arr_conn[CSP_CONN_MAX]	0				.....	CSP_CONN_MAX-1
	队列	元素	长度	创建		
	rx_event	int	CSP_CONN_QUEUE_LENGTH	csp_conn_init		
	rx_queue[CSP_RX_QUEUES]	csp_packet_t *	CSP_RX_QUEUE_LENGTH	csp_conn_init		
	socket	csp_packet_t * csp_conn_t*	CSP_CONN_QUEUE_LENGTH	csp_socket		

往socket.socket队列发送包地址

出列sock->socket

出列conn->rx_event, conn->rx_queue[prio]

出列socket->socket

csp_accept

csp_read

csp_recvfrom

用户代码使用packet

csp_buffer_free (packet_copy) 用户使用完后释放

csp_route_work RTE

csp_conn_check_timeouts

csp_qfifo_read

csp_promisc_add

csp_buffer_free

csp_dedup_is_duplicate

目标地址不是本机且非广播

csp_route_check_options =CSP_ERR_NONE

socket = csp_port_get_socket(packet->id.dport); 查找socket

socket && (socket->opts & CSP_SO_CONN_LESS)

csp_conn_find(packet->id.ext,CSP_ID_CONN_MASK); 查找conn

1.socket不存在，conn存在，
2.socket存在,conn不存在则创建conn

is loop?

csp_send_direct

往conn->rx_queue[rxq]发送包地址往conn->rx_event发送通知消息int数据

csp_queue_enqueue(socket->socket, &packet, 0) != CSP_QUEUE_OK

csp_buffer_free

csp_route_security_check?

conn == NULL

socket=0

csp_route_security_check==OK?

csp_conn_new(packet->id, idout)==OK?

csp_route_security_check==OK?

if (packet->id.flags & CSP_FRDP)?

csp_rdp_new_packet(conn, packet);

csp_udp_new_packet(conn, packet);

csp_conn_enqueue_packet(conn, packet)

成功

csp_buffer_free(packet);

csp_queue_enqueue(conn->socket, &conn, 0)

csp_qfifo_read

csp_qfifo_read

driver_can

CAN接收中断

CAN接收缓冲

can_rxtask

can_rx_handle

driver_can_recv

有数据?

超过次数?

csp_can_pbuf_cleanup

csp_can_process_frame

csp_buffer_free

csp_can_pbuf [PBUF_ELEMENTS]	0			.....	PBUF_ELEMENTS-1
	rx_count	2B	包已接收长度		
	remain	4B	包剩余接收长度		
	cfpid	4B	包ID		
	packet	4B	指向包数据区		
	state	4B	该缓冲是否使用		
	last_used	4B	最后使用该缓冲时间		

csp_buffer_pool

packet=csp_buffer_get(CSP_CAN_MTU);

见csp_buffer页详述

csp_qfifo_write

qfifo[CSP_ROUTE_FIFOS]	0			PBUF_ELEMENTS-1
	qfifo[0]			qfifo[0]
qfifo_events				

item 0
struct
{
csp_iface_t * interface;
csp_packet_t * packet;
} csp_qfifo_t;
.....
item CSP_FIFO_INPUT-1
struct
{
csp_iface_t * interface;
csp_packet_t * packet;
} csp_qfifo_t;

csp_qfifo_read

csp_bridge BRIDGE

克隆内存

packet_copy = csp_buffer_clone (packet)

csp_qfifo_read

csp_promisc_add

csp_send_direct

csp_buffer_free

csp_buffer_clone

csp_queue_enqueue(csp_promisc_queue

克隆出来的地址值入列

csp_promisc_queue		0	.....	buf_size-1 csp_promisc_enable函数指定
	队列元素	csp_packet_t *		qfifo[0]

出列地址值

csp_promisc_read

用户代码使用packet_copy

csp_buffer_free (packet_copy) 用户使用完后释放

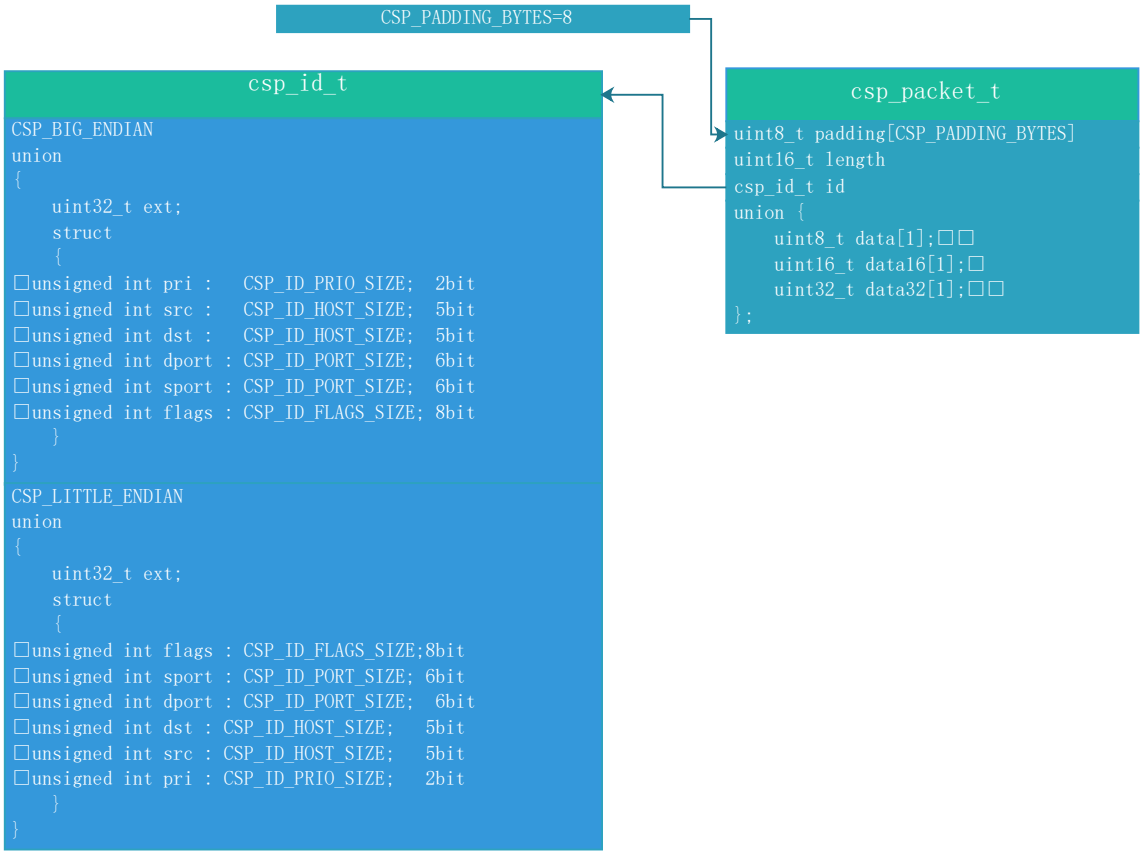
CAN ID 数据流 (32bit) 高位先发 csp_can_tx时设置				
b[28-24]	b[23-19]	b[18]	b[17-10]	b[9-0]
src 发送节点地址	dst 目的节点MAC	type 0 开始帧 1 后续帧	remain 后续帧数	id 全局变量 csp_can_id时递增

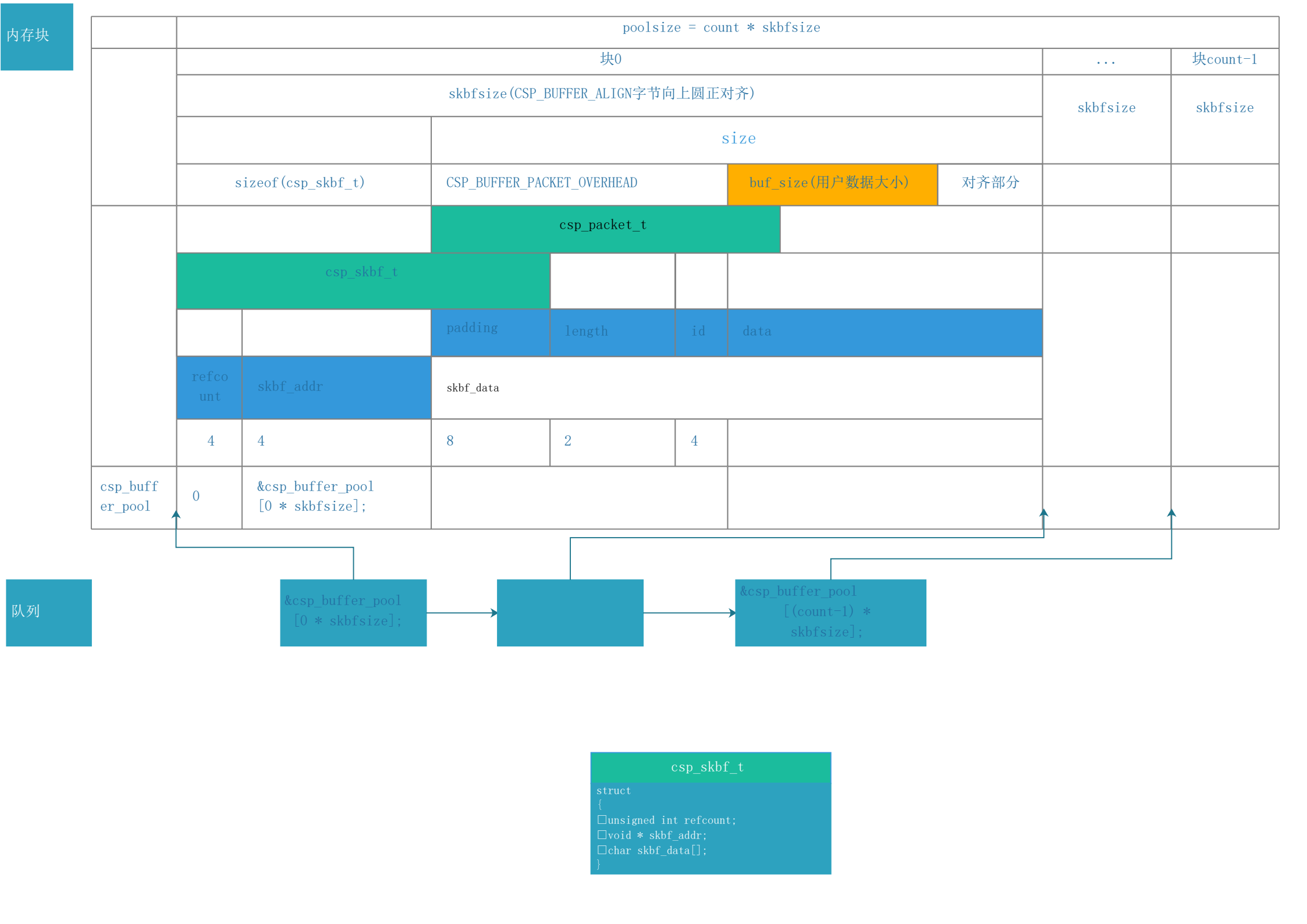
开始帧 8B 数据流 (高字节在前先发 csp_connect时设置ID)							
B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]B[7]	
b7 b0	b7 b0	b7 b0	b7 b0	长度高字节	长度低字节	2字节数据	
pri	src	dst	dport	sport	flags		
优先级0-3,3最低	源地址	目的地址	目的端口	源端口csp_conn_init时初始化全局变量,发数据递增	CSP_FCRC32等标志		

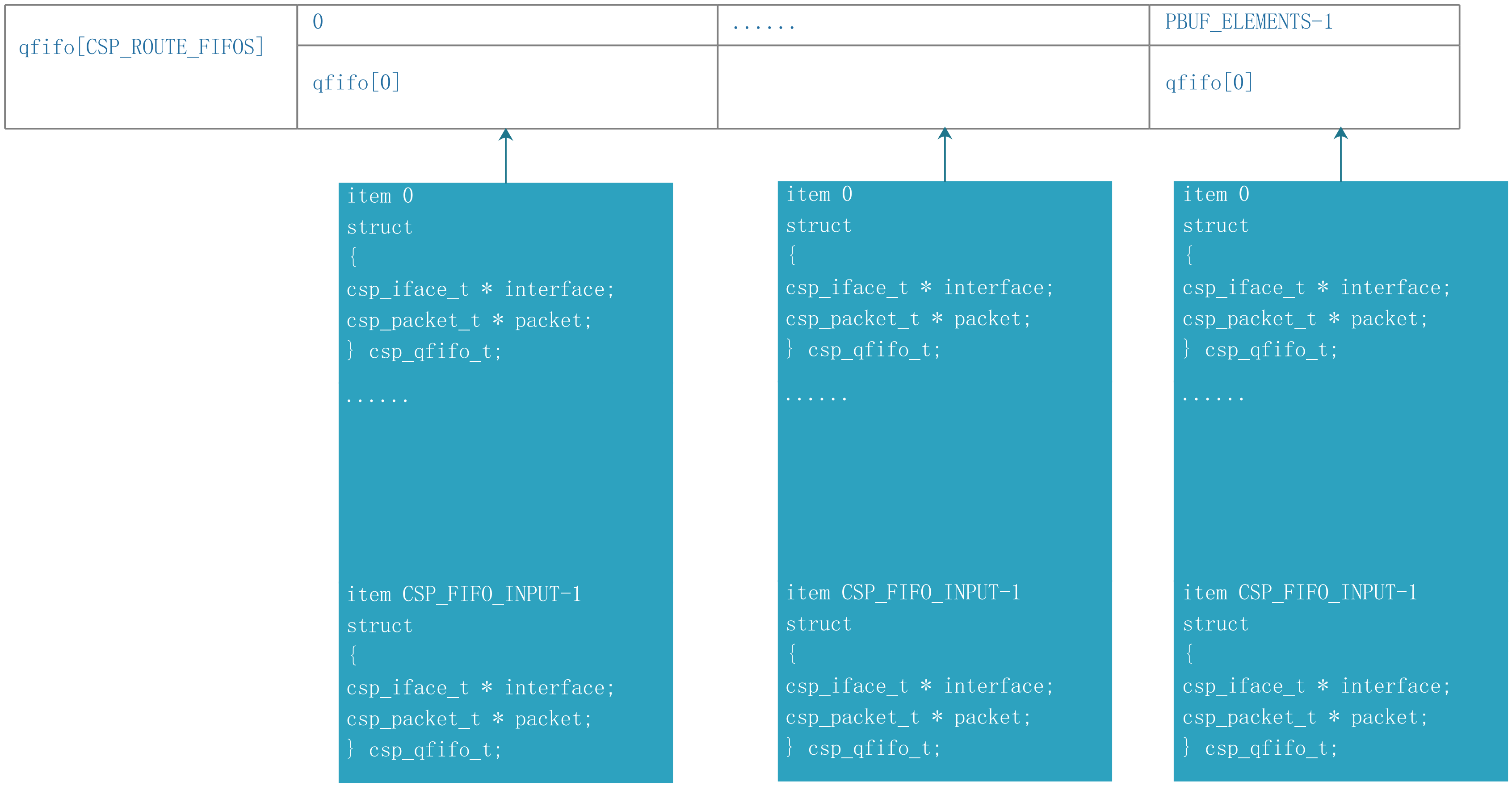
后续帧 数据流8B							
B[n]	B[n+1]	B[n+2]	B[n+3]	B[n+4]	B[n+5]	B[n+6]	B[n+7]
				HMAC			

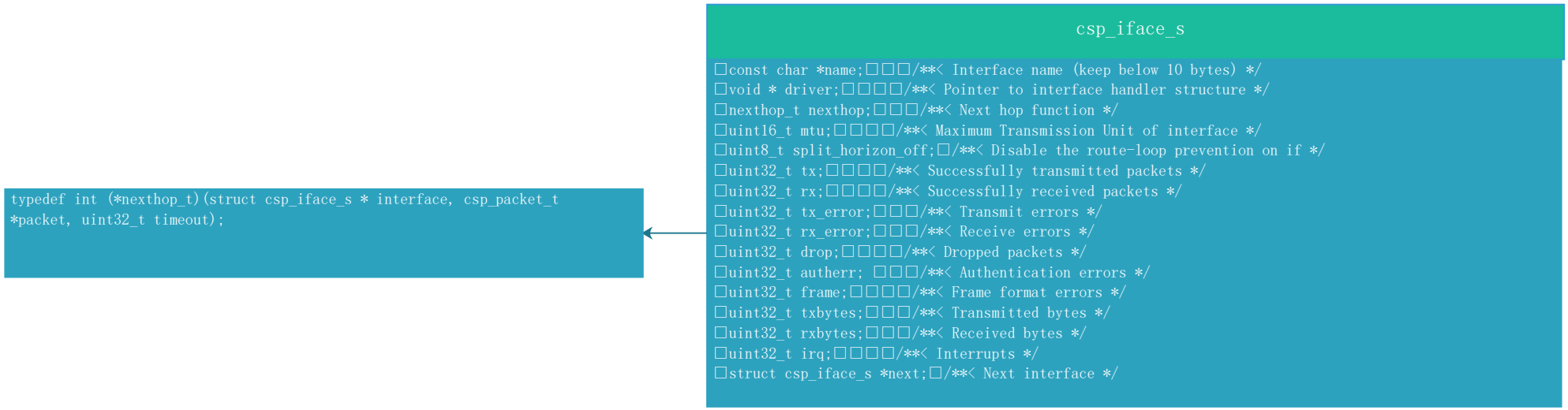
后续帧 校验加密等可选信息							
B[n]	B[n+1]	B[n+2]	B[n+3]	B[n+4]	B[n+5]	B[n+6]	B[n+7]
CRC32				FXTEA			

<table>
<tr><td colspan="10">csp_packet_t</td></tr>
<tr><td>域</td><td>padding</td><td>length</td><td colspan="6">id</td><td>data</td></tr>
<tr><td>字节</td><td>8B</td><td>2B</td><td colspan="6">4B</td><td>n</td></tr>
<tr><td rowspan="3">位域</td><td rowspan="3"></td><td rowspan="3"></td><td colspan="6">ext</td><td rowspan="3"></td></tr>
<tr><td>flags</td><td>spor
t</td><td>dpor
t</td><td>ds
t</td><td>src</td><td>pr
i</td></tr>
<tr><td>8b</td><td>6</td><td>6</td><td>5b</td><td>5b</td><td>2b</td></tr>
</table>



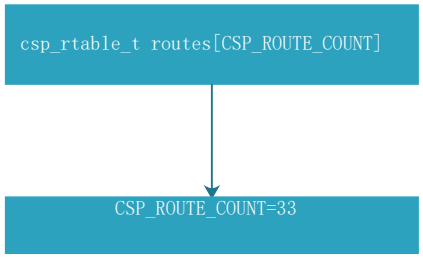

内存块
poolsize = count * skbfsize
块0
...
块count-1
skbfsize(CSP_BUFFER_ALIGN字节向上圆正对齐)
skbfsize
skbfsize
size
sizeof(csp_skbf_t)
CSP_BUFFER_PACKET_OVERHEAD
buf_size(用户数据大小)
对齐部分
csp_packet_t
csp_skbf_t
padding
length
id
data
refco unt
skbf_addr
skbf_data
4
4
8
2
4
csp_buff er_pool
0
&csp_buffer_pool [0 * skbfsize];
队列
&csp_buffer_pool [0 * skbfsize];
&csp_buffer_pool [(count-1) * skbfsize];
csp_skbf_t
struct
{
unsigned int refcount;
void * skbf_addr;
char skbf_data[];
}


qfifo[CSP_ROUTE_FIFOS]
0
......
PBUF_ELEMENTS-1
qfifo[0]
qfifo[0]
item 0
struct
{
csp_iface_t * interface;
csp_packet_t * packet;
} csp_qfifo_t;
......
item CSP_FIFO_INPUT-1
struct
{
csp_iface_t * interface;
csp_packet_t * packet;
} csp_qfifo_t;
item 0
struct
{
csp_iface_t * interface;
csp_packet_t * packet;
} csp_qfifo_t;
......
item CSP_FIFO_INPUT-1
struct
{
csp_iface_t * interface;
csp_packet_t * packet;
} csp_qfifo_t;
item 0
struct
{
csp_iface_t * interface;
csp_packet_t * packet;
} csp_qfifo_t;
......
item CSP_FIFO_INPUT-1
struct
{
csp_iface_t * interface;
csp_packet_t * packet;
} csp_qfifo_t;

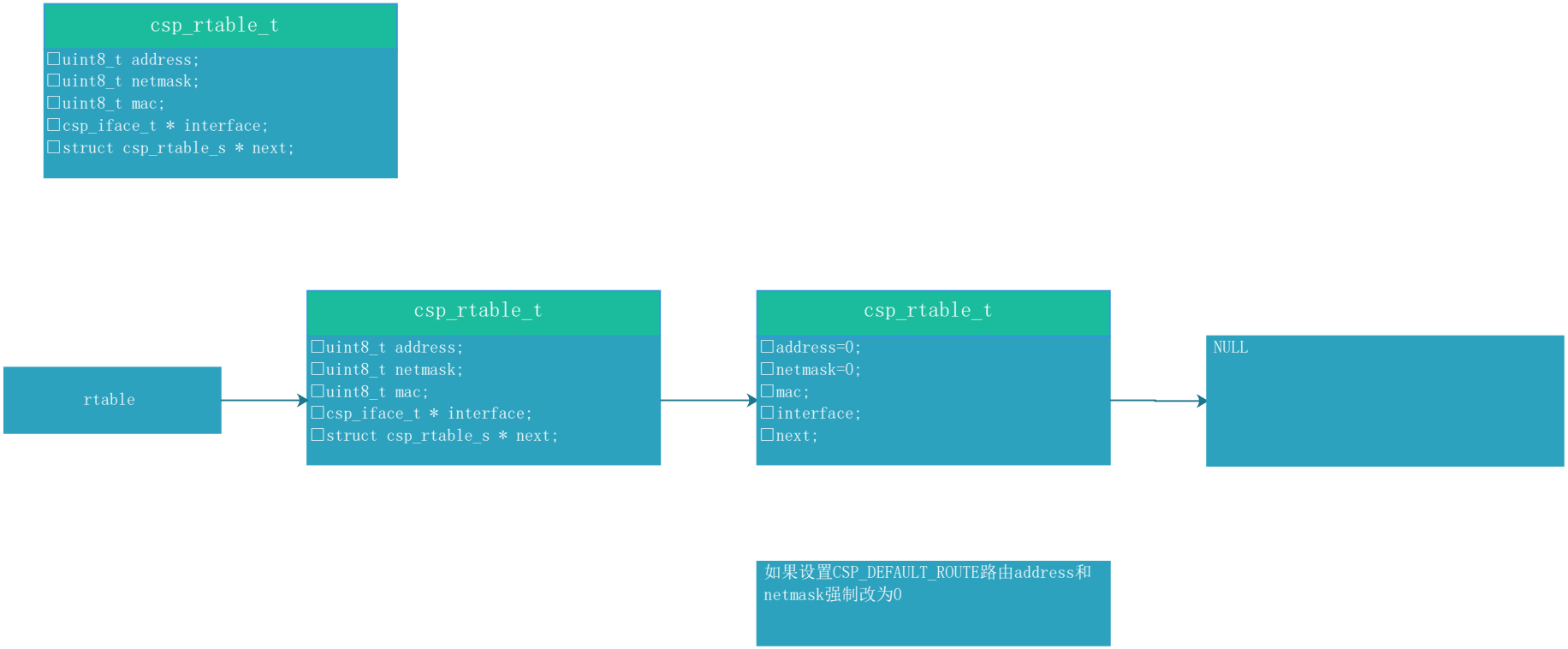

csp_iface_s
□const char *name;□□□/**< Interface name (keep below 10 bytes) */
□void * driver;□□□□/**< Pointer to interface handler structure */
□nexthop_t nexthop;□□□/**< Next hop function */
□uint16_t mtu;□□□□/**< Maximum Transmission Unit of interface */
□uint8_t split_horizon_off;□/**< Disable the route-loop prevention on if */
□uint32_t tx;□□□□/**< Successfully transmitted packets */
□uint32_t rx;□□□□/**< Successfully received packets */
□uint32_t tx_error;□□□/**< Transmit errors */
□uint32_t rx_error;□□□/**< Receive errors */
□uint32_t drop;□□□□/**< Dropped packets */
□uint32_t autherr; □□□/**< Authentication errors */
□uint32_t frame;□□□□/**< Frame format errors */
□uint32_t txbytes;□□□/**< Transmitted bytes */
□uint32_t rxbytes;□□□/**< Received bytes */
□uint32_t irq;□□□□/**< Interrupts */
□struct csp_iface_s *next;□/**< Next interface */
typedef int (*nexthop_t)(struct csp_iface_s * interface, csp_packet_t *packet, uint32_t timeout);

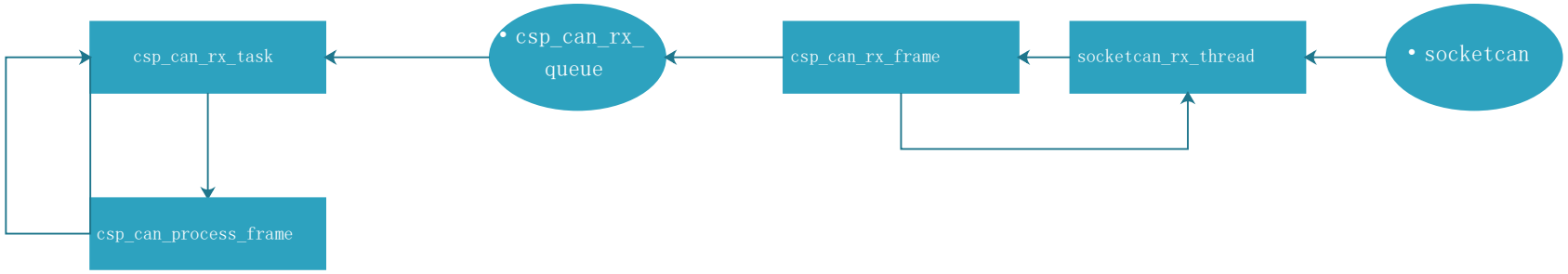
csp_rtable_t

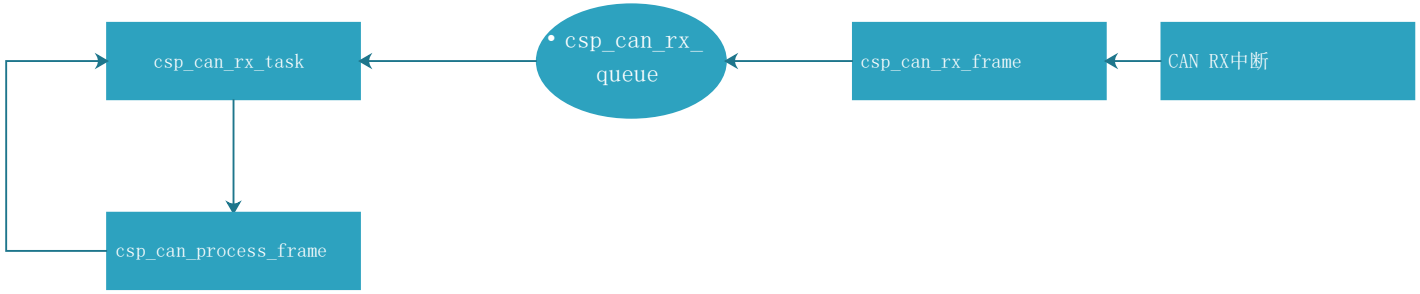
```
struct
{
    csp_iface_t * interface;
    uint8_t mac;
}
```

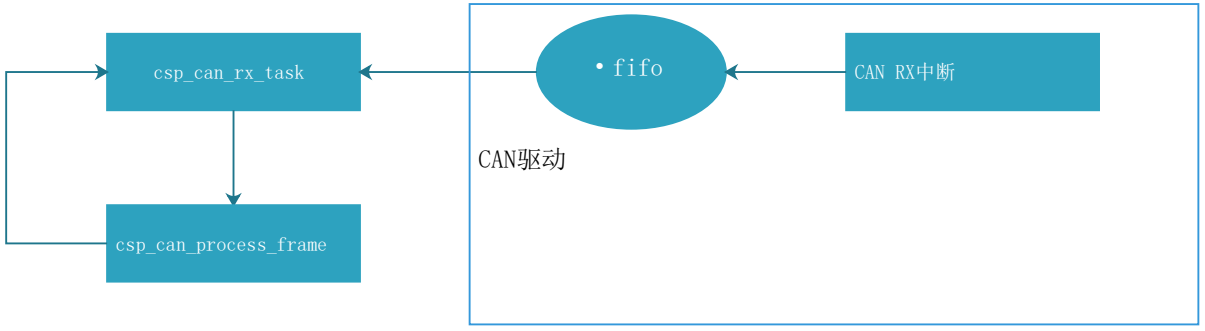


地址/索引	0	...	CSP_BROADCAST_ADDR (31)	CSP_DEFAULT_ROUTE (32)
routes	interface	interface		csp_if_lo
	mac	mac		CSP_NODE_MAC(0xFF)


csp_rtable_t
uint8_t address;
uint8_t netmask;
uint8_t mac;
csp_iface_t * interface;
struct csp_rtable_s * next;
rtable
csp_rtable_t
uint8_t address;
uint8_t netmask;
uint8_t mac;
csp_iface_t * interface;
struct csp_rtable_s * next;
csp_rtable_t
address=0;
netmask=0;
mac;
interface;
next;
NULL
如果设置CSP_DEFAULT_ROUTE路由address和netmask强制改为0


csp_can_rx_task
• csp_can_rx_
queue
csp_can_rx_frame
socketcan_rx_thread
• socketcan
csp_can_process_frame


csp_can_rx_task
• csp_can_rx_
queue
csp_can_rx_frame
CAN RX中断
csp_can_process_frame


csp_can_rx_task
• fifo
CAN RX中断
CAN驱动
csp_can_process_frame