U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Arquitecturas da Computação Industrial

2015/2016

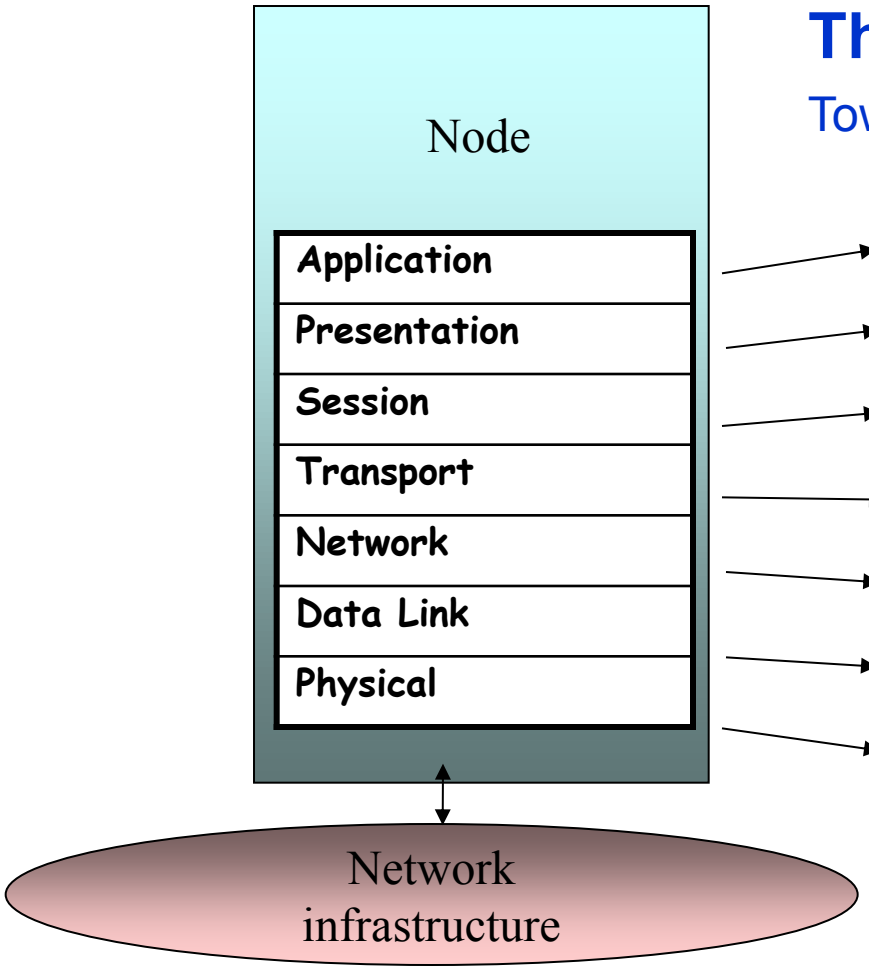
Trabalho prático 1

Pilha protocolar / Modbus TCP

Objetivos de aprendizagem

- Compreender os conceitos de camada, serviço e protocolo em redes industriais.
- Formalizar os conceitos anteriores através da implementação de uma API para desenvolver clientes e servidores Modbus TCP, usando sockets.
- Aprofundar o conhecimento do protocolo Modbus.

Protocol stack

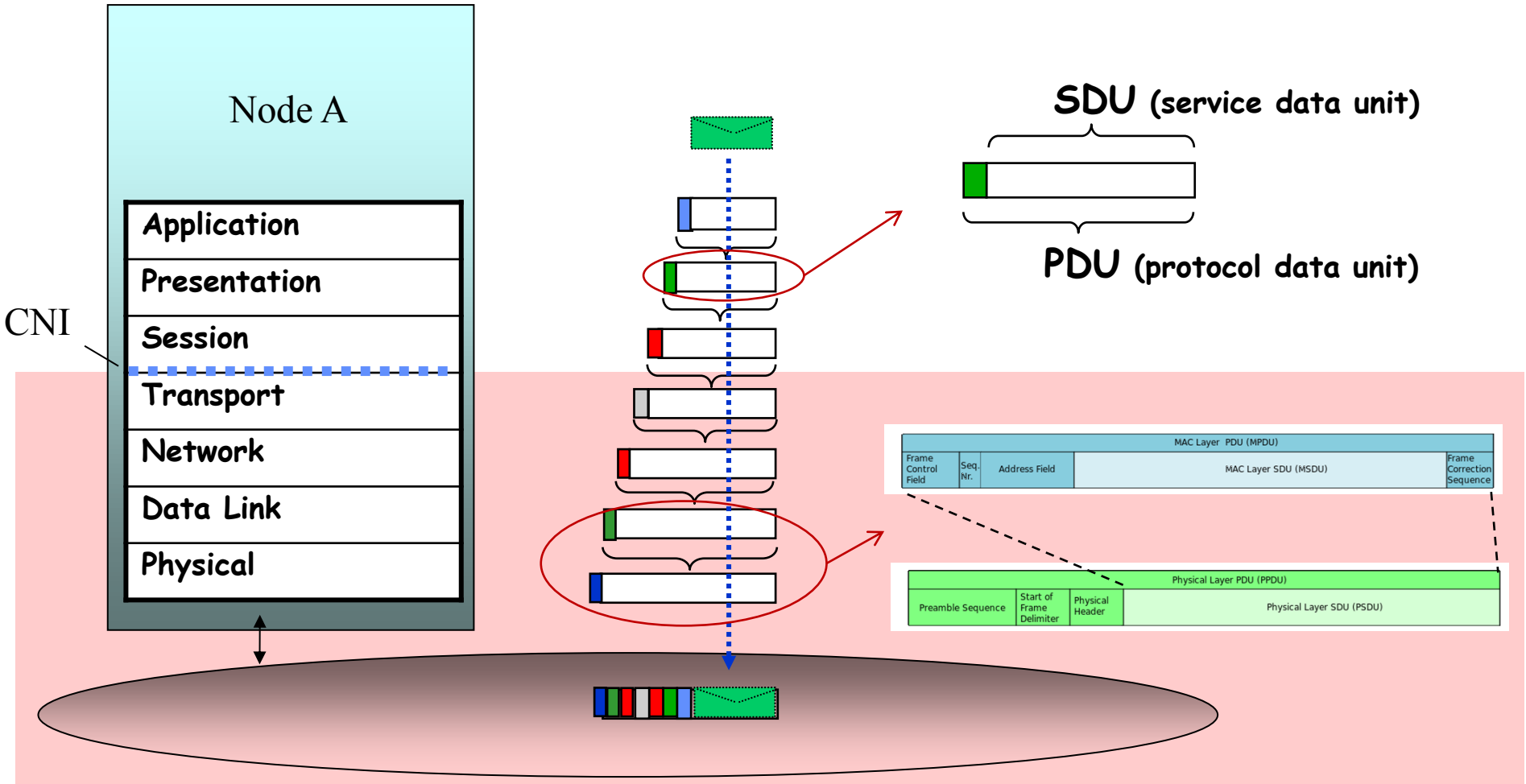


The OSI reference model

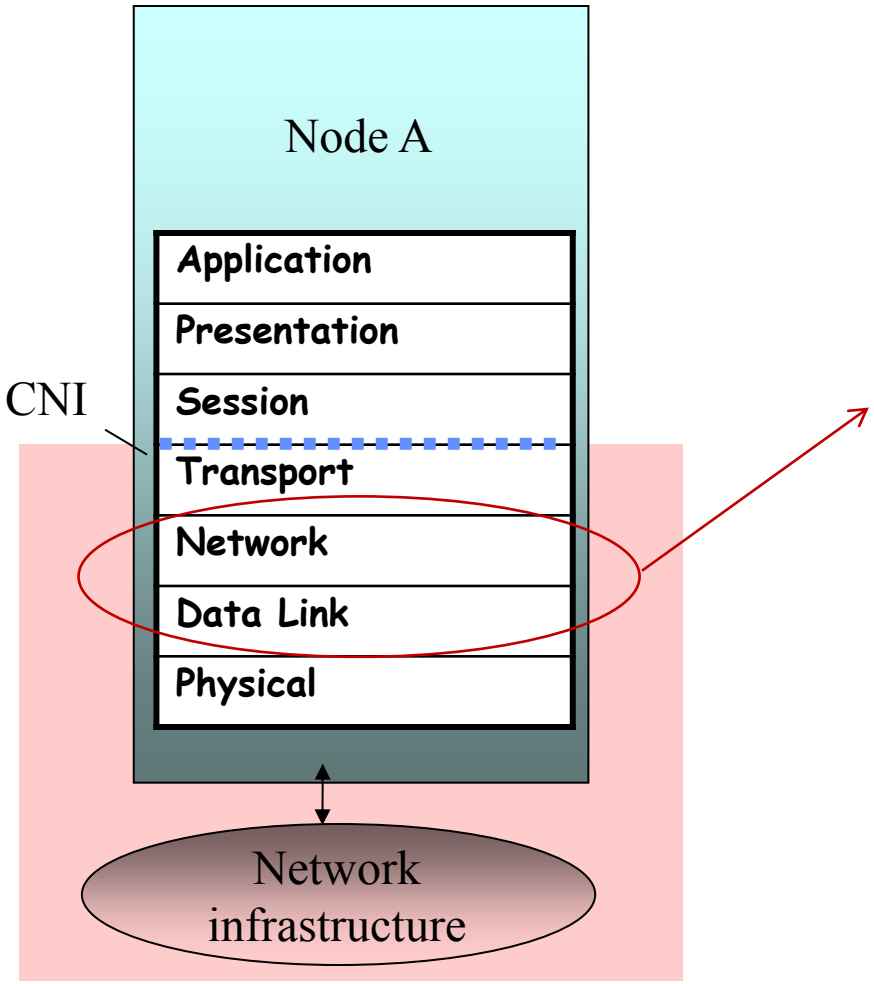
Towards interoperability and open systems

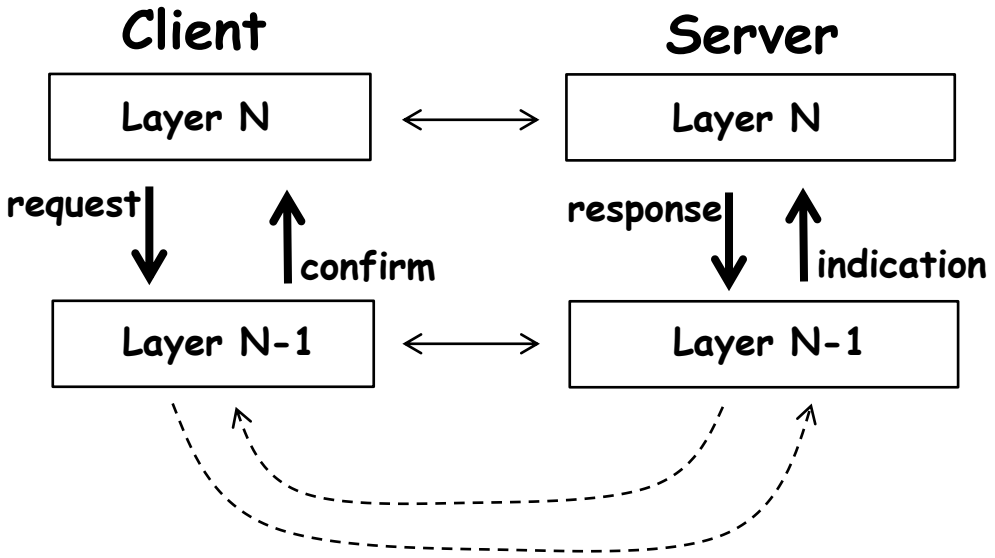
- Common services for class of apps.
- Data semantics
- Remote actions
- End-to-end comm. control
- Routing, logical addressing
- Physical addressing, MAC, LLC
- Topology, medium, bit-encoding

Protocol / Service Data Units



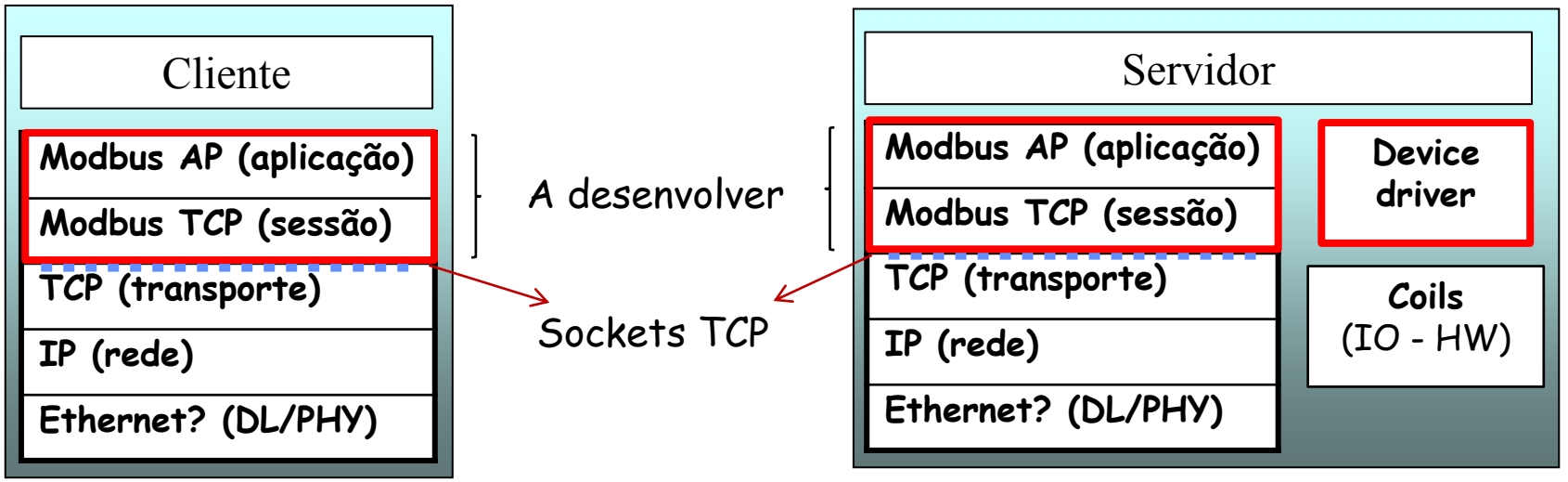
Interaction model between layers



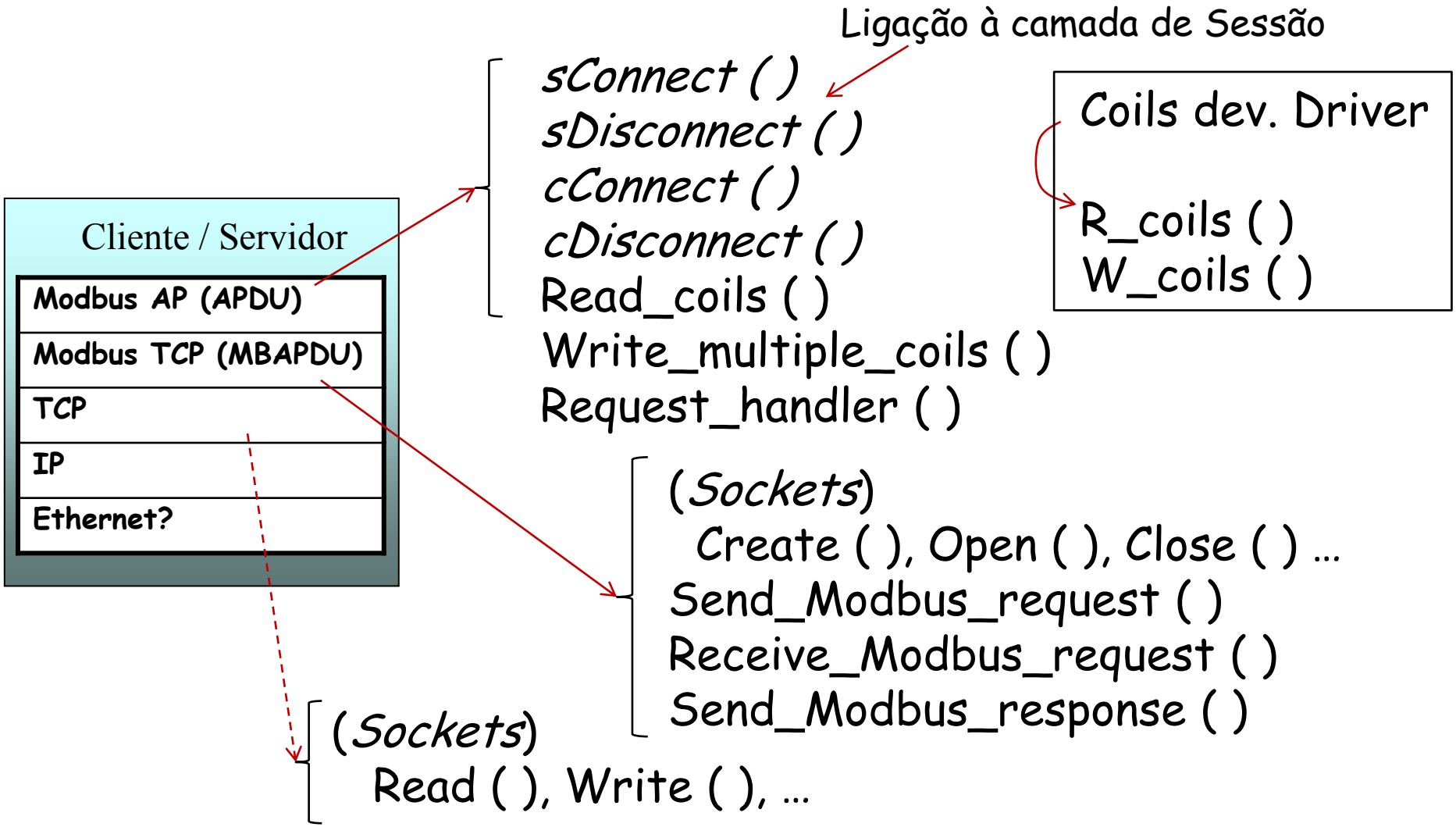


Pilha Modbus a desenvolver

- Funções a desenvolver (camada aplicação – cliente)
  - **Read coils**
  - **Write multiple coils**



Métodos da pilha protocolar Modbus TCP



Camada Modbus TCP – lado do cliente

- **Send_Modbus_request (fd, APDU, APDU_R)**
  {
  // gera TI (trans.ID →número de sequência)
  // constroi PDU = APDU(SDU) + MBAP
  write (fd, PDU) // envia Modbus TCP PDU
  read (fd, PDU_R) // resposta ou timeout
  // se resposta, remove MBAP, PDU_R → APDU_R
  // retorna: APDU_R e 0 – ok, <0 – erro (timeout)
  }

Camada Modbus AP – lado do cliente

- **Write_multiple_coils (fd, st_c, n_c, val)**

```
{
        // verifica consistência dos parâmetros
        // constroi APDU (PDU de MODBUS)
        Send_Modbus_request (fd, APDU, APDU_R)
        // analiza resposta (APDU_R ou código erro)
        // retorna:  num coils escritas – ok, <0 – erro
}
```

- **Read_coils (fd, st_c, n_c, val)**

```
{
        // verifica consistência dos parâmetros
        // constroi APDU (PDU de MODBUS)
        Send_Modbus_request (fd, APDU, APDU_R)
        // analiza resposta (APDU_R ou código erro)
        // retorna:  num coils lidas – ok, <0 – erro
}
```

Camada Modbus AP – lado do cliente

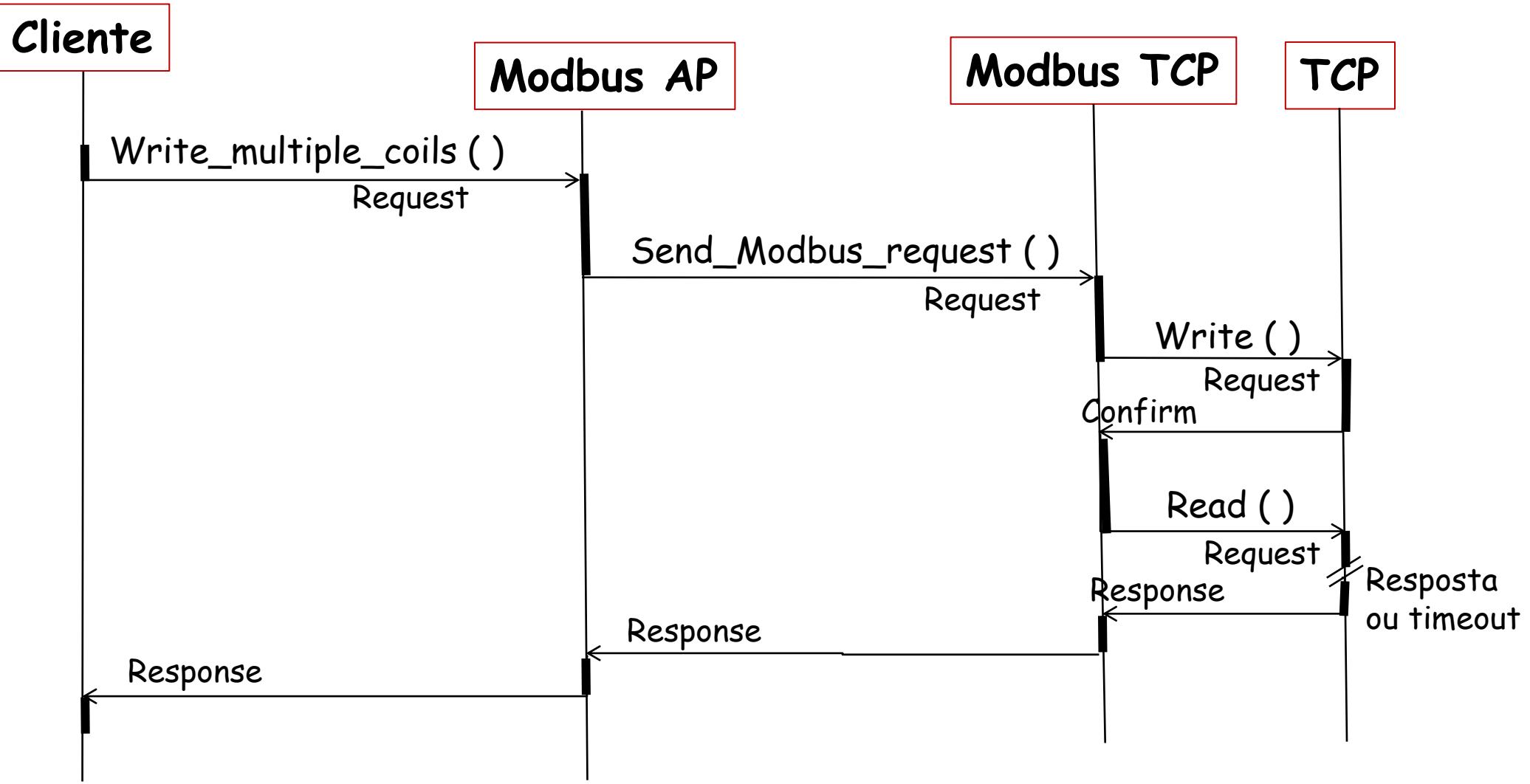
- **cConnect (server_add, port)**

```
{
        // cria socket
        // liga ao servidor
        // retorna  fd – ok,  <0 – erro
}
```

- **cDisconnect (fd)**

```
{
        // fecha / destroi socket
        // retorna  >0 – ok,  <0 – erro
}
```

Diagrama de sequência – lado do cliente



Camada Modbus TCP – lado do servidor

- **Receive_Modbus_request (fd, APDU_P, TI)**
  {
  read (fd, PDU) // lê PDU com pedido
  // extrai MBAP (PDU→APDU_P) e TI
  // retorna: APDU_P e TI – ok, <0 – erro
  }

- **Send_Modbus_response (fd, APDU_R, TI)**
  {
  // constroi PDU = APDU_R + MBAP (com TI)
  write (fd, PDU) // envia Modbus TCP PDU com resposta
  // retorna: >0 – ok, <0 – erro
  }

Camada Modbus AP – lado do servidor

- **Request_handler (fd)**
  {
  Receive_Modbus_request (fd, APDU_P, TI)
  // analiza e executa pedido se correto
  W_coils (st_c, n_c, val) ou **R_coils (st_c, n_c, val)**
  // prepara e envia APDU de resposta
  Send_Modbus_response (fd, APDU_R, TI)
  // retorna: >0 – ok, <0 – erro
  }

- **sConnect (port)**
  {
  // cria socket
  // espera conexão, aceita
  // retorna fd – ok, <0 – erro
  }

- **sDisconnect (fd)**
  {
  // fecha / destroi socket
  // retorna >0 – ok, <0 – erro
  }

IO Device Driver – lado do servidor

- **R_coils (st_c, n_c, val)**

```
{
        // lê n_c coils a partir de st_c e escreve val
        // retorna:  num coils lidas, valores em val – ok,
                              <0 – erro
}
```

- **W_coils (st_c, n_c, val)**

```
{
        // escreve n_c coils a partir de st_c e escreve val
        // retorna:  num coils escritas, valores em val – ok,
                              <0 – erro
}
```

Diagrama de sequência – lado do servidor