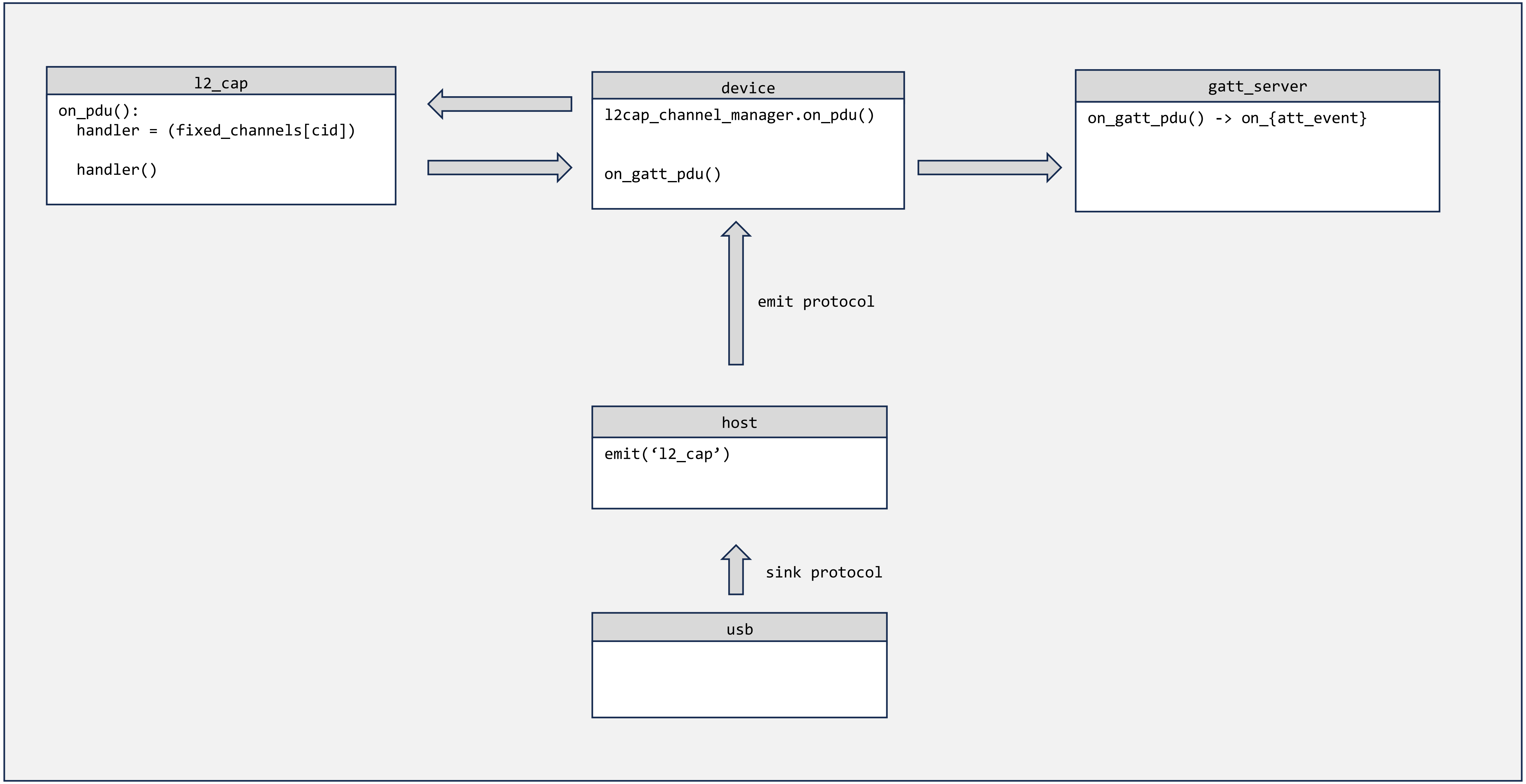
overview


l2_cap
on_pdu():
  handler = (fixed_channels[cid])

  handler()
device
l2cap_channel_manager.on_pdu()
on_gatt_pdu()
gatt_server
on_gatt_pdu() -> on_{att_event}
emit protocol
host
emit(‘l2_cap’)
sink protocol
usb

gatt_server

gatt_server.py

Server

on_gatt_pdu()

```
handler_name =f’on_{att_pdu.name.lower()}’
handler = getattr(self, handler_name)
if handler is not None:
  handler(connection, att_pdu)
else:
  if att_pdu.op_code in ATT_REQUESTS:
    on_att_request(connection, att_pdu)
```

send_gatt_pdu()

```
device.send_l2cap_pdu()
```

send_response()

```
send_gatt_pdu()
```

on_att_request()

```
response = ATT_Error_Response()
send_response(connection, response)
```

on_{ATT event name}

```
# do work on ATT request then
send_response(connection, response)
```

GATT events:

```
on_disconnection(self, connection):
on_gatt_pdu(self, connection, att_pdu):
on_att_request(self, connection, pdu):
```

ATT events:

```
on_att_exchange_mtu_request(self, connection, request):
on_att_find_information_request(self, connection, request):
on_att_find_by_type_value_request(self, connection, request):
on_att_read_by_type_request(self, connection, request):
on_att_read_request(self, connection, request):
on_att_read_blob_request(self, connection, request):
on_att_read_by_group_type_request(self, connection, request):
on_att_write_request(self, connection, request):
on_att_write_command(self, connection, request):
on_att_handle_value_confirmation(self, connection, _confirmation):
```

device

device.py

Device event_handlers:

```
on_inquiry_result
on_flush
on_link_key
on_connection
on_connection_failure
on_connection_request
on_disconnection
on_disconnection_failure
on_inquiry_complete
on_connection_authentication
on_connection_authentication_failure
on_ssp_complete
on_authentication_io_capability_request
on_authentication_io_capability_response
on_authentication_user_confirmation_request
on_authentication_user_passkey_request
on_pin_code_request
on_authentication_user_passkey_notification
on_remote_name
on_remote_name_failure
on_connection_encryption_change
on_connection_encryption_failure
on_connection_encryption_key_refresh
on_connection_parameters_update
on_connection_parameters_update_failure
on_connection_phy_update
on_connection_phy_update_failure
on_connection_att_mtu_update
on_connection_data_length_change
on_role_change
on_role_change_failure
on_l2cap_pdu
```

device

device.py

Device

__init__()

```
gatt_server = gatt_server.Server()
sdp_server = sdp.Server()
l2cap_channel_manager = l2cap.ChannelManager()
l2cap_channel_manager.register_fixed_channel(smp.SMP_CID, self.on_smp_pdu)
l2cap_channel_manager.register_fixed_channel(ATT_CID, self.on_gatt_pdu)
l2cap_channel_manager.register_fixed_channel(smp.SMP_BR_CID, self.on_smp_pdu)
```

host()

```
#set event handlers
for event_name in device_host_event_handlers:
  host.on(event_name, ‘on_{event_name}’)
l2cap_channel_manager.host = host
```

on_l2cap_pdu()

```
l2cap_channel_manager.on_pdu()
```

on_smp_pdu()

```
smp_manager.on_smp_pdu()
```

on_gatt_pdu()

```
# if client
connection.gatt_client.on_gatt_pdu(att_pdu)
#if server
connection.gatt_server.on_gatt_pdu(connection, att_pdu)
```

send_l2cap_pdu()

```
host.send_l2cap_pdu()
```

send_command()

```
host.send_command()
```

Connection

send_l2cap_pdu()

```
device.send_l2cap_pdu()
```

```
Advertisement
LegacyAdvertisement
ExtendedAdvertisement
AdvertistmentDataAccumulater
AcvertisingType
LePhysOptions
Peer
ConnectionParametersPreferences
Connection
DeviceConfiguration
```

l2cap

l2cap.py

ChannelManager

host(host)

```
_host.remove_listener(‘disconnection’)
host.on(‘disconnection’)
```

register_fixed_channel()

```
fixed_channels[cid] = handler
```

on_pdu()

```
#if signalling
  on_control_frame()
#if fixed channels
  handler = fixed_channels[cid]
  handler()
#else
  channel.on_pdu()
```

on_control_frame()

```
handler(on_{event})
```

send_pdu()

```
host.send_l2cap_pdu()
```

send_control_frame()

```
host.send_l2cap_pdu()
```

LeConnectionOrientedChannel (EventEmitter)

Channel (EventEmitter)

send_pdu()

```
manager.send_pdu()
```

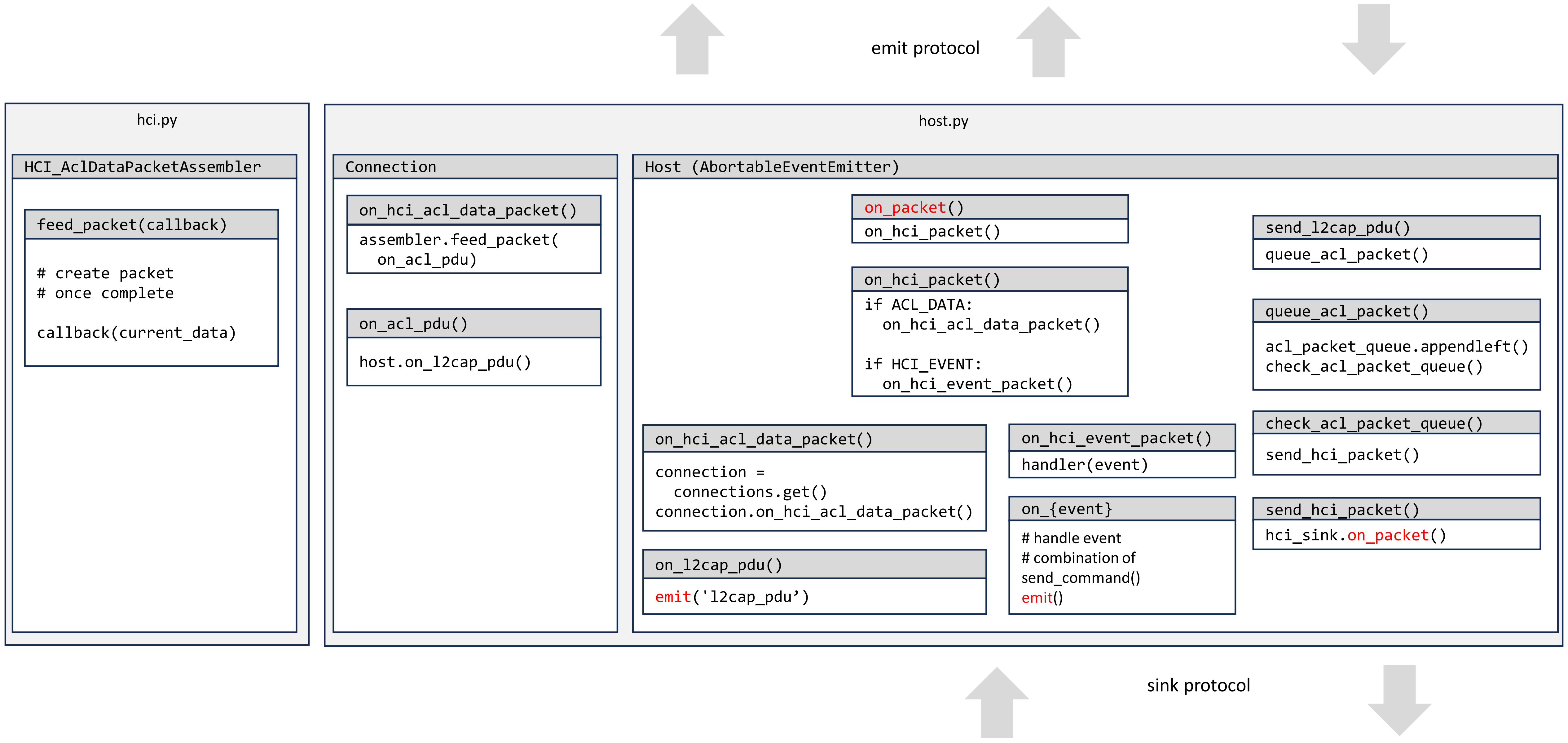
send_control_frame()

```
manager.send_control_frame()
```

on_pdu()

```
#if response
  response.set_result()
#if sink
  sink(pdu)
```

host

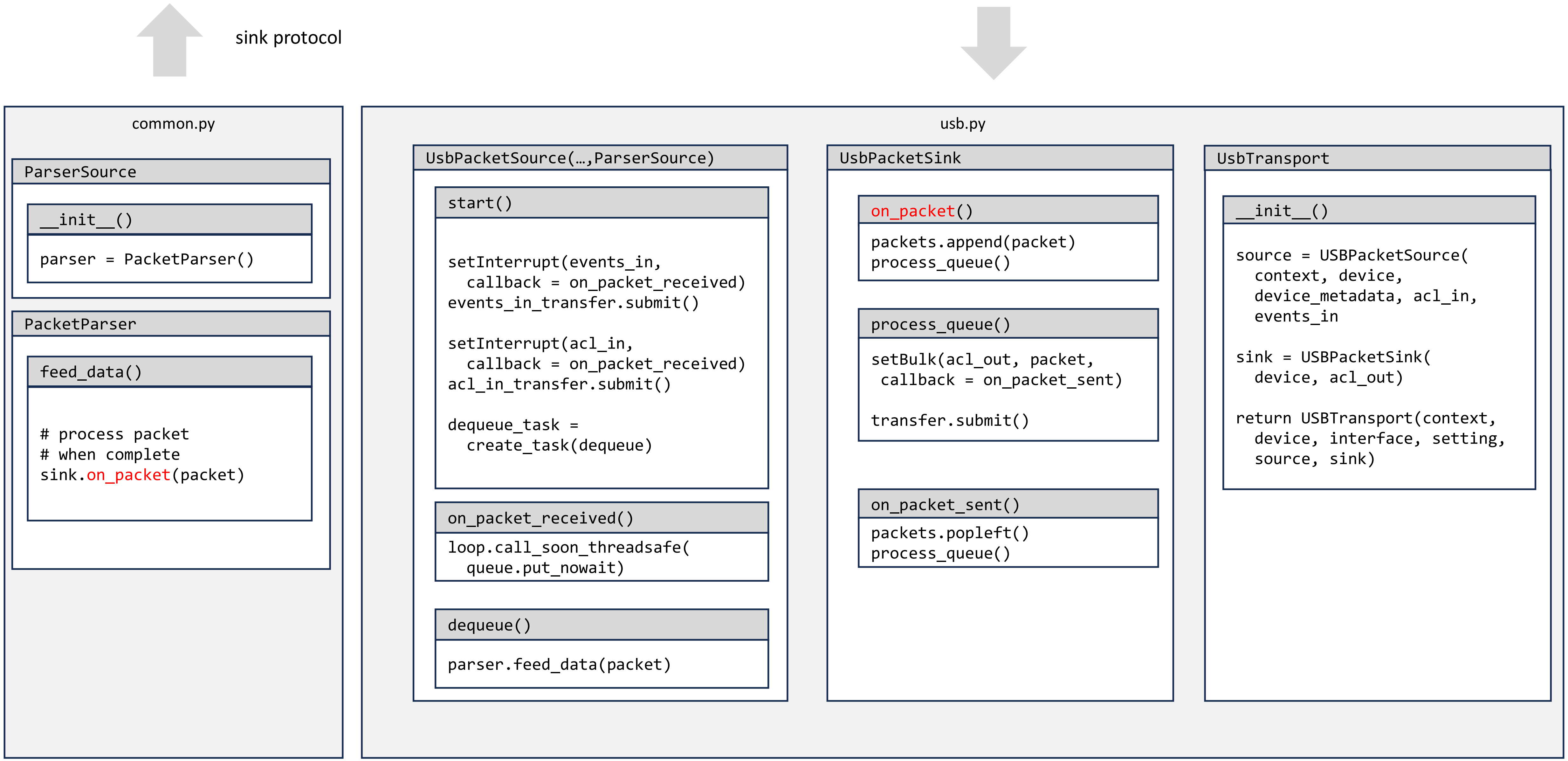


```
HCI_AclDataPacketAssembler

feed_packet(callback)

# create packet
# once complete

callback(current_data)
```

```
Connection

on_hci_acl_data_packet()
assembler.feed_packet(
  on_acl_pdu)

on_acl_pdu()
host.on_l2cap_pdu()
```

```
Host (AbortableEventEmitter)

on_packet()
on_hci_packet()

on_hci_packet()
if ACL_DATA:
  on_hci_acl_data_packet()

if HCI_EVENT:
  on_hci_event_packet()

on_hci_acl_data_packet()
connection =
  connections.get()
connection.on_hci_acl_data_packet()

on_l2cap_pdu()
emit('l2cap_pdu')

on_hci_event_packet()
handler(event)

on_{event}
# handle event
# combination of
send_command()
emit()

send_l2cap_pdu()
queue_acl_packet()

queue_acl_packet()
acl_packet_queue.appendleft()
check_acl_packet_queue()

check_acl_packet_queue()
send_hci_packet()

send_hci_packet()
hci_sink.on_packet()
```

transport/usb



common.py

ParserSource

```
__init__()
parser = PacketParser()
```

PacketParser

```
feed_data()
# process packet
# when complete
sink.on_packet(packet)
```

usb.py

UsbPacketSource(…,ParserSource)

```
start()

setInterrupt(events_in,
  callback = on_packet_received)
events_in_transfer.submit()

setInterrupt(acl_in,
  callback = on_packet_received)
acl_in_transfer.submit()

dequeue_task =
  create_task(dequeue)
```

```
on_packet_received()
loop.call_soon_threadsafe(
  queue.put_nowait)
```

```
dequeue()
parser.feed_data(packet)
```

UsbPacketSink

```
on_packet()
packets.append(packet)
process_queue()
```

```
process_queue()
setBulk(acl_out, packet,
 callback = on_packet_sent)

transfer.submit()
```

```
on_packet_sent()
packets.popleft()
process_queue()
```

UsbTransport

```
__init__()
source = USBPacketSource(
  context, device,
  device_metadata, acl_in,
  events_in

sink = USBPacketSink(
  device, acl_out)

return USBTransport(context,
  device, interface, setting,
  source, sink)
```