INTERNET OF THINGS
DEPARTMENT OF COMPUTER ENGINEERING
SANTA CLARA UNIVERSITY, CALIFORNIA
BEHNAM DEZFOULI, PHD

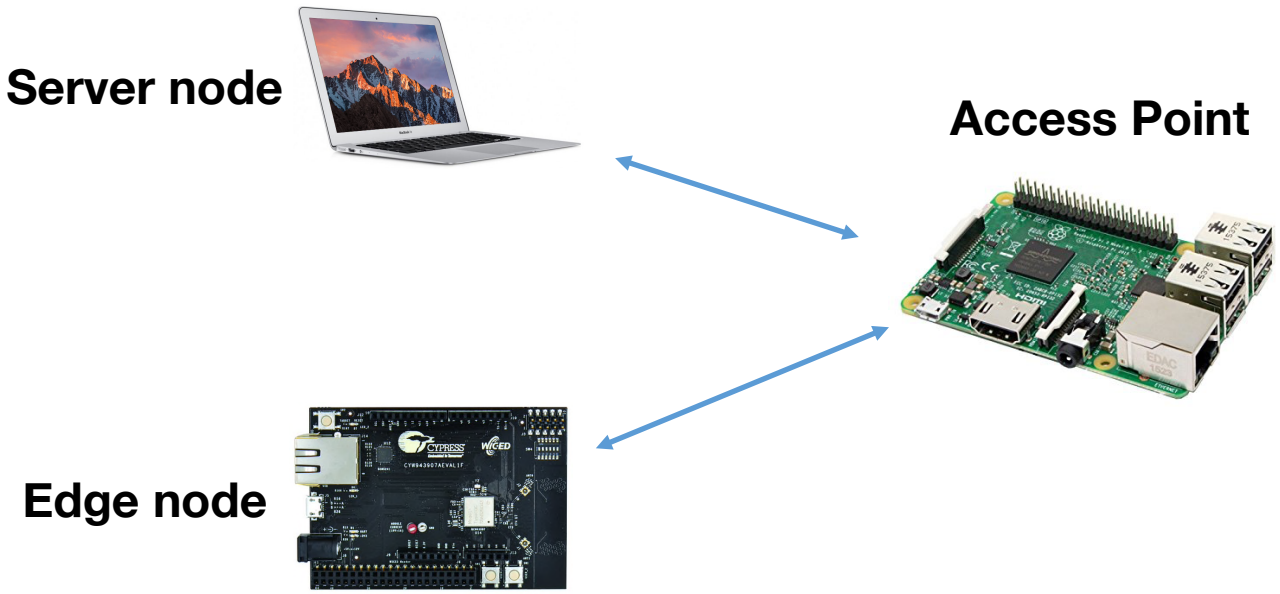
Santa Clara University
Internet of Things
RESEARCH LAB
SIOTLAB

Internet of Things

Lab 9
Application Layer Programming

Outline

- We plan to:
  - Write a **server code on your laptop** that is connected to the Raspberry Pi's access point
  - Write a **client code on the edge node** that is connected to the Raspberry Pi's access point
  - **Exchange messages** between the client and server



Server: Python Socket Programming

Python Socket Programming

- **Client - Server Model**
  In a client-server model of computing, a **server** hosts a resource or a service which is accessed by **clients**
- A **server** creates a socket in its communication end and binds itself to a port and then listens (waits) for incoming connections from **client**
- A client creates a socket in its end and connects to the listening server at the specified port

A Python Server Program

- A server has a **`bind()`** method which binds it to a specific IP and port so that it can **listen to incoming requests** on that IP and port

- A server has a **`listen()`** method which puts the server into listen mode
  - This allows the server to listen to incoming connections

- A server has an **`accept()`** and **`close()`** method
  - The accept method initiates a connection with the client
  - The close method closes the connection with the client

Python Socket Programming

A Python Server Program

```
# import the socket library
import socket

# create a socket object
s = socket.socket()
print "Socket successfully created"

# reserve a port on your computer
# it can be anything as long it is not a reserved port
port = 7000


# Next bind to the port
s.bind(('', port))
print "socket binded to %s" %(port)


# put the socket into listening mode
s.listen(5)
print "socket is listening"
```

- We bind the server to the specified port
- Passing an empty string means that the server can listen to incoming connections from other computers as well
- If we would have passed 127.0.0.1 then it would have listened to only those calls made within the local computer

- We put the server into listen mode
- 5 here means that if a 6th socket tries to connect then the connection is refused

Python Socket Programming

A Python Server Program

```
# a forever loop until we interrupt it or an error occurs
while True:
# Establish connection with client.
        c, addr = s.accept()
        print 'Got connection from', addr
        # send a thank you message to the client.
        c.send('Thank you for connecting')

        # Close the connection with the client
        c.close()
```

Client: RTOS Socket Programming
(Using Streams)

RTOS Socket Programming

- **“raw” sockets inherently don’t have security**
- The TCP socket just sends whatever data it was given over the link
- It is the responsibility of a layer above TCP such as SSL or TLS to encrypt/decrypt the data if security is being used (which we will cover later on)

Programming Edge Devices

To setup the **TCP client connection**, the client firmware will:

- **Create** the TCP socket by calling:
  - `wiced_tcp_create_socket( &socket, WICED_STA_INTERFACE );`
- **Bind** to TCP port 7001 by calling:
  - `wiced_tcp_bind( &socket, 7001);`
  - We may use `WICED_ANY_PORT` if the port number is not important
- **Connect** to port 7000 through the network
  - `wiced_tcp_connect( &socket, &serverAddress, 7000, TIMEOUT);`
  - `serverAddress` is the server's IP address
  - In our local network the timeout (in milliseconds) can be small <1s
  - In a WAN situation the timeout may need to be extended to as long as a few seconds

Programming Edge Devices

To find the server address:

- IP address is a WICED data structure of type `wiced_ip_address_t`

- You can initialize the structure in one of two ways

  - **Static** method: use the macros provided by the WICED SDK as follows:
    - `SET_IPV4_ADDRESS( serverAddress, MAKE_IPV4_ADDRESS(198, 51, 100, 3 ) );`

  - **DNS**: To initialize it by performing a DNS loop
    `wiced_hostname_lookup( "server_name", &serverAddress, 10000 );`

Programming Edge Devices

- Once the connection has been created, your application will want to transfer data between the client and server
- The simplest way to transfer data over TCP is **to use the stream functions** from the SDK
- The stream functions allow you to send and receive **arbitrary amounts of data without worrying about the details of packetizing data** into uniform packets

- To use a stream you must first declare a stream structure and then initialize that with the socket for your network connection:
  - `wiced_tcp_stream_t stream;`
  - `wiced_tcp_stream_init(&stream, &socket);`
- Once this is done it is simple to write data using the `wiced_tcp_stream_write()` function
  - This function takes the stream and message as parameters
  - The message is just an array of characters to send
- When you are done writing to the stream, you need to call the `wiced_tcp_stream_flush()` method

Programming Edge Devices

❑ Example: The following code demonstrates writing a single message:

```
➢ char sendMessage[] = "TEST_MESSAGE";
➢ wiced_tcp_stream_write(&stream, sendMessage, strlen(sendMessage));
➢ wiced_tcp_stream_flush(&stream);
```

- **Reading data from the stream** uses the `wiced_tcp_stream_read()` function
- This method takes a stream and a message buffer as parameters
- The function also requires you to specify the maximum number of bytes to read into the buffer and a timeout
- The function returns a `wiced_result_t` value which can be used to ensure that reading the stream succeeded
- See the next slide…

Programming Edge Devices

Read data from a TCP stream

```
 *
 * @param[in,out] tcp_stream    : A pointer to a stream handle where data will
                                                                 be written
 * @param[out]    buffer        : The memory buffer to write data into
 * @param[in]     buffer_length : The number of bytes to read into the buffer
 * @param[in]     timeout       : Timeout value in milliseconds or
                                                      WICED_NEVER_TIMEOUT
 *
 * @return @ref wiced_result_t
 */


wiced_result_t wiced_tcp_stream_read( wiced_tcp_stream_t* tcp_stream,
                      void* buffer, uint16_t buffer_length, uint32_t timeout );
```

Programming Edge Devices

- Example: Putting all together…

```
#define SERVER_PORT (7000)
#define TIMEOUT (2000)
.
.
wiced_tcp_socket_t socket;
wiced_tcp_stream_t stream;
char sendMessage[]="A Message from Behnam's Cypress Board!";
.
.
wiced_tcp_create_socket(&socket, WICED_STA_INTERFACE);
wiced_tcp_bind(&socket, WICED_ANY_PORT );


wiced_tcp_connect(&socket, &serverAddress, SERVER_PORT, TIMEOUT);
wiced_tcp_stream_init(&stream, &socket);
wiced_tcp_stream_write(&stream, sendMessage, strlen(sendMessage);
wiced_tcp_stream_flush(&stream);


wiced_tcp_stream_deinit(&stream);
wiced_socket_delete(&socket);
```

Programming Edge Devices

- Behind the scenes, reading and writing via streams uses uniform sized packets
- The stream functions in the SDK hides the management of each of these packets from you so you can focus on the higher levels of your application
- However, if you desire more control over the communication, **you can use the WICED SDK API to send and receive packets directly**

RTOS Socket Programming

Example: **socket program**
C Code: Edge Side –
(`client-stream`)
Part 1/7

```
#include "wiced.h"
#include "register_map.h"

#define TCP_CLIENT_STACK_SIZE      (6200)
#define SERVER_PORT                (7000)


static wiced_ip_address_t serverAddress; //Server IP address
static wiced_semaphore_t button0_semaphore, button1_semaphore;
static wiced_thread_t buttonUpdate;
static wiced_mac_t myMac; //Edge device's (Cypress board) MAC address



void button_isr1(void *arg)
{
    wiced_rtos_set_semaphore(&button1_semaphore);
}


void button_isr0(void *arg)
{
    wiced_rtos_set_semaphore(&button0_semaphore);
}
```

Button 1 **sends status** update to the server

Button 0 **requests status** from the server

RTOS Socket Programming

Example: **socket program**
C Code: Edge Side – (client-stream)
Part 2/7

```
void sendData(char type, char* led, uint8_t status)
{
    wiced_tcp_socket_t socket;     // The TCP socket
    wiced_tcp_stream_t stream;     // The TCP stream
    char sendMessage[20];
    wiced_result_t result, conStatus;

    // format the data
    if (type == 'W')
        sprintf(sendMessage,"WRITE-%02X%02X%02X%02X%02X%02X-%01X\n",
                myMac.octet[0], myMac.octet[1], myMac.octet[2],
                myMac.octet[3], myMac.octet[4], myMac.octet[5],
                status);
    else{
        WPRINT_APP_INFO(("Invalid command type\n"));
        return;
    }

    WPRINT_APP_INFO(("Prepared Message = %s\n",sendMessage));

    // Open the connection to the remote server via a socket
    wiced_tcp_create_socket(&socket, WICED_STA_INTERFACE);
    wiced_tcp_bind(&socket,WICED_ANY_PORT);
    conStatus = wiced_tcp_connect(&socket,&serverAddress, SERVER_PORT, 2000);
    // 2 second timeout
```

Preparing a server update message
'W' means this is a write message

Establish connection to the server

RTOS Socket Programming

Example: **socket program**
C Code: Edge Side – (client-stream)
Part 3/7

```
    if(conStatus == WICED_SUCCESS)
        WPRINT_APP_INFO(("Successful connection!\n"));
    else {
        WPRINT_APP_INFO(("Failed connection!\n"));
        wiced_tcp_delete_socket(&socket);
        return;
    }

    wiced_tcp_stream_init(&stream, &socket);

    wiced_tcp_stream_write(&stream, sendMessage, strlen(sendMessage));
    wiced_tcp_stream_flush(&stream);

    // Get the response back from the server
    char rbuffer[30] = {0};
    // The first 27 bytes of the buffer will be sent by the server.
    // Byte 28 will stay 0 to null terminate the string
    uint32_t read_count;
    result = wiced_tcp_stream_read_with_count( &stream, rbuffer, 28, 500,
                                               &read_count );
    if(result == WICED_SUCCESS)
        WPRINT_APP_INFO(("Server Response = %s\n\n\n", rbuffer));
    else
        WPRINT_APP_INFO(("Malformed response = %s\n\n\n", rbuffer));

    wiced_tcp_stream_deinit(&stream);
    wiced_tcp_delete_socket(&socket);
}
```

Initialize the TCP stream

Send the data via the stream

Force the data to be sent right away even if the packet isn't full yet

Delete the stream and socket

Example: **socket program**
C Code: Edge Side – (`client-stream`)
Part 4/7

```
// buttonThreadMain:
// This function is the thread that waits for button presses and then sends the
// data via the sendData function
//
// This is done as a separate thread to make the code easier to copy to a
// later program.

void buttonUpdateMain()
{
    // Main Loop: wait for semaphore.. then send the data
    while(1)
    {
        wiced_rtos_get_semaphore(&button1_semaphore,WICED_WAIT_FOREVER);
        wiced_gpio_output_low( WICED_SH_LED1 );
        sendData('W', "LED 1", 0);

        wiced_rtos_get_semaphore(&button1_semaphore,WICED_WAIT_FOREVER);
        sendData('W', "LED 1", 1);
        wiced_gpio_output_high( WICED_SH_LED1 );
    }
}
```

RTOS Socket Programming

Example: **socket program**
C Code: Edge Side – (`client-stream`)
Part 5/7

```
void application_start(void)
{
    wiced_init( )
    wiced_network_up( WICED_STA_INTERFACE,
                      WICED_USE_EXTERNAL_DHCP_SERVER, NULL );

    wiced_result_t result;

    wwd_wifi_get_mac_address(&myMac, WICED_STA_INTERFACE );

    // Use DNS to find the address..
    // if you can't look it up after 5 seconds then hard code it.

    WPRINT_APP_INFO(("DNS Lookup iot server\n"));
    result = wiced_hostname_lookup( "name_of_your_server",
                      &serverAddress, 5000, WICED_STA_INTERFACE );

    if ( result == WICED_ERROR || serverAddress.ip.v4 == 0 )
    {
        WPRINT_APP_INFO(("Error in resolving DNS using hard coded address\n"));

// Replace this with the IP address of the server
        // running on your machine
        SET_IPV4_ADDRESS( serverAddress, MAKE_IPV4_ADDRESS( 10, 16, 230, 20 ) );
    }
```

Note: Replace this with the name of your server

Note: Replace this with the IP address of your server

RTOS Socket Programming

Example: **socket program**
C Code: Edge Side – (`client-stream`)
Part 6/7

```
WPRINT_APP_INFO(("server's IP : %u.%u.%u.%u\n\n",
(uint8_t)(GET_IPV4_ADDRESS(serverAddress) >> 24),
            (uint8_t)(GET_IPV4_ADDRESS(serverAddress) >> 16),
            (uint8_t)(GET_IPV4_ADDRESS(serverAddress) >> 8),
            (uint8_t)(GET_IPV4_ADDRESS(serverAddress) >> 0)));

    WPRINT_APP_INFO(("MY MAC Address: "));
    WPRINT_APP_INFO(("%X:%X:%X:%X:%X:%X\r\n",
            myMac.octet[0], myMac.octet[1], myMac.octet[2],
            myMac.octet[3], myMac.octet[4], myMac.octet[5]));


    // Setup the Semaphore and Button Interrupt
    // the semaphore unlocks when the user presses the button
    wiced_rtos_init_semaphore(&button0_semaphore);


    // the semaphore unlocks when the user presses the button
    wiced_rtos_init_semaphore(&button1_semaphore);
```

Example: **socket program**
C Code: Edge Side – (`client-stream`)
Part 7/7

```
    // call the ISR when the button is pressed
    wiced_gpio_input_irq_enable(WICED_SH_MB0,
                  IRQ_TRIGGER_FALLING_EDGE, button_isr0, NULL);

    wiced_gpio_input_irq_enable(WICED_SH_MB1,
                  IRQ_TRIGGER_FALLING_EDGE, button_isr1, NULL);


    wiced_rtos_create_thread(&buttonUpdate, WICED_DEFAULT_LIBRARY_PRIORITY,
         "Button Update", buttonUpdateMain, TCP_CLIENT_STACK_SIZE, 0);


    WPRINT_APP_INFO(("Activated button threads..."));
}
```

Deliverables

- **Task 9-1. Develop a system as follows:**
  - **The format of commands sent from your laptop to the Cypress board is "*x*-*y*", where *x* is the shield's LED number and *y* is the LED value**
  - **For example, if you send "0-1" to the Cypress board, then LED 0 is turned on, and "0-0" turns off the LED**

Appendix 1 (Optional)
Client: RTOS Socket Programming (Using Packets)

Programming Edge Devices

- When a connection is established, instead of transmitting data through **streams**, we can send data using **packets**
- **Note:**
  - Using streams, we just write to the stream without worrying about packet size, packet preparation, and deletion
  - Using packets, we must be careful about the size of data written to the packet, and the buffer must be released after the packet is sent
- At the beginning of your application, when you run the `wiced_init()` function, on the console you will see the message "`Creating Packet pools`"
- The packet pools are just RAM buffers which store either incoming packets from the network (i.e. receive packets) or will hold outgoing packets which have not yet been sent (i.e. transmit packets)
- **By default, there are two receive packets and two transmit packets, but this can be configured in your firmware**
- If you run out of receive packets then TCP packets will be tossed
- If you run out of transmit packets you will get an error when you try to create one

Programming Edge Devices

- Each packet in the buffer contains:
  - An allocation reference count
  - The raw data
  - A pointer to the start of the data
  - A pointer to the end of the data
  - The TCP packet overhead

Packet Buffer				
Type	Ref Count	Data Pointer		Buffer
		Start	End	
R	0	null	null	
R	0	null	null	
T	0	null	null	
T	0	null	null	

- A packet starts its life unallocated, and as such, the reference count is 0

Programming Edge Devices

- When you want to send a message, you call `wiced_tcp_packet_create()` which has the prototype of:

```
wiced_result_t wiced_packet_create_tcp(
        wiced_tcp_socket_t* socket, uint16_t content_length,
                wiced_packet_t** packet, uint8_t** data,
                        uint16_t* available_space );
```

- **This function will look for an unallocated packet (i.e., the reference count == 0) and assigns it to you**

- `socket`: A pointer to the socket that was previously created by `wiced_tcp_connect()`
- `content_length`: How many bytes of data you plan to put in the packet

Programming Edge Devices

```
wiced_result_t wiced_packet_create_tcp(
        wiced_tcp_socket_t* socket, uint16_t content_length,
                wiced_packet_t** packet, uint8_t** data,
                        uint16_t* available_space );
```

- `packet`: a pointer to a packet pointer
  - This enables the create function to give you a pointer to the packet structure in the RAM
  - To use it, you declare: `wiced_packet_t *myPacket;` Then when you call the `wiced_packet_create_tcp()` you pass a pointer to your pointer e.g. `&myPacket`
  - When the function returns, `myPacket` will then point to the allocated packet in the packet pool

Programming Edge Devices

```
wiced_result_t wiced_packet_create_tcp(
        wiced_tcp_socket_t* socket, uint16_t content_length,
                wiced_packet_t** packet, uint8_t** data,
                        uint16_t* available_space );
```

- `data`: a pointer to a `uint8_t` pointer
  - To use it, you declare: `uint8 *myData;` then when you call the `wiced_packet_create_tcp()` you pass a pointer to your pointer e.g. `&myData`
  - When the function returns, **`myData` pointer will then point to the place inside of the packet buffer where you need to store your data**

- `available_space`: This is a pointer to an integer that will be set to **the maximum amount of data that you are allowed to store inside of the packet**
  - It works like the previous two in that the function changes the instance of your integer

Programming Edge Devices

- Once you have created the packet, you need to:
  - **Copy your data** into the packet in the correct place i.e. using `memcpy()` to copy to the data location that was provided to you
  - **Tell the packet where the end of your data is** by calling `wiced_packet_set_data_end()`
  - **Send the data** by calling `wiced_tcp_send_packet()`
    - This function will increment the reference count (so it will be 2 after calling this function)
  - **Release control of the packet** by calling `wiced_packet_delete()`
    - This function will decrement the reference count
  - Once the packet is actually sent by the TCP/IP stack, it will decrement the reference count again, which will make the packet buffer available for reuse

Programming Edge Devices

- After the call to `wiced_tcp_packet_create_tcp`:
  - The pointer `myPacket` points to the packet in the packet pool that is allocated to you
  - `available_space` will be set to the maximum number of bytes that you can store in the packet (about 1500)
    - You should make sure that you don't copy more into the packet than it can hold
  - The pointer `data` will point to the place where you need to copy your message

RTOS Socket Programming

- Here is how we need to change the `sendData()` method

```
wiced_packet_t* tx_packet; //Pointer to the allocated packet
uint8_t *tx_data; //Pointer to the payload of the packet
uint16_t available_data_length; //How much data you can insert

wiced_packet_create_tcp(&socket, strlen(sendMessage),
        &tx_packet, (uint8_t**)&tx_data, &available_data_length);

memcpy(tx_data, sendMessage, strlen(sendMessage));

wiced_packet_set_data_end(tx_packet,
                (uint8_t*)&tx_data[strlen(sendMessage)]);

wiced_tcp_send_packet(&socket, tx_packet);

wiced_packet_delete(tx_packet);
```

Allocate a TCP packet from the pool

Put data in the packet

Set the size of data in a packet
If data has been added to a packet, this function should be called to ensure the packet length is updated

Send TCP data packet

Releases a packet that is in use, back to the main packet pool, allowing re-use

Programming Edge Devices

- To receive data we use:

```
wiced_tcp_receive( wiced_tcp_socket_t* socket,
      wiced_packet_t** packet, uint32_t timeout )
```

- The `wiced_packet_t` ** packet means that you need to give it a pointer of type `wiced_packet_t` so that the receive function can set your pointer to point to the TCP packet in the packet pool

- This function will also increment the reference count of that packet
- When you are done, you need to delete the packet by calling `wiced_packet_delete`

Programming Edge Devices

- Finally, you can get the actual TCP packet data by calling *wiced_packet_get_data* which has the following prototype:

```
wiced_result_t wiced_packet_get_data( wiced_packet_t* packet,
       uint16_t offset, uint8_t** data,
                uint16_t* fragment_available_data_length,
                      uint16_t *total_available_data_length )
```

- This function is designed to let you grab pieces of the packet, hence the offset parameter
- To get your data you need to pass a pointer to a uint8_t pointer
- The function will update your pointer to point to the raw data in the buffer

RTOS Socket Programming

- Here is how we need to change the `sendData()` method

```
wiced_packet_t *rx_packet;


result = wiced_tcp_receive(&socket, &rx_packet, 500);

if(result == WICED_SUCCESS)
{
    char *rbuffer;
    uint16_t request_length;


    wiced_packet_get_data( rx_packet, 0, (uint8_t**) &rbuffer,
                           &request_length, &available_data_length );

    if(available_data_length < 30)
    {
         rbuffer[available_data_length]=0;
         WPRINT_APP_INFO(("Server Response=%s\n",rbuffer));
    }
    else
        WPRINT_APP_INFO(("Malformed response\n"));

        wiced_packet_delete(rx_packet);
    }
```

Attempts to receive a TCP data packet from the remote host
If a packet is returned successfully, then ownership of it has been transferred to the caller, and it must be released as soon as it is no longer needed

Retrieves a data buffer pointer for a given packet handle at a particular offset