Electronics and
Computer Science

UNIVERSITY OF
Southampton

ELEC3227 Embedded Networked Systems

Coursework 2019/20

Assignment Set:	Friday 04th October 2019, during lecture
Assignment Type:	Individual coursework, but working in teams
Submission Deadline(s):	Thurs 07th Nov 2019, 10:00-13:00 (*design review meeting*) Thurs 12th Dec 2019, 10:00-13:00 (*practical demonstration*) Mon 06th Jan 2020, by 4pm (*written report*)
Feedback:	You will receive oral feedback on your design at the Design Review Meeting and Practical Demonstration sessions. There will also be a Q&A drop-in session. You will receive written feedback and a mark within 3 weeks of the submission deadline.
Mark Contribution:	This assignment is worth 40% of your mark for ELEC3227
Required Effort:	You should expect to spend up to 60 hours on this coursework
Examiners:	Professor Geoff Merrett and Dr Alex Weddell

Learning Outcomes

Having successfully completed this coursework, you will be able to:

- demonstrate knowledge and understanding of layered networking models
- interpret standardisation documents
- communicate your technical work
- design, implement and test embedded networking protocols on practical hardware

Individual or Team?

This coursework is an individual piece of coursework, but you will be working in *teams*. If you want to pick who else is in your team (of three people), all 3 of you should separately email gvm@ecs.soton.ac.uk with the subject "ELEC3227 Team Request", listing full names of the 3 people in the team by 4pm on Fri 11th Oct. Teams will fit into the rough structure below:

Team A1	Team A2	Team C1	Team C2
Team B1	**Team B2**	**Team D1**	**Team D2**

Some **important** definitions:

- Individuals that are in the same *team* (e.g. everyone in Team A1) are *teammates*.
- *Teams* that share the same letter (e.g. A1, A2 and A3) are *peer-teams*.
- Individuals working on the same layer(s) in *peer-teams* are *peer-teammates*.

Your Task

The Scenario

Your *team's* task is to design and implement a full, working, network architecture capable of supporting a smart lighting application(s), for example like Philips Hue lighting. You must use the Il Matto and RFM12B hardware. Your architecture should support multiple 'inputs' (e.g. light switches or motion sensors) and multiple outputs (e.g. lights) in the network, where each input can control one or more outputs. The architecture should support a distributed application, rather than requiring a centralised 'hub'.

Figure 1 shows an **example** of this, where there are three devices (or network 'nodes'): one with 1 switch and 1 light, one with 2 lights, and one with 1 switch and 3 lights. Different presses of the switches cause the lights on other nodes to change to different states.

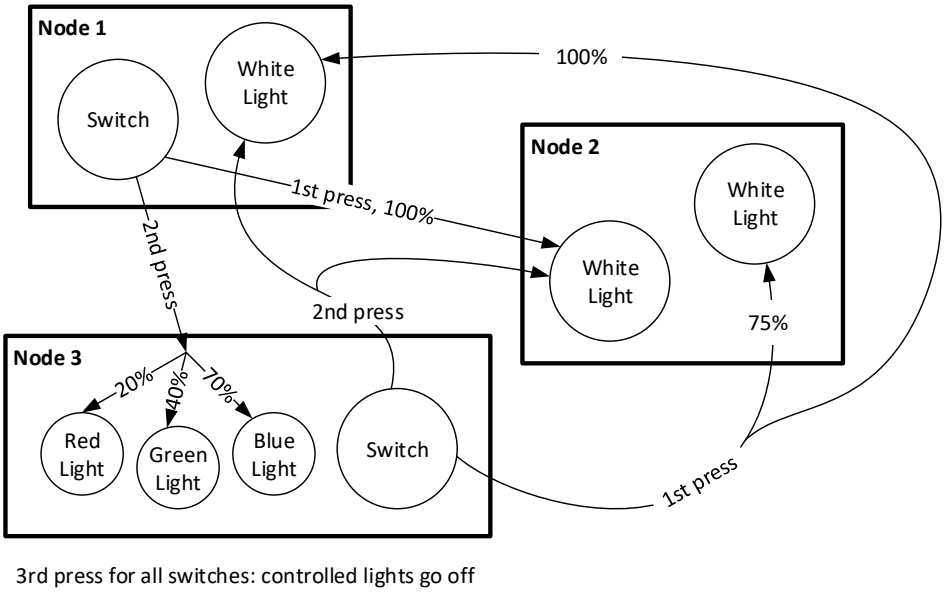

Figure 1: an example of a smart lighting application. Note the arrows indicate functionality, not necessarily direct communication: for example, Node 2 could be out-of-range of Node 1, and hence packets need to be routed via Node 3.

How you interpret the specification, the services that you offer, and how you elaborate upon the brief (e.g. supporting multiple applications on nodes which requires the use of multiple ports, or more elaborate payloads, for example specifying a sequence of times, brightness and colours that the lights should go through, which illustrate splitting the payload at the various layers) are entirely up to *teams* and *teammates* to decide.

The coursework is on networking and requires you to implement a protocol stack, including the *application layer*. Marks are not awarded for an *application* itself though – so don't waste time on a fancy application. Some switches and LEDs connected to the Il Matto via GPIO will suffice (although you probably also want to consider how you are going to configure the operation of the switches). We will however crown the best, working, application as 'winner' (there may even be a chocolate-based prize!).

The Networking Architecture

You are not being asked to implement an existing network architecture (e.g. TCP/IP or ZigBee); in fact we want you to create your own. That said, there is no requirement to invent completely new algorithms (e.g. you can implement existing error control algorithms, routing algorithms, etc). Your architecture must be implemented using the Il Matto + RFM12B hardware. Your *team* should already have at least 3x Il Matto boards and 3x RFM12B radio modules, and you should demonstrate your network using all of these.

Note the wording of the assignment: you are being asked to implement a network architecture, not a protocol stack. This means that your *team* could choose to implement multiple protocols at a particular layer (for example both a reliable and an unreliable DLL). However, there is no necessity to do this, and your network architecture may be a single protocol stack.

TRAN:	Control [2]	SRC Port [1]	DEST Port [1]	Length [1]	APP Data [1-114]	Checksum [2]
NET:	Control [2]	SRC Address [1]	DEST Address [1]	Length [1]	TRAN Segment [8-121]	Checksum [2]
DLL:	Header [1]	Control [2]	Addressing [2]	Length [1]	NET Packet (or part of) [1-23]	Checksum [2]	Footer [1]

Figure 2: Segment/packet/frame field structures defined by the coursework

As a starting point for your protocols, the packet/frame field structures in Figure 2 are defined by the coursework and are non-negotiable. There are multiple ways to use some of these fields (e.g. the Checksum field: this does not have to be a true 'checksum', and could be a parity bit, interleaved parity, a checksum, a CRC, etc; the bits in the Control fields can be used however you like, etc). The numbers in square brackets refer to the field size in bytes; if you do not want to use a particular field (or some of the bits within), fill the field's bits with zeros.

As can be seen in Figure 1, If any NET packet is >23 bytes, it will need to be split into multiple frames. Likewise, if the APP data is >114 bytes, it will need to be split into multiple TRAN segments. You may wish to start by designing your system to work with small amounts of data (<9 bytes, so that it all gets encapsulated into a single DLL frame), and then expand it later.

You'll notice there are addressing fields in the DLL (often referred to as the 'MAC address'):

- You may use these to help route data through the network, by the NET asking the DLL to transmit a packet to a particular next hop, addressed by the addressing field. This is similar to the approaches that Alex will present in his lectures.
- You may however use broadcasting in the DLL, whereby a node/station broadcasts the packet to all neighbours that can hear it. A sensible approach would be to assume that a destination MAC address of 0x00 means that it's broadcast - this is something that should be included in your standard(s). The NET then has to do the filtering on rebroadcasting (and dropping) packets. This is an approach often adopted by epidemic approaches (e.g. packet flooding). If you do this, you need to think about how you'll constrain it from going on for ever (for example, if a station has already received and rebroadcast the packet, it doesn't do it again!).

Splitting the Task across your Team

Although you are working in *teams*, you must each work on unique and clearly defined tasks. The recommended way to implement this would be to give each *teammate* different layer(s) from the 5-layer reference model considered on this module (PHY, DLL, NET, TRAN, APP), where everyone should have at least one layer to work on. However, in some cases, it might be necessary to split a layer into sub-layers (e.g. the MAC and LLC sub-layers of the DLL) in order to appropriately divide tasks. Whatever you choose, it should be clearly defined, agreed amongst *teammates*, and in-line with the spirit/principles of protocol layering.

When you settle on your division of labour, one of the first things that you need to do is agree on the (initial) functionality of each layer and then define the software interfaces. That is, you need to agree on a prototype for the function to call when one layer needs the services of another. You have to keep in mind that the end product is one integrated program.

Although you will be working in *teams*, this is an individual coursework. You should only interact with your *teammates* through clearly defined and agreed interfaces (you may also need to communicate to manage the limited hardware resources, e.g. RAM, CPU cycles, etc).

Standardising your Network Architecture

A proprietary network architecture is fine for communicating within your own network, but limits the potential of a device. A reason for the success of products like Philips Hue lighting is its support for third party components, helped by its use of the Zigbee Light Link standard. We would therefore like, as a stretch goal, for you to demonstrate interoperating with your peer-team's network. To enable this, you must also agree a standard (defining your network architecture) with your *peer-team(s)*, such that you can also demonstrate successful communication with the nodes/stations in their network(s). Your standard should provide enough information for someone to communicate with your stack, even if it's implemented differently (a protocol only defines how communication between peer-layers should *function*).

Although you will be working in *teams*, this is an individual coursework. You should interact with your *peer-teammate(s)* only to define and agree protocol(s). No other collaboration should be necessary (in fact, you will be penalised if it occurs). You do not need to subdivide the tasks in the same way as your *peer-team(s)* (i.e. there does not have to be a 1:1 mapping between the tasks being undertaken by *peer-teammates*).

Your agreed standard may include some optional functionality that is not necessarily implemented by both/all *peer-teammates*. For example, your DLL might include a bit in the Control field which indicates whether the Checksum field is using a single even parity bit (core functionality) or a 32-bit CRC (optional functionality). However, the presence or absence of these optional functionalities should not stop the 'core' (non-optional) functionality from working. This ensures no-one is limited by the capabilities/aspirations of *peer-teammate(s)*.

You MUST define and agree your standard at an early stage in the design process, we expect to see a draft of it at the Design Review Meeting. This can evolve as design progresses (but make sure you agree changes to the stack with the *peer-teammate(s)*).

If you are not sure what a standard looks like, look at http://standards.ieee.org/about/get/. Yours, however, should be a maximum of two pages long!

Observations, Hints and Tips

Here are some observations from previous years' courseworks:

1. You each, as individuals, have a choice over the level of ambition/functionality that you are attempting. Individuals and teams often try to implement complex functionality (and a number of alternative options), but these often end up not working or consuming large amounts of time. We would prefer to see something working and well analysed/evaluated, rather than something complicated but is only part-implemented and not thoroughly evaluated.

2. Regardless, **get something simple working first**. The Il Mattos are constrained devices – it's a much better (and safer) idea to get a simple stack working first, and then expand on the functionality. Think about the resource implications of your design decisions.

3. Your demo session should be used to demo something. We'll read about everything else in the report. Think about what you can **demonstrate** that you won't be able to write in the report – i.e. don't just talk us through your code. If you can't demo anything, don't just try and fill the time regardless!

4. Teams that have struggled to get a stack working tend to be those that 1) leave it too late, and 2) don't have proper discussions around how the stack is going to work, and what the use-cases are for the interfaces between them. **Communicate!**

5. In previous years, there have been numerous cases where *peer-teams* hadn't agreed on their **standards** early on in the design process, and they typically suffered as a result. We don't expect that you'll actually get peer communication working (there are a lot of things that can go wrong), but the standards must be agreed and you must implement what you say in the standard.

6. The coursework requires you to implement your architecture using the Il Matto and RFM12B hardware. Don't use something else. We would recommend developing on the Il Matto from the outset, rather than 'prototyping' on a PC with the intention of porting it across later on.

Disclaimer: *We are trying to make this engaging, practical and useful (and fun), rather than having an individual or theoretical coursework. However, bear in mind that this makes it quite intricate and complex. As such, we may need to make minor tweaks and modifications to the specification and rules as the coursework develops (while ensuring fairness). Apologies if this happens, but please be patient with us!*

Deliverables and Marking Scheme

This coursework is marked out of 100, and contributes 40% of the credit for this module. More marks will be awarded for ambitious implementations if, and only if, the ambitious functionality works. Hence, there is no point aiming for something really complicated if you know you won't be able to get it to work! The coursework has the following deliverables:

- **Design Review Meeting [total of 0 marks]**
  (held during the lab slot on Thursday 07th November 2019)
  - The examiners will meet with all *teams* (a schedule for these will be announced nearer the time). The meetings are deliberately time-limited, so make sure you have a plan for how to approach it. This deliverable is not assessed, but provides feedback.
  - *Teams* (and all of the *teammates* in them) should explain their proposed network architecture and the protocols and services of each layer. Printed hand-outs are encouraged to aid discussion.
  - Each *team* must bring an agreed document outlining the interfaces between their different layers/contributions.
  - Each *teammate* must bring a draft of the standard document that they have been working on with their peer *teammate*.

- **Practical Demonstration [total of 30 marks]**
  (assessed during the lab slot on Thursday 12th December 2019)
  If an individual does not attend their scheduled demonstration session, they will receive a mark of zero for this element.
  - **Demonstration of functioning layer(s) in isolation** [20 marks]: Individual *teammates* should demonstrate the functionality of their layer(s) and explain what is happening and what services are provided. Think about how you are going to demonstrate this to us (even if your *teammates* do not get their parts working); hand-outs are encouraged.

    IMPORTANT NOTE: you may worry that there is an over-reliance on your *teammates* here, i.e. if their layer(s) doesn't work, you'll lose marks. It is important to stress that this is an *individual* coursework, but you are working in *teams*. We want a demonstration of the functionality that *you've* implemented in *your* layer (e.g. a single teammate demonstrating what they have implemented). You do not need your *teammates'* or *peer-teammates'* contributions to be working in order to demonstrate this.

    You may choose to do this demonstration as part of a fully working protocol stack from your team, as long as you can 'isolate' the layers to explain and show what your layer is doing. Alternatively, you may build a test-harness that allows your layer to communicate directly to itself. For example, if you were working on the DLL, you might demonstrate that you pass it a payload of data, show that it splits it into multiple frames with sequence numbers, adds error detection, etc. You could then pass that frame back into the DLL, and show that it can check for errors, recombine all of the frames, and extract and return the payload. For the MAC, you might come up with a quick test harness which

shows roughly how it works, or you might be able to show a timing diagram of its operation on a scope. Be creative, and think how you can show off the functionality you've spent time implementing. However, please note that it's a **demo**, and we want to see you're able to demonstrate. We'll read about everything else in the report!

- **Demo of layer(s) functioning as part of stack** [10 marks]: *Teams* should demonstrate their network architecture permitting communication across their network. Individual *teammates* should explain the relevance of their part.

  IMPORTANT NOTE: you may worry that there is an over-reliance on your *teammates* here, i.e. if their layer(s) doesn't work, you'll lose marks. It is important to stress that this is an *individual* coursework, but you are working in *teams*. Ideally, your *team* will demonstrate the complete protocol stack working and communicating, along with evidence of what's happening at the different layers (e.g. debug output). However, we appreciate that this may not be possible. In this case, we'll be looking to see (and for you to show us) that *your* layer(s) were developed with clear interfaces and services that could be easily used by other teammates. How did *you* write *your* code such that it would work on a single-threaded micro-controller that also needed to execute other layers of the stack? However, please note that it's a **demo**, and we want to see you're able to demonstrate. We'll read about everything else in the report!

- **Demo of networking between *peer-teams*** [0 marks]: *Peer-teams* should demonstrate their networks communicating with each other. No marks are allocated to this, for fairness, but it will help the examiners interpret reports!

- **Individual Report [total of 70 marks]**
  (due by 4pm on Monday 06th January 2020, online hand-in)
  Your report MUST be single-spaced, single-column, using Times New Roman 12pt, and be structured to include ALL of the following:

  - **Introduction** [0 marks]**:**
    - Your report should begin with your name, student ID, and team letter/number. It should then include a diagram of your team's network architecture (protocol stack(s)), highlighting the layers/sub-layers that you worked on. The introduction carries no marks of its own. However, failing to include it will mean we cannot fully understand the remainder of the report, and hence will lose you marks in those sections.
    - *1 page maximum, including figures. Just include a suitable diagram of the protocol stack your team adopted, with clear annotations on the layers you contributed to. No accompanying text is required provided your figure conveys all of this information in a clear manner.*

  - This should be followed by a section for each layer/sub-layer you worked on. For each (note, you may have only worked on one), include subsections for:

    - **Standard Document** [20 marks]:
      - Agreed and collaborated on between *peer-teammate(s)*. It is assumed that this is a joint-effort between *peer-teammates*, and hence all will get the same mark for this part.

- *2 pages maximum, including figures. If you are working on multiple layers/sub-layers, this limit is the total (e.g. if you were working on three different layers, you only have 2 pages to present and show results for all of them).*

- **Design** [20 marks]:
  - A clear account of how *you* chose to implement different parts of the standard, and the design and implementation of the algorithms. What existing algorithms did you implement or adapt? How did they work? How did you structure your code to ensure that you followed the principles of a layered architecture (i.e. interfaces, services and protocols)?
  - *2 pages maximum, including figures. If you are working on multiple layers/sub-layers, this limit is the total (e.g. if you were working on three different layers, you only have 2 pages to present and show results for all of them).*

- **Testing, Results and Analysis** [20 marks]:
  - How do you know your implementation worked and implemented the protocol? How well did it work? Did it work on its own (i.e. in a test-harness) and as part of your team's network architecture? What evidence do you have to support all of this?
  - *2 pages maximum, including figures. If you are working on multiple layers/sub-layers, this limit is the total (e.g. if you were working on three different layers, you only have 2 pages to present and show results for all of them).*

- **Critical Reflection and Evaluation** [10 marks]:
  - What would you do differently if you were to repeat it? What are the weaknesses of your design? What problems did you encounter working in a *team*, agreeing on interfaces between layers, and agreeing on the standard with your *peer-teammate(s)*?
  - *0.5 pages maximum*

- **Appendix** [0 marks]:
  - Please include all the code you wrote in an appendix at the end of your report (i.e. within the same pdf file as your main report). The code doesn't need to be beautifully formatted with line numbers etc (i.e. it shouldn't take more than 5 minutes to do this), but please make sure that you do include it.

The marks for the demonstration will be based on what is demonstrated at the practical demonstration. The marks for the individual report will be based on what is documented in the report when submitted. If there is more in the report, i.e. evidence of design + testing + results, then the mark for the report will reflect this. Clearly, academic integrity regulations still apply, so you can't say that you implemented something in the report that you did not, or fabricate results - this is falsification or cheating!

Academic Integrity

Please ensure that you are aware of the University's regulations on academic integrity, particularly on collaboration vs collusion. While you will be working in *teams*, this is an individual coursework and the report that you submit must be your own work. Any collaboration must be clearly declared. While you are expected to discuss (and standardise) the protocol for your layer(s) with your *peer-teammate(s)*, these discussions should only be at the abstract level, i.e. not discussing the implementation, code, etc.

Note: it is however expected that the 'standard' document in your individual report will be the same for peer-teammates, as it is inherently collaborative and should be identical for it to be a standard!

To avoid any potential issues with recycling, any student who is repeating the coursework must not attempt the same layer(s) again (i.e. you should pick a different layer to design and implement this year).