Pbuf: intro

24 February 2026 23:16

In lwIP, a **pbuf (packet buffer)** is the fundamental memory container used to hold:

- Ethernet frame
- IP header
- UDP/TCP header
- Payload

Instead of copying packet data multiple times between layers, lwIP uses **chained buffers**.
Think of it as:

[Ethernet][IP][UDP][DATA]

In lwIP, a struct pbuf does **NOT** mean "one full packet always".
It means:
A buffer node that *may contain part of a packet*
Multiple pbufs can chain together to represent **one logical packet**

7 bytes Preamble
1 byte Start Frame Delimiter
14 bytes Ethernet header
20 bytes IP
8 bytes UDP
5 bytes Data
4 bytes FCS (Frame Check Sequence)
12 bytes Interframe gap (not part of frame but spacing)

} Lwip scope

Case 1 — Packet Fits in One pbuf

Your image example:

14 bytes Ethernet
20 bytes IP
8 bytes UDP
5 bytes Data
---
47 bytes total
If the pool buffer size is, say, 512 bytes:

pbuf A
+--------------------------------------+
| ETH | IP | UDP | DATA |
+--------------------------------------+
next = NULL
Everything fits → only one pbuf needed.

◇ Case 2 — Packet Larger Than Pool Buffer

Assume:

- Pool buffer = 128 bytes
- Packet = 300 bytes

Now lwIP builds a **chain**:

Pbuf: intro cntd..

25 February 2026 14:26

```
pbuf A (128 bytes)
+-----------------------------------+
| ETH | IP | UDP | Data (first part)  |
+-----------------------------------+
          |
          v

pbuf B (128 bytes)
+-----------------------------------+
| Data (middle part)                |
+-----------------------------------+
          |
          v

pbuf C (remaining bytes)
+-----------------------------------+
| Data (last part)                  |
+-----------------------------------+
          |
          v
        NULL
```

This represents:

pbuf A --> pbuf B --> pbuf C --> NULL
But logically, lwIP sees:

| ETH | IP | UDP | COMPLETE DATA |

Visualizing tot_len and len in a Chain

Let's extend your diagram with real lwIP fields:

pbuf A:
len = 128
tot_len = 300

pbuf B:
len = 128
tot_len = 172

pbuf C:
len = 44
tot_len = 44
So:

tot_len = len + next->tot_len
This matches the lwIP invariant you were studying earlier.

Zero-Copy RX Example (Very Important for Zynq)

On Zynq + lwIP + Ethernet DMA:
The DMA hardware fills multiple descriptors.
Each descriptor becomes one pbuf:

DMA Desc1 → pbuf A
DMA Desc2 → pbuf B
DMA Desc3 → pbuf C
No memcpy.
That's why pbuf is a linked list — it mirrors hardware RX descriptors.

Why Descriptors Exist

The Ethernet MAC does **not** directly understand lwIP's pbuf.
Instead, it uses **DMA (Direct Memory Access)**.
DMA needs instructions:

- Where is the buffer in RAM?
- How large is it?
- Is this the first buffer?
- Is this the last buffer?
- Has transmission finished?

That information is stored in a **descriptor**.

pbuf: struct pbuf *next

24 February 2026 23:22

Purpose

Forms a **singly linked list** of buffers.

Why?

Packets may not fit in a single buffer.

Example:

pbuf A --> pbuf B --> pbuf C --> NULL

Used when:

- Packet is larger than pool size
- Zero-copy RX from Ethernet driver
- Fragmented IP packets
- TCP segmentation

pbuf len & tot_len

25 February 2026 15:03

len

Length of the data stored in **this pbuf only**.

Important Distinction

If the packet is chained:

pbuf A: len = 128
pbuf B: len = 128
pbuf C: len = 44
Each len refers only to that segment.

tot_len

Total length of this pbuf **plus all "following" (not considering previous pbuf in same cahin) pbufs in the chain**.

Invariant

p->tot_len = p->len + (p->next ? p->next->tot_len : 0)

Example

pbuf A: len = 128, tot_len = 300
pbuf B: len = 128, tot_len = 172
pbuf C: len = 44, tot_len = 44
Only the first pbuf's tot_len equals the total packet size.

Why It Matters

When transmitting or processing a packet, the full packet size is obtained from tot_len, not len.

Pbuf: LWIP_PBUF_REF_T ref & idx

25 February 2026 16:13

LWIP_PBUF_REF_T ref

Definition

Reference counter.

Purpose

Tracks how many entities are referencing this pbuf.
Possible holders:

- Network stack
- Application
- Another pbuf via chaining

Behavior

- Incremented when shared
- Decremented via pbuf_free()
- Memory released when ref == 0

This prevents premature deallocation.

Example:

- Stack holds 1 reference
- Application holds 1 reference
- Total ref = 2

When you call:

pbuf_free(p);
It decreases ref.
Only when ref == 0 → memory is freed.

if_idx

Index of network interface where packet arrived.
In Zynq usually:

0 → first netif

- 1 → Second network interface

Primarily used internally by the stack.

Pbuf: if_idx & flags

25 February 2026 16:14

if_idx

Index of the network interface that received the packet.

Purpose

Allows identification of the ingress interface in multi-interface systems.

Example:

- 0 → First network interface
- 1 → Second network interface

Primarily used internally by the stack.

flags

Miscellaneous control flags.

Examples

- Broadcast packet indicator
- Multicast packet indicator
- Checksum-related flags

Primarily used internally by lwIP.

Now this is more interesting.

These flags indicate whether:

- Checksum is already verified by hardware
- Checksum must be computed in software
- Checksum generation is required

Examples:

PBUF_FLAG_L4_CHECKSUM_OK
PBUF_FLAG_L4_CHECKSUM_BAD

Indicates that the Ethernet frame was sent to:

FF:FF:FF:FF:FF:FF
i.e., Layer-2 broadcast.
lwIP sets a flag internally such as:

PBUF_FLAG_LLBCAST

Indicates Ethernet destination MAC is multicast:

01:00:5E:xx:xx:xx
Used by:

- IGMP
- mDNS
- Streaming protocols
- Some UDP-based discovery systems

Flag example:

PBUF_FLAG_LLMCAST

pbuf: type_internal -> PBUF_POOL

25 February 2026 16:40

nternal bitfield describing pbuf type and allocation source.

Indicates:

- Memory origin (pool, RAM, ROM, reference)
- Allocation flags

Typical categories:

- PBUF_POOL
- PBUF_RAM
- PBUF_ROM
- PBUF_REF

Used internally for memory management decisions.

PBUF_POOL

When lwIP starts, it allocates a **fixed number of buffers in DDR memory**.
You configure:

```
#define PBUF_POOL_SIZE     N
#define PBUF_POOL_BUFSIZE  M
```

This means:

At boot, lwIP reserves N blocks of memory.
Each block contains space for one packet payload of size M bytes.

It is a **preallocated array of packet buffers in RAM**.
If:

PBUF_POOL_SIZE = 512
PBUF_POOL_BUFSIZE = 1536
Total DDR used ≈

512 × (1536 + ~32) ≈ ~800 KB
This memory is reserved once at startup.
It does NOT grow.
It does NOT shrink.
It does NOT fragment.
It is fixed.

What Happens During RX (Step-by-Step)

Now let's trace one Ethernet packet.

Step 1 — Before Packet Arrives

At boot:

Pool contains N empty buffers
All marked FREE

Step 2 — Packet Arrives at GEM

Ethernet frame arrives:

| Dest MAC | Src MAC | Type | Data |
GEM hardware:

1. Takes an RX descriptor
2. Gets buffer address (which points to a PBUF_POOL payload)
3. Writes packet directly into that payload buffer

So physically:

DDR Memory (Pool Entry #17)

```
+----------------------+
| struct pbuf          |
+----------------------+
|  Ethernet Frame      |  <-- DMA writes here
+----------------------+
```

NO memcpy.
DMA writes directly into pool memory.

Step 3 — Driver Wraps It

Driver marks this pool entry as:

ALLOCATED
It sets:

p->len = received_length
p->tot_len = received_length
Then passes pointer to lwIP stack.

Why This Is Called Zero-Copy

Because:

- Hardware writes into pool memory
- Stack processes same memory
- Application reads same memory

There is **no intermediate buffer copy**.
This is extremely important for:

- High UDP throughput
- Low CPU usage
- Deterministic latency

PBUF_RAM

25 February 2026 17:39

PBUF_RAM means:

Allocate a new buffer dynamically from lwIP heap.

Unlike PBUF_POOL, this memory is **not preallocated at boot**.

It is allocated when requested.

What Exactly Gets Allocated in Memory

When:

```
p = pbuf_alloc(PBUF_TRANSPORT, length, PBUF_RAM);
```

lwIP performs a heap allocation:

```
+------------------------+
| struct pbuf            |
+------------------------+
| payload buffer (length)|
+------------------------+
```

This is one contiguous block from the lwIP heap.

On Zynq-7000 bare-metal:

- Heap is in DDR
- Usually cacheable memory
- Managed by lwIP's internal allocator (mem_malloc)

Where This Memory Lives on Zynq

Typically:

```
DDR
 ├── Code
 ├── Stack
 ├── Heap  ← PBUF_RAM allocated here
 ├── PBUF_POOL region
 └── Other data
```

So PBUF_RAM uses general-purpose heap memory.

It is:

- DMA accessible (if DDR)
- Not reserved in advance
- Subject to fragmentation

4 What Happens During TX (Step-by-Step)

Now let's trace an outgoing UDP packet.

Step 1 — Application Creates Packet

```
p = pbuf_alloc(PBUF_TRANSPORT, 100, PBUF_RAM);
memcpy(p->payload, data, 100);
```

Memory now looks like:

Heap:

```
+----------------------------+
| struct pbuf                |
+----------------------------+
| UDP header + data (100B)   |
+----------------------------+
```

Step 2 — Stack Adds Headers

lwIP pushes:

- UDP header
- IP header
- Ethernet header

All inside the same RAM buffer.

Memory becomes:

```
| Ethernet | IP | UDP | Data |
```

Still inside same allocated block.

Step 3 — GEM Transmits

Driver:

- Creates TX descriptor
- Sets descriptor buffer pointer to p->payload
- Starts DMA

DMA reads from DDR and sends to wire.

PBUF_ROM

25 February 2026 18:11

PBUF_ROM means:

The payload already exists somewhere in read-only memory.
lwIP should NOT allocate or copy payload memory.

Only the struct pbuf is allocated.
The payload memory is external and constant.

Where the Payload Actually Lives

Typically in:

.rodata section
Which may be:

- QSPI flash (if XIP)
- DDR (if program copied to DDR at boot)
- On-chip memory

Example:

static const char msg[] = "HELLO";
This string is stored in .rodata.
Memory layout:

Flash or DDR (.rodata):
"HELLO"

Heap:
[struct pbuf only]
The payload is NOT allocated again.

What Happens During Transmission (Zynq GEM)

Step 1 — pbuf points to constant memory

p->payload → address of msg in .rodata

Step 2 — lwIP adds headers

Depending on configuration, lwIP may create a small header pbuf in front.

Step 3 — Driver sets TX descriptor

Descriptor buffer pointer = address of .rodata.

Step 4 — GEM DMA reads directly from that memory

DMA transfers from:

Flash or DDR
to Ethernet wire.
No copy.

PBUF_REF

25 February 2026 18:22

PBUF_REF means:

lwIP does NOT allocate payload memory.

It only wraps an already-existing RAM buffer.

The payload buffer is owned by the application (or external system).

lwIP just references it.

When you do:

```
p = pbuf_alloc(PBUF_TRANSPORT, 0, PBUF_REF);
p->payload = external_buffer;
p->len = size;
p->tot_len = size;
```

Allocated:

[struct pbuf]

NOT allocated:

No payload memory

So memory layout is:

Application Buffer (DDR or stack)

[data .....]

Heap

[struct pbuf]

Where Payload Memory Lives on Zynq

Payload could be:

- DDR buffer
- On-chip memory
- Static global buffer
- Stack buffer (dangerous)

For DMA transmission on Zynq:

The buffer MUST:

- Be in DDR
- Be physically contiguous
- Be DMA accessible

Otherwise GEM cannot transmit it.

What Happens During TX (Step-by-Step)

Let's trace it clearly.

Step 1 — Application Has Buffer

```
uint8_t txbuf[100];
```

Memory in DDR:

0x10005000:

[100 bytes of data]

Step 2 — Create PBUF_REF

```
p = pbuf_alloc(PBUF_TRANSPORT, 0, PBUF_REF);
p->payload = txbuf;
p->len = 100;
p->tot_len = 100;
```

Now:

p->payload → 0x10005000

Step 3 — lwIP Adds Headers

lwIP may create small header pbufs in front.

Final structure might be:

pbuf A (Ethernet header)

↓

pbuf B (IP + UDP header)

↓

pbuf C (payload → txbuf)

Step 4 — Driver Creates TX Descriptors

Driver walks the pbuf chain.

For each pbuf:

Descriptor buffer pointer = pbuf->payload

For payload part:

Descriptor → 0x10005000

Step 5 — GEM DMA Reads Directly

DMA reads directly from that external buffer and sends to wire.

No copy.

pbuf internal summary

25 February 2026 18:42

Type	Who Owns Payload	Allocates Payload?	Zero Copy	Deterministic	Risk Level
PBUF_POOL	lwIP pool	Yes (preallocated)	Yes (RX)	High	Low
PBUF_RAM	lwIP heap	Yes (dynamic)	No	Medium	Medium
PBUF_ROM	Read-only memory	No	Yes	High	Low
PBUF_REF	Application	No	Yes	High	High

pbuf functions: pbuf_alloc, pbuf_free

25 February 2026 18:51

pbuf_alloc

Syntax

```
struct pbuf *pbuf_alloc(pbuf_layer layer,
                        u16_t length,
                        pbuf_type type);
```

Purpose

Allocates a packet buffer.

Operates On

- Allocates memory from:
  - PBUF_POOL
  - PBUF_RAM
  - PBUF_ROM
  - PBUF_REF
- ~~Returns pointer to struct pbuf~~

pbuf_free

Syntax

```
u8_t pbuf_free(struct pbuf *p);
```

Purpose

Decrements reference count.
Frees buffer when ref == 0.

Operates On

- Returns memory to:
  - Pool
  - Heap
- Releases RX buffer back to system