The structure of the project/this presentation:

* Conducted a literature review into WSN protocols, platforms and the issues they face
* project followed a generic implementation and testing lifecyle, but with more focus on the analysis
* started by designing and implementing our own WSN protocol
* Optimised the network’s parameters
* We produced response time models and evaluated them
* Assesed intergirty against external interference
* Conclusions and suggested further work from results

SLIDE

A WSN is a large interconnected network of resource constrained devices known as motes used for real-time environment montiring – they’re part of a real time system and increasingly used within applications from snail detection to more serious things like monitoring wildfire conditions

WSN protocol used must be able to manage external and internal interference. External – coming from outside the network (deliberate – trying to disrupt network, accidental – heating your lunch in a microwave nearby). Internal – within the network, transmissions overlapping with eachother. Both types need to be managed to ensure the network makes progress.

There are a wide range of protocols to choose from, but the common ground is the use of the IEEE 802.15.4 physical layer because of it low power operation. It provides two operating modes, either beacon enabled or non-beacon enabled. It provides a large range of frequencies to transmit on, broken down into sections known as channels

SLIDE

We looked into two different protocols. The operation of NBE (e.g. zigbee) is essentially “transmit when you and the channel are ready”. BE e.g. WHART. Assigns each transmission a timeslot (a bit of bandwidth) which are then ordered to form a superframe which prevents transmissions from overlapping naturally – as long as all of the devices are synchronized (by the network manager node). External interference is managed by channel hopping and blacklisting – changing channels every transmission and avoiding channels that are too noisy

SLIDE

We also considered two execution platforms that ease development. TinyOS uses a C-like language nesC. Mote Runner which supports most HLLs and alo has a range of development tools – simulator means we don’t have to keep loading the physical motes. A brief look into response time analysis was also done to give us some background for when we produced our response time models

SLIDE

We began the project by defining a data transfer specification – just randomly generated that we wanted to implement with our new protocol. We assumed that firstly, all of the devices (A to E) were in the range of eachother such that they don’t have to route packets through the network and also that all the tarnsfers have the same period, such that none need to repeat before all have been completed.

SLIDE

Based on the speicifcation, our protocol was easy to base off of whart since it has deteminisim about when the transmissions will happen – zigbee is very random and can lead to starvation of some devices as others get all the bandwith. Hence we define our protocol, LikeWHART. Implemented in mote, we used the ame synchronisation policy with a centralized network manager. We employed channel hopping, but did this on a per superframe basis such that each time a superframe started a new channel was used. We were planning to implemented packet acknowledgement to ensure delivery of messages but Mote Runner lacked the documentation on the key functions with which to handle the packets and as such we couldn’t implement it.

SLIDE

So we began our definition of response time models based on two ordering policies – FPS (where we’ve already assigned priorities to the transfers in the speicication) and STF where we just order transmissions by their increasing size. Using our previous assumptions we produced two response time models that we’d later prove by measurement. We introduced the concept of the synchronisation constant, lamba which is the time between the end of the superframe and the start of the next one – effectively the idle time of transmissions – we’d investigate whether this is constant.

We have W being the size of the transfer, tau bein the maximum payload size of the network’s transmission and delta as the length of the timeslot.

SLIDE

Simulator used for analysis and use the logger outputs were used to act as packet acknolwegdment – when a device received a packet it outputted “packet received”.

Optimising the timeslot length (bandwith assigned) to devices to complete a transmission. Larger payload = more time so we optimized this for a range of payload sizes based on the spread of the data transfers specification. we wanted packet fault rate (number of packets lost) of 0 for the smallest timeslot possible.

Synchronisation constant found by measuring time between end of superframe and start of next.

Intergrity of LWHART tested by introducing a rogue device which cycled through each of the 16 channels broadcasting on them, We increased rate of channel changing to see the effects on network.

theoretical response times calculated for our transfer specification and then deployed it as an actual LikeWHART network and measured the real values to determine the accuracy of our model

Here’s a brief demo in the simulator….

SLIDE

So results.

optimal timeslot doesn’t actually double in size for a doubling of the payload size in the network it more slowly grows. We can see on the left however that the choice of timeslot is important as the packet fault rate grows exponentially when the timeslot gets increasingly too small for the transmission size.

We found the synchronisation constant was in fact constant at 25ms (on average) across all timeslot sizes.

We compared the response times to our theroretically computed response times for the data transfer specification – our response time models were right!

Our integrity analysis was hardly surprisingly. LikeWHART doesn’t handle external interference at all well. As we increased the rate of channel hopping in the rouge device the packet fault rate grew rapidly since it has more chance of casuing a collision.

SLIDE

So what have we learnt?

* LikeWHART is a slow protocol – whart uses 10ms for its timeslot size and we got nowehere near that
* As mentioned, likewhart is terrible at handling external interference. It doesn’t manage it, it just tries to avoid it
* The synchronisation constant is a constant, but it’s also huge and needs to be minimized
* Our response time models were correct when compared to the actual values and STF has a much lower average response time for our data specification
* Essentially, LikeWHART needs a lot more work to be useful to anyone

SLIDE

So likewhart’s implementation was severely limited by the choice of mote runner especially packet acknowledgement. In the near furture we don’t have enough time to fix any of the funtiocnality so we’d just condut more analysis moving the physical motes to remove the doubt about the simulator. In the considerable future we’d change execution platform straight away and then be able to improve the integrity handling of the protocol as well as make the topologies more complex and of course we’d define new response time models for these .