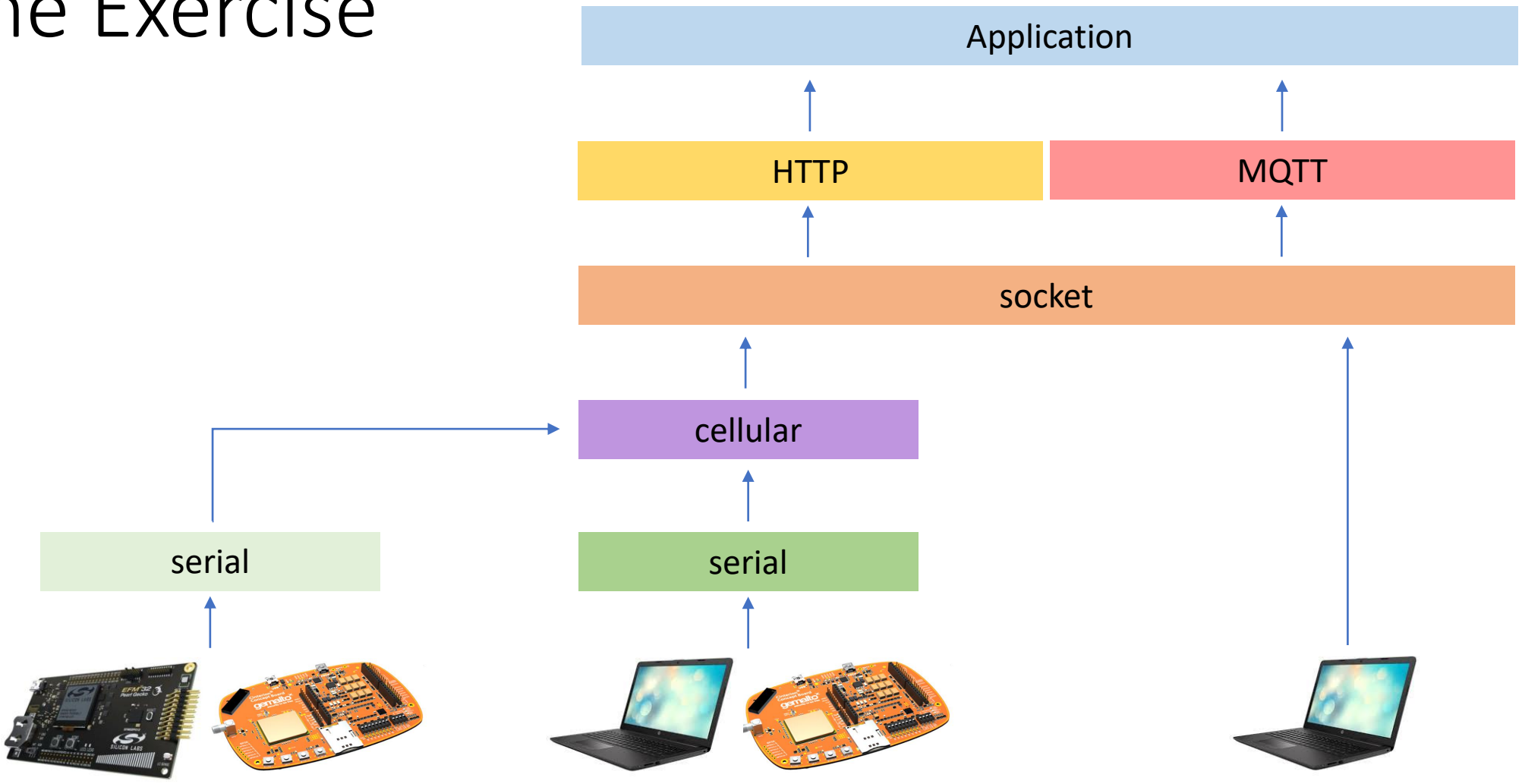
WORKSHOP ON **INTERNET OF THINGS** 67612

Exercise 5

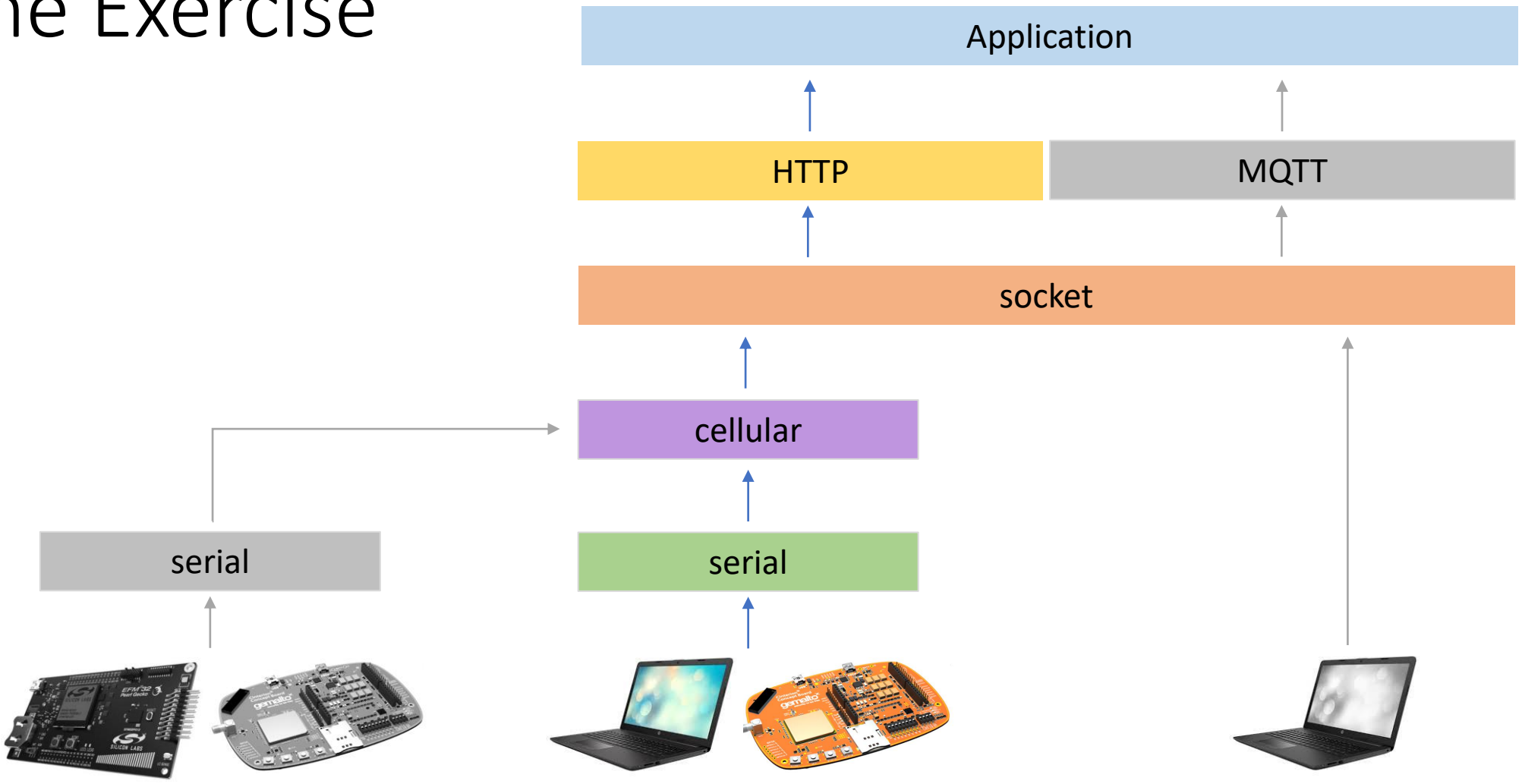
HTTP OVER CELLULAR

Prof. David Hay, Dr. Yair Poleg, Mr. Samyon Ristov

The Exercise

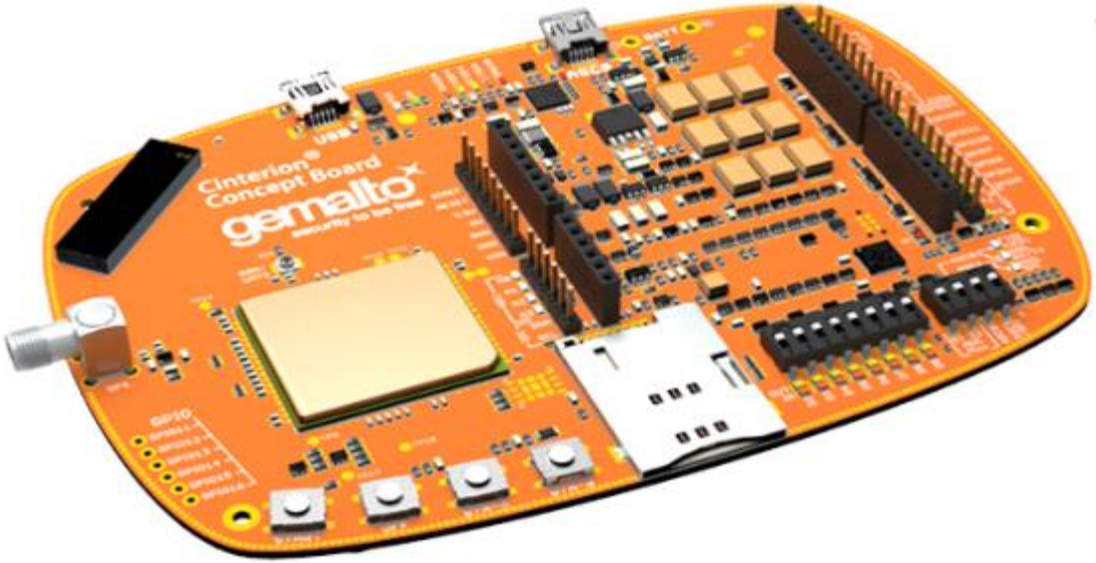


The Exercise



The Exercise

- Initiate an internet connection
- Send HTTP GET and POST requests to an HTTP server
- Print the progress of the communication



Guidance

- Follow Cinterion EHS6 AT Command Set. You can start with chapter 10, Internet Service Commands, page 217.
- Commands to pay a special attention to: AT^SICS, AT^SISS, AT^SICI, AT^SISO, AT^SISC, AT^SIST, AT^SISR, AT^SISW, AT^SISI, AT^SISE
- You may want to be able to handle URCs (especially, “^SIS” URCs. Chapter 10.14 Internet Service URC “^SIS”, page 255)
- Chapter 10.15, and especially 10.15.9 and 10.15.10 can help you with examples of transparent TCP socket

Guidance

- To monitor and debug your HTTP session use the following tool:
  - Try surfing here with a browser or curl: https://en8wtnrvtnkt5.x.pipedream.net/
  - Inspect your request here: https://requestbin.com/r/en8wtnrvtnkt5

Guidance cont’d

- An example of a request over AT interface

```
AT^SICS=0,conType,"GPRS0"
AT^SICS=0,apn,"postm2m.lu"
AT^SICS=0,inactTO,"60"
AT^SISS=1,SrvType,"Socket"
AT^SISS=1,conId,"0"
AT^SISS=1,address,"socktcp://3.223.232.168:80;etx;timer=40"
AT^SISO=1
// check connection status, with or without URCs
AT^SIST=1
// some message goes here
// use +++ to exit transparent mode
AT^SISC=1
```

Guidance cont’d

Tips:

- Pay special attention to \r (= CR = 0D) and \n (= LF = 0A)
- Some terminals send 0d instead of 0a. Putty (and most likely in “screen” as well), sends only 0a.
- In Windows, use Docklight to see the communication

Guidance cont’d

- Create a socket.h file with the following functions (same as ex1):

Guidance cont’d

```
/*
 * Initializes the socket.
 * Host: The destination address
 * as DNS: en8wtnrvtnkt5.x.pipedream.net,
 * or as IPv4: 35.169.0.97.
 * Port: The communication endpoint, int, e.g.: 80.
 * Returns 0 on success, -1 on failure
 */
int SocketInit(char *host, int port);
```

Guidance cont’d

```
/*
 * Connects to the socket
 * (establishes TCP connection to the pre-defined host and port).
 * Returns 0 on success, -1 on failure
 */
int SocketConnect(void);
```

Guidance cont’d

```
/*
 * Writes len bytes from the payload buffer
 * to the established connection.
 * Returns the number of bytes written on success, -1 on failure
 */
int SocketWrite(unsigned char *payload, unsigned int len);
```

Guidance cont’d

```
/*
 * Reads up to max_len bytes from the established connection
 * to the provided buf buffer,
 * for up to timeout_ms (doesn’t block longer than that,
 * even if not all max_len bytes were received).
 * Returns the number of bytes read on success, -1 on failure
 */
int SocketRead(unsigned char *buf, unsigned int max_len,
unsigned int timeout_ms);
```

Guidance cont’d

```
/*
 * Closes the established connection.
 * Returns 0 on success, -1 on failure
 */
int SocketClose(void);
```

Guidance cont’d

```
/*
 * Frees any resources that were allocated by SocketInit
 */
void SocketDeInit(void);
```

Guidance cont’d

- Implement these functions in a file called socket_linux_modem.c
- This file will use the cellular interface you’ve built in the previous exercises
- OK to assume a single thread, also OK to use global/static variables

Guidance cont’d

- Create a HTTP_client.h file with the following functions (same as ex1):

Guidance cont’d

```
/*
 * Initializes the client.
 * Host: The destination address
 * as DNS: en8wtnrvtnkt5.x.pipedream.net,
 * or as IPv4: 35.169.0.97.
 * Port: The communication endpoint, int, e.g.: 80.
 * Returns 0 on success, -1 on failure
 */
int HTTPClientInit(char *host, int port);
```

Guidance cont’d

```
/*
 * Writes a simple HTTP GET request to the given URL (e.g.: “/”),
 * and pre-defined host (appears in HTTP body) and port.
 * Reads up to response_max_len bytes from
 * the received response to the provided response buffer.
 * The response buffer and the provided response_max_len
 * are used only for the payload part
 * (e.g.: {"success":true} - 16 bytes) and not the entire message.
 * i.e. response like HTTP/1.1 200 OK and headers are not included
 * Returns the number of bytes read on success, -1 on failure
 */
int HTTPClientSendHTTPGetDemoRequest(char *url, char *response, int
response_max_len);
```

Guidance cont’d

```
/*
 * Writes a simple HTTP POST request to the given URL (e.g.: “/”),
 * and pre-defined host (appears in HTTP body) and port.
 * The POST request sends the provided message_len from the message buffer.
 * Reads up to response_max_len bytes from the
 * received response to the provided response buffer.
 * The response buffer and the provided response_max_len
 * are used only for the payload part
 * (e.g.: {"success":true} - 16 bytes) and not the entire message.
 * i.e. response like HTTP/1.1 200 OK and headers are not included
 * Returns the number of bytes read on success, -1 on failure.
 */
int HTTPClientSendHTTPPostDemoRequest(char *url, char *message, unsigned int
message_len, char *response, int response_max_len);
```

Guidance cont’d

```
/*
 * Closes any open connections and cleans all the defined and
allocated variables
 */
void HTTPClientDeInit(void);
```

Guidance cont’d

- Implement these functions in a file called HTTP_client.c
- In fact, you should be able just to copy your ex1 implementation and use it here
- OK to assume a single thread, also OK to use global/static variables

Guidance cont’d

- Add declaration of the following functions in “cellular.h”:

Guidance cont’d

```
/*
 * Initialize an internet connection profile (AT^SICS)
 * with inactTO=inact_time_sec and
 * conType=GPRS0 and apn="postm2m.lu". Return 0 on success,
 * and -1 on failure.
 */
int CellularSetupInternetConnectionProfile(int inact_time_sec);
```

Guidance cont’d

```
/*
 * Initialize an internal service profile (AT^SISS)
 * with keepintvl=keepintvl_sec (the timer)
 * and SrvType=Socket
 * and conId=<CellularSetupInternetConnectionProfile_id>
 * (if CellularSetupInternetConnectionProfile is already initialized.
 * Return error, -1, otherwise)
 * and Address=socktcp://IP:port;etx;time=keepintvl_sec.
 * Return 0 on success, and -1 on failure.
 */
int CellularSetupInternetServiceProfile(char *IP, int port, int
keepintvl_sec);
```

Guidance cont’d

```
/*
 * Connects to the socket (establishes TCP connection to the pre-defined host and port).
 * Returns 0 on success, -1 on failure.
 */
int CellularConnect(void);
```

Guidance cont’d

```
/*
 * Closes the established connection.
 * Returns 0 on success, -1 on failure.
 */
int CellularClose();
```

Guidance cont’d

```
/*
 * Writes len bytes from payload buffer to the established connection
 * Returns the number of bytes written on success, -1 on failure
 */
int CellularWrite(unsigned char *payload, unsigned int len);
```

Guidance cont’d

```
/*
 * Reads up to max_len bytes from the established connection
 * to the provided buf buffer, for up to timeout_ms
 * (doesn’t block longer than that, even
 * if not all max_len bytes were received).
 * Returns the number of bytes read on success, -1 on failure.
 */
int CellularRead(unsigned char *buf, unsigned int max_len, unsigned int
timeout_ms);
```

Guidance cont’d

- Create a program (main.c) that:
- Uses the implemented HTTP_client.h (that uses socket.h (that uses cellular.h (that uses serial.io.h)))
- Connects to a hardcoded host and port (as requested by main.c)
- Sends HTTP GET request and prints both the HTTP level response (in HTTP_client.c) and the application-level response (in main.c)
- Sends HTTP POST request with a requested message (hardcoded and requested by main.c), prints both the HTTP level response (in HTTP_client.c) and the application-level response (in main.c)
- Closes the connection and exits
- Prints the progress of the communication. If any error occurs, print it as well.
- No need to use args

Guidance cont’d

- You may use URCs to check socket connection status or decide to pull the status. Whatever you choose, do it explicitly and don’t assume the modem comes in that state.
- i.e., use one of these:
- `AT^SCFG="Tcp/WithURCs","off“`
- `AT^SCFG="Tcp/WithURCs","on"`

Guidance cont’d

- GET example:

```
GET / HTTP/1.1\r\n
Host: en8wtnrvtnkt5.x.pipedream.net\r\n
\r\n
```

```
1 GET · / · HTTP/1.1CRLF
2 Host: · en8wtnrvtnkt5.x.pipedream.netCRLF
3 CRLF
```

- If *URL* isn’t “/” but “/IOT/class/2021/2022”, then the request will be:

```
GET /IOT/class/2021/2022 HTTP/1.1\r\n
Host: en8wtnrvtnkt5.x.pipedream.net\r\n
\r\n
```

- Pay attention that the *host* is used both for TCP (converted to IP) and HTTP

Guidance cont’d

- Received response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Date: Mon, 19 Oct 2020 19:41:20 GMT
x-pd-status: sent to primary
X-Powered-By: Express
Content-Length: 16
Connection: keep-alive

{"success":true}
```

- `HTTPClientSendHTTPGetDemoRequest` returns only: {"success":true}
  - The provided `response_max_len` is used only for the payload part ({"success":true} – 16 bytes) and not the entire message.

Guidance cont'd

POST example:

```
POST / HTTP/1.1\r\n
Host: en8wtnrvtnkt5.x.pipedream.net:80\r\n
Content-Type: text/plain\r\n
User-Agent: GemaltoModem\r\n
Cache-Control: no-cache\r\n
Content-Length: 21\r\n
Connection: keep-alive\r\n
\r\n
hello cellular world!
```

No \r\n at the end!

```
1  POST · / · HTTP/1.1CRLF
2  Host: · en8wtnrvtnkt5.x.pipedream.net:80CRLF
3  Content-Type: · text/plainCRLF
4  User-Agent: · GemaltoModemCRLF
5  Cache-Control: · no-cacheCRLF
6  Content-Length: · 21CRLF
7  Connection: · keep-aliveCRLF
8  CRLF
9  hello · cellular · world!
10
```

Guidance cont’d

- By using `HTTPClientSendHTTPPostDemoRequest` the user controls:
  - URL of the POST request (e.g.: “`/`”)
  - host and port (e.g.: “`en8wtnrvtnkt5.x.pipedream.net:80`”)
  - `Content-Length` (e.g.: `21`)
  - Message: `hello cellular world!`
- The user doesn’t control any other parameters, like HTTP version and headers (which are hard-coded in your code).

Exercise #5

- Work & submit in pairs
- Make sure that your submissions works on the VM (try importing it as a new project)
- Deliverables:
  - Provide all the project files, and/or export the project
  - Create makefile or CMakeLists.txt (in CLion).
  - A README file with your names, email addresses, IDs and adequate level of documentation of the deliverables and software design-architecture-flow description
  - If anything special is needed (compilation instructions and environment requirements), add it to the README
- Pack all the deliverables as .zip or .tar and upload to Moodle
- Deadline: 23.11.2021, 23:59pm
- The grade will be based on code's functionality, description, and clear implementation

Contact

- Moodle's 'Workshop Discussions' forum is the best place for questions.

- But if needed, contact us personally:
- David Hay – dhay@cs.huji.ac.il
- Yair Poleg – yair.poleg@mail.huji.ac.il
- Samyon Ristov – samyon.ristov@mail.huji.ac.il