    len: SHORTINT;
BEGIN
    (* emac-poll *)
    CL1FLUSH;
    writeOffset := EmacRead(SYSTEM.VAL(ADDRESS, GRB + base + EmacRXBufferStartAddress + pid * 4));
    writeOffset := LSH(writeOffset, 16);
    writeOffset := LSH(writeOffset, -16);
    IF (writeOffset # 0) & (rxReadOffset # writeOffset) THEN
        (* emac-rx *)
        readOffset := rxReadOffset;

        WHILE readOffset # writeOffset DO
            (* assume that write-offset has no error, ignore write-offset sanity check *)
            readOffset := readOffset + 1; (* increase pointer to read *)
            IF (readOffset < 1) OR (readOffset > rxBufferMax) THEN
                readOffset := 1;
            END;
            addr := RXBufStart + (readOffset * 32);
            SYSTEM.GET(addr, len);
            (* assume that packet has no over/underflow, ignore sanity check *)
            (* determine how to receive data without socket buffer *)
            (* TODO *)
            IF readOffset < writeOffset THEN
            ELSE
            END;
            (* end rx *)
            EmacWrite(readOffset, SYSTEM.VAL(ADDRESS, GRB + base + EmacRXBufferReadOffset + pid *
            rxReadOffset := readOffset;
        END;
    ELSE
        ClearInterrupt;
    END
END ReadFrames;
```

ReadFrames (emac_rx)

- Allocating buffer
  - We don’t use “sk_buff”
  - Use “Network.Buffer” instead (From RTL8169.Mod)
- Addresses should be of “Network.LinkAddr” type



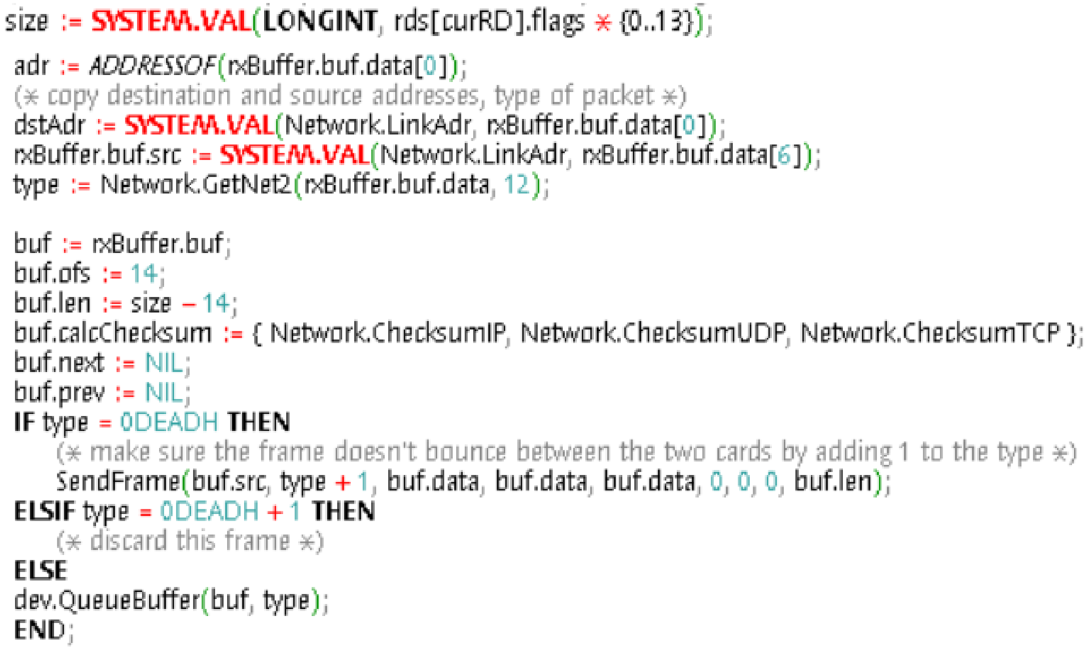
```
/* allocate buffer */
skb = dev_alloc_skb(len);
if (!skb) {
    if (printk_ratelimit()) {
        printk(KERN_NOTICE "emac_rx(): low on mem - packet dropped\n");
    }

    priv->stats.rx_dropped++;
    return 0;
}
```



```
PROCEDURE ReadFrames;
VAR
    adr: ADDRESS; type, size: LONGINT;
    dstAdr: Network.LinkAdr;
    buf: Network.Buffer;
    s: SET;
```

ReadFrames (emac_rx)

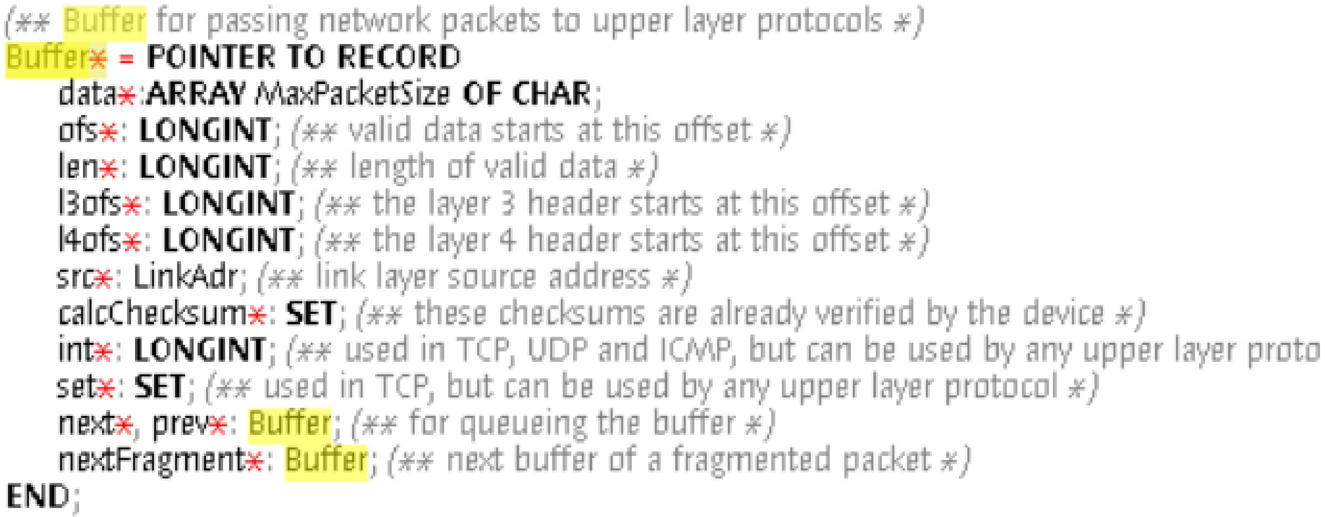
- Set fields in buf (Network.Buffer) accordingly
  - Use SYSTEM.PUT for memcpying data into buf
  - size, ofs?
  - More on the buffer structure in the next slide
- type variable should be set
  - Will be passed to dev.QueueBuffer(), which is equivalent to netif_rx()



```
size := SYSTEM.VAL(LONGINT, rds[curRD].flags * {0..13});
 adr := ADDRESSOF(rxBuffer.buf.data[0]);
 (* copy destination and source addresses, type of packet *)
 dstAdr := SYSTEM.VAL(Network.LinkAdr, rxBuffer.buf.data[0]);
 rxBuffer.buf.src := SYSTEM.VAL(Network.LinkAdr, rxBuffer.buf.data[6]);
 type := Network.GetNet2(rxBuffer.buf.data, 12);

 buf := rxBuffer.buf;
 buf.ofs := 14;
 buf.len := size - 14;
 buf.calcChecksum := { Network.ChecksumIP, Network.ChecksumUDP, Network.ChecksumTCP };
 buf.next := NIL;
 buf.prev := NIL;
 IF type = 0DEADH THEN
     (* make sure the frame doesn't bounce between the two cards by adding 1 to the type *)
     SendFrame(buf.src, type + 1, buf.data, buf.data, buf.data, 0, 0, 0, buf.len);
 ELSIF type = 0DEADH + 1 THEN
     (* discard this frame *)
 ELSE
 dev.QueueBuffer(buf, type);
 END;
```

Buffer Structure

- Defined in I386.Network.Mod
- In ReadFrames, we need to set
  - data
  - ofs
  - len
  - next, prev



```
(** Buffer for passing network packets to upper layer protocols *)
Buffer* = POINTER TO RECORD
    data*:ARRAY MaxPacketSize OF CHAR;
    ofs*: LONGINT; (** valid data starts at this offset *)
    len*: LONGINT; (** length of valid data *)
    l3ofs*: LONGINT; (** the layer 3 header starts at this offset *)
    l4ofs*: LONGINT; (** the layer 4 header starts at this offset *)
    src*: LinkAdr; (** link layer source address *)
    calcChecksum*: SET; (** these checksums are already verified by the device *)
    int*: LONGINT; (** used in TCP, UDP and ICMP, but can be used by any upper layer proto
    set*: SET; (** used in TCP, but can be used by any upper layer protocol *)
    next*, prev*: Buffer; (** for queueing the buffer *)
    nextFragment*: Buffer; (** next buffer of a fragmented packet *)
END;
```