Pressure variations:

Variations in pressure can also be used as a power source. If piezoelectric generators are incorporated in the shoe heel, then it could generate power during human walk. However, application of such technologies to WSN is unclear.

Flow of air/liquid: The flow of air or liquid in wind mills & turbines are also source of power. But these techniques doesn't have noticiable results for WSN applications.

* Energy scavenging usually associated with secondary batteries along with actual power sources. Hence, additional circuitry for battery recharge or battery technology that are recharged at low current is essential.

2.2 Energy consumption of sensor nodes

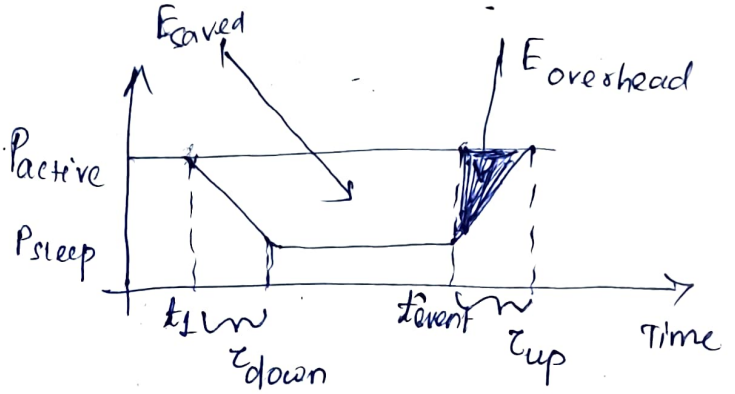
Operation states with different power consumption:

The energy consumption in a sensor need to be tightly controlled as batteries have small capacity & recharging by energy scavenging is complicated and volatile.

- For example if a battery can store 1J. If this is used to power a node for single day, then the node must not consume continuously more than $1/24.60.60$ W/s ≈ 11.5 μW. Actually no current controller of entire node able to work at thos power levels. The lot of lees on designing low power chip but these chip improper operation can loose its advantages.
- Therefore, the best energy consumption can be achieved by properly operating sensor node, as most of time sensor node are inactive. Hence they need to be turn off. The sensor nodes wake up again when external stimulus is given. Therefore, introducing and using multiple states of operation with reduced energy consumption in return for reduced functionality is the core technique for energy-efficient WSNs.

⇒ For a controller typical states are "active", "idle", and "sleep". Similarly, a radio modem can turn on transmitter, receiver or both on or off. Likewise sensors and memory could also be turned on or off. The most frequently used terminology is "deeper" sleep state if power consumption is low.

→ Furthermore, transition betⁿ. states take energy and time. The deeper is the sleep state more time and energy it takes to wake up again to operational state. Therefore, it is worthy to be in idle state rather entering into deep sleep states from energy consumption point.



→ At time t1 decision is taken for a component to be in sleepmode or not to reduce power consumption from P_{active} to P_{sleep}.

If it is active and next event occurs at time t_{event}, then total energy $E_{active} = P_{active}(t_{event} - t_1)$ is spent unnecessary for idealing the component.

→ If component is kept in sleep mode for time τ_{down}, when sleep mode reached completely, then average power consumption during this phase is $(P_{active} + P_{sleep})/2$. The P_{sleep} is consumed till t_{event}. Hence, total energy required for sleep mode until t_{event} is

$$\tau_{down}(P_{active} + P_{sleep})/2 + (t_{event} - t_1 - \tau_{down})P_{sleep}$$ as compared to $(t_{event} - t_1)P_{active}$ for active time. The energy saving is

$$E_{saved} = (t_{event} - t_1) \cdot P_{active} - (\tau_{down}(P_{active} + P_{sleep})/2 + (t_{event} - t_1 - \tau_{down})P_{sleep}).$$

→ When the event is processed to start an additional overhead of $E_{overhead} = \tau_{up}(P_{active} + P_{sleep})/2$ is incurred to come back to operational state, before the event processed

this overhead is the time when no useful activity is undertaken. Therefore, switching to a sleep mode is only beneficial if $E_{overhead} < E_{saved}$ or if the time to the next event is very large.

$$(t_{event} - t_1) > \frac{1}{2}\left(\tau_{down} + \frac{P_{active} + P_{sleep}}{P_{active} - P_{sleep}}\tau_{up}\right).$$

Microcontroller energy consumption

As discussed in previous section the embedded control implement multiple operational states.

Intel StrongARM : It provides three sleep modes

→ In normal mode all parts of processor are fully powered. (400mw)

→ In idle mode clocks to CPU are stopped, clocks to peripherals are active so that any interrupt will cause return to normal mode (100mw)

→ In sleep mode only real-time clock remains active. Wake up occurs after a timer interrupt and ~~takes~~ consumes power ~~of~~ upto 50mw.

Texas Instruments MSP 430 :

It has one fully operational mode of 1.2mw power consumption, it has four sleep modes. In deepest sleep mode the controller is woken up by external interrupts. In next mode clock is used for scheduled wake ups.

Atmel ATmega : It has six different modes of power consumption. Its power consumption varies bet^n 6mw & 15mw in idle and active modes.

Dynamic voltage scaling :

Rather than the discrete operational states, continuous operation is acheived by adapting the operational speed of controller. Therefore, the controller need to be in full operational mode to complete a task at high speed and then go back to sleep mode quickly. The major fact is controller that runs at low speed, low clock rates takes less power than full speed. This is possible because of supply voltage reduction at lower clock rates, while guaranteeing correct operation. This technique is called Dynamic voltage scaling (DVS).

$P \propto f \cdot V_{DD}^2$. DVS also reduces energy consumption. $E = PT$

Memory :

From energy perspective, most relevant memory are on-chip memory of a microcontroller and FLASH memory. The construction and usage of FLASH memory heavily influences node lifetime. The performance metrics are read and write times and energy consumption.

→ Read time & read energy consumption are quite similar for FLASH m/m. However, writing is complicated and is time and energy consuming task.

Radio transceivers :

The transceivers are made energy efficient by making them off for most of time and is activated when necessary. Therefore, it has low duty cycle. However, it incurs additional complexity, time, and power overhead.

Modeling energy consumption during transmission

The most of energy consumption by transmitter is due to (i) RF signal generation (depends on modulation type, transmission power P_{tx} of antenna)
(ii) Electronic components required for frequency synthesis, frequency conversion, filters etc.

→ The most critical decision made is to choose transmitting power P_{tx} for a packet. P_{tx} depends on E_b/N_0, bandwidth efficiency η_{BW}, distance d and path loss coefficient γ. The transmitted power consumed by amplifier is P_{amp}. Then

$$P_{amp} = \alpha_{amp} + \beta_{amp} P_{tx}$$, where α_{amp} and β_{amp} are constants depend on process technology & amplifier architecture. ($\alpha = 125$, $\beta = 5$

Efficiency of power amplifier η_{PA} for $P_{tx} = 0\,dBm = 1\,mW$ power is

$$\eta_{PA} = \frac{P_{tx}}{P_{amp}} = \frac{1\,mW}{174\,mW + 50 \cdot 1\,mW} \approx 5.5\%$$

→ The amount of energy to transmit a n-bit long data, depends on nominal bit rate R and coding rate R_{code}, and total consumed

Power during transmission.

$$E_{tx}(n, R_{code}, P_{amp}) = T_{start} P_{start} + \frac{n}{R R_{code}} (P_{txElec} + P_{amp})$$

where P_{txElec} is the power required for other circuit than amplifier during transmission.

Modeling energy consumption during reception

Like transmitter a receiver can also made turned on or off. while on on state it may receive a packet or be idle, observe the channel, and ready to receive. Therefore, the power consumption while it is turned off is negligible.

→ Let the energy required to receive a packet E_{rcvd} has startup components $T_{start} P_{start}$. The receiver also has component proportional to Packet time $\frac{n}{R R_{code}}$. During actual receiving, the receiver circuit is powered up with Power P_{rxElec} which is required to drive LNA ⟨or⟩ RF font end. The final component encurred with energy bit is the decoding overhead. The final received energy E_{rcvd}

$$E_{rcvd} = T_{start} P_{start} + \frac{n}{R R_{code}} \cdot P_{rxElec} + n E_{decBit}$$

From all these energy components, decoding energy is complicated model and depends on number of hardware and system parameters. for example decoding done through a dedicated hardware or in software on a microcontroller. It further depends on supply voltage, decoding time per bit, length of code etc.

Dynamic scaling of radio Power consumption:

The idea of these approaches is to dynamically adapt modulation, coding and other parameters to maximize the system metrics like energy efficiency.

→ Applying concept of DVS for controllers to radio transceivers is not trivial. This is because scaling down supply voltage or frequency

to obtain low Power consumption in exchange of higher latency is only applicable to some electronic component. Components such as amplifiers cannot be scaled down, as its radiated and consumed Power depends on the communication distance and has to run at high power for extended period.

→ Similarly some Parameter versus Performance trade-off like choice of modulation and/or code leads to Dynamic Modulation Scaling (DMS), Dynamic code scaling (DCS). Dynamic Modulation code Scaling (DMCS) optimization techniques.

Relationship between computation and communication

Considering the energy consumption for both microcontrollers and radio transceivers we need to choose the best coarse among both to Provide energy resources of a sensor node. Is it better to expense energy or data sending or on data computation.

→ ~~Computing~~ Communication of 1kB data over 100 m consumes almost same amount of energy for computing 3 million instructions. considering these results, it is clear that communication is more expensive than Computation. Therefore the main idea is to invest on computation in a network whenever possible to limit Communication costs, this lead to the concept of in-network processing and aggregation.

Power consumption of sensor and actuators

Providing exact guidelines for power consumption in sensors & actuators is difficult because of their diverse operation. for example passive sensors require low power compared to active sonar, ~~where~~ where Power consumption is quite large.

2.3: Operating Systems and execution environments

2.3.1 Embedded operating systems:-

Operating System or an execution environment in WSN support specific tasks of system such as need for energy efficient execution, i.e energy management support in form of controlled shutdown

of components or Dynamic voltage scaling (DVS) technique.
Similarly, the external sensors, temeses should be handled efficiently.
→ to avail the above mentioned facility appropriate Programming model, Structured Protocol stack and efficient energy management is required.

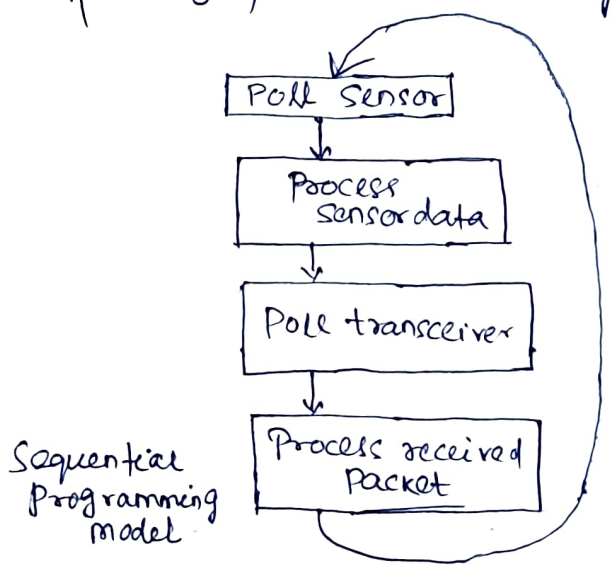
2.7.2: Programming Paradigms and application Programming interfaces

Concurrent Programming:

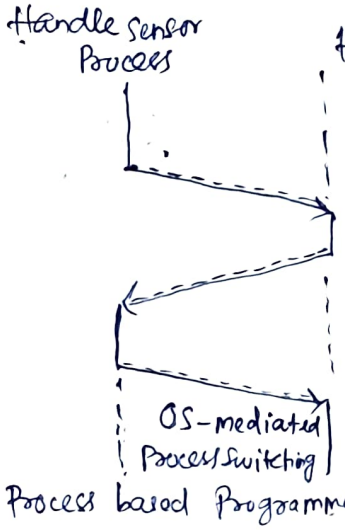
Concurrency in programming is essential in WSN because WSNs handle data from arbitary sources. The WSN system polls a sensor to check the data availability and process the data, if then pole a transceiver to check packet availability and process the packet. However, this sequential model lead to the risk of missing data either during packet processing or sensor data processing. This risk becomes high when processing of data or packets take more time. Therefore, Sequential Programming model is not sufficient.

Process based concurrency:

Most operating system support concurrent (Parallel) execution of different processes on one CPU. But there are granularity mismatches indicating that individual protocol layers or functions with individual process lead to a high switching overhead from one to other process. This may get severe when smaller tasks need to be performed compared to the switching overhead between tasks. Secondly, there is requirement of memory (stack space in) for each task causing memory constraints of sensor nodes.



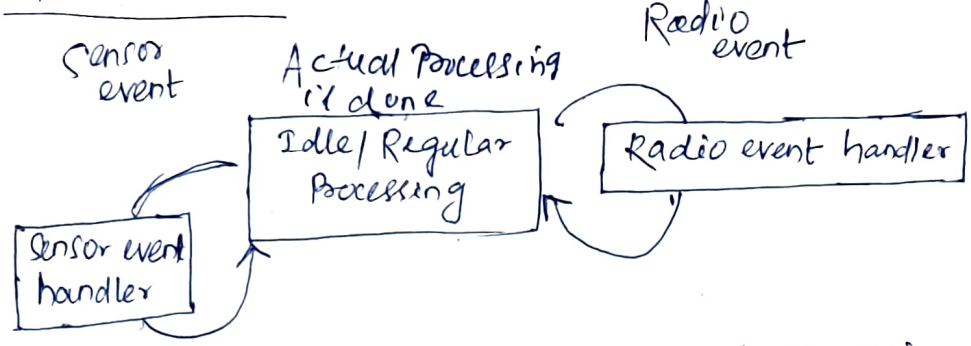
Sequential Programming model



Process based Programming model

Event-based Programming

In this system, it waits for an event to occur. The event may be data availability from sensor, packet arrival at transceiver, or timer expiry. These events are handled through short instructions that can only store information about which event has occured, whether a byte is for a packet or for sensor data.



→ These event handler interrupt the normal code processing. Event handler don't interrupt each other and executed one after other. Hence, event-based programming allows time-critical event handlers without interruption in their execution and processing of normal code that is triggered by event handlers.

Interfaces to the operating system:

Apart from the programming model it is also necessary to specify the interfaces to know how the internal system's state are enquered and set. These interfaces need to be accessible from protocol implementations and the interface allows these implementations to access each other. These interfaces is closely tied with the structure of Protcol stack → OSI model: Physical, datalink, Network, Transport, Session, Presentation, Application

* Protocols are the set of rules that are used for communication between different layer of a network.
* Protcol Stack or network stack are the group of protocols combinedly operate to function the network robustly & efficiently. Protcol stack have different layers to perform error finding in some layers, data routing & ~~data traffic~~ in some other layers.

An application Programming Interface (API) consists of functional interface, object abstractions and detailed behavioral semantics. Abstractions are wireless links, nodes. The functions include state inquiry and manipulation, transmitting data, hardware access (sensors, actuators, transceivers), Device for energy trade-off.

Structure of operating System and Protocol Stack

Traditional comm^n. protocol structure use layering of individual protocols stacked on top of each other. Each layer uses functions of the layer directly below of it. This layer stacking helps in managing protocols, deals complexity, promotes modularity and reuse. WSN may not follow this strict layered approach, for example considering the information of RSS from communication partner. This is a Physical layer information, which can assist network protocols to decide about routing changes. This is because the signal may become weak due to node movement or removal of node in next communication, This Physical layer information help in computing location information ~~through~~ by estimating distance from the signal strength, this information also assist link layer Protocols in channel adaptive schemes. Therefore, one single source of informat^n can be used to the advantage of many other protocols not directly associated with the source of this information. This technique is known as (cross-layer information exchange).

Also layered architecture is replaced by component model. In this case large, monolithic layers are broken into small, self contained 'components', "building blocks", or "modules". These components are used to accomplish a well defined function. The major difference compared to layered architecture is that the component interactions are not confined to immediate neighbors in an up/down relationship, but can be with any other component.

→ The component model not only solves the structuring Problems for protocol stacks, it also fits to event based approach for programming WSNs. Wrapping of hardware, communication primitives, in-network Processing functionalities can be designed and implemented as components. Example of such as operating system is TinyOS.

Dynamic energy and power management.

Energy efficiency in WSNs can be improved by switching components into different sleep states, reducing the performance by scaling down voltage or frequency, selecting modulation and coding techniques. To control all these a decision need to be made by the operating system, Protocol stack, or by application event, when to switch among these states. Therefore, Dynamic Power management is needed.

→ The complicated factors to DPM is energy & time requirement for the transition of component between two states. Different DPM scheme are

Probabilistic state transition policies:

considering the policies that regulate transition bet? different sleep states. considering sensors that randomly placed in fixed area and assuming the events arrive with certain temporal distribution (time behavior) and spatial distribution (space). This information helps in computing the probabilities for the time of next event, after processing of one event.

→ This probability is used to select the deep sleep state out of several possible states.

Controlling dynamic voltage scaling

As many task need to be run in an operating system, it require to decide which clock rate to be used in each situation to meet the deadlines. This require feedback from applications. DVS based control into the kernel of operating system provide energy efficiency in mixed workloads without modifications to user programs.

Trading off fidelity against energy consumption: The DVS based controlling of task assumes hard deadlines for a task which is not appropriate

assumption in WSNs. However, certain WSN tasks can be computed with higher & lower level of accuracy. In WSNs there is a trade-off against energy required to compute a task. These WSNs have the characteristics of inaccuracies for lower energy consumption. for an example considering energy-quality trade-off for algorithm design, for Signal Processing Purposes. Here task is to transform an algorithm such that it quickly approximate the final result and keeps computing as long as energy is available, producing refined results, a counterpart of imprecise computation where computation is only continuing as long as time is available. (Time vs energy)

Case Study : Tiny OS and nesc

It has been observed that the concurrency requirement of sensor node software is feasible with confined resources and simple sensor node hardware is supported through event based programming model.

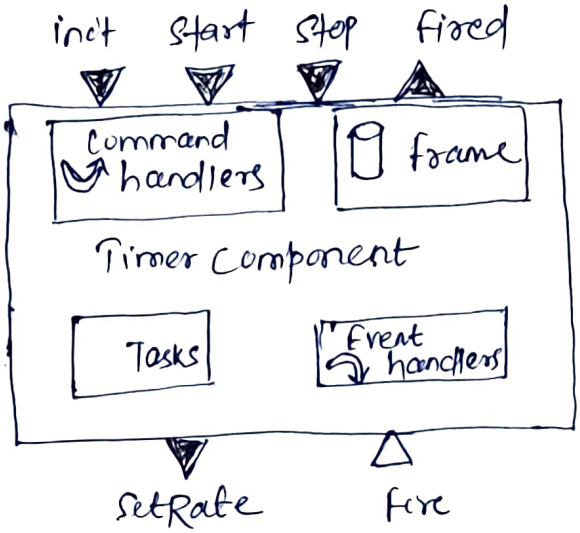
→ The major challenge for the event based programming is how to tackle the power efficiency of such model considering the complexity associated with many individual state machines and data transfer between each other. In addition modularity to be supported for easy exchange between the state machines.

→ The operating system TinyOS along with nesC programming language addresses these challanges.

TinyOS

→ It supports modularity and event based programming through the concept of components. Components have systematic functions such as handling radio interfaces or for computing routes. components contains these information in frames the required (state information in a frame) the program code for normal tasks, and handlers for events and commands components are arranged hierarchically from low-level components (close to hardware) to high-level components (actual application). Events originate from hardware and passed upward from low-level to high-level components. Similarly, commands are passed from high-level to low-level components.

Timer component example



→ This component provide abstract version of hardware time.

→ It has 3 commands "init", "start", and "stop". It also handles one event "fire" from another component. For example the wrapper component around a hardware time issues "setRate" commands to this componet and emit a fired event itself.

→ In this event-based paradigm both command and event handlers conclude their run. The commands are not blocked for indeterminate amount of time because they simply request for some task of lower heirarchy component. Likewise, event handler provide information of its components frame and arrange it for later execution.

→ The actual computation is done in tasks. In TinyOS tasks are run to completion but are also interrupted by handlers

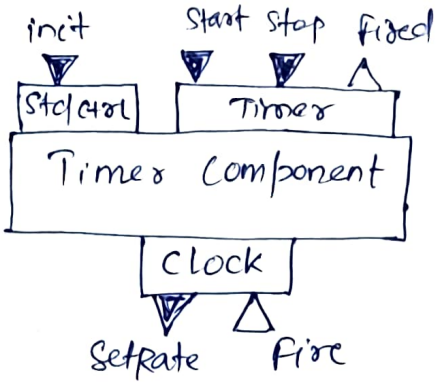
→ When handlers and tasks need to run to completion, the component need to get feedback from other component about the ~~time~~ final Operation of a command. For example how the Automatic Repeat request (ARQ) Protocol learn from the MAC Protocol whether Packet is ~~received~~ sent successfully or not. For this split-phase programming approach requires for each command. In first phase command is sent and in second phase outcome of the operation is delivered by a second event.

→ As commands and events are the only ways of interaction between components, therefore structuring of commands and events together forms an interface betn two components. Hence the commands that a component understands and the events a component may generate are becoming its interface to the components of higher layer. i.e. components can execute certain commands at its lower component and receive events from it.

→ The nesC language allows a programmer to define interface types that define commands and events that belong together. This allows split phase programming style by putting commands and their corresponding completion events into the same interface.

Organizing timer component using interfaces

Timer component is reorganized into using a clock interface and two interfaces StdCtrl and Timer. The Timer component is defined as a module since it is a primitive component directly containing handlers and tasks.

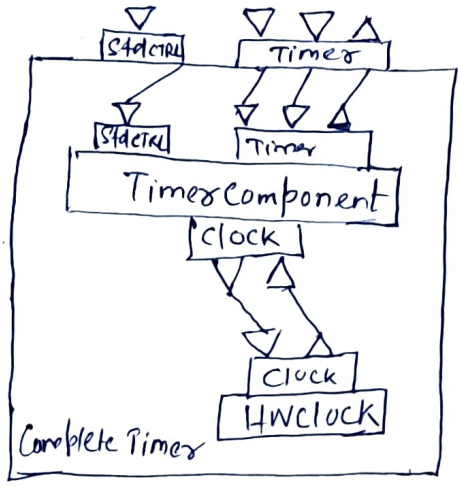


→ The above component can be combined into larger configuration by wiring appropriate interface together. For wiring components with correct interface types can be plugged together.

Example of new component CompleteTimer

When ~~Timer component~~ TimerComponent is wired with additional HWclock a new component CompleteTimer is formed. This component exposes only StdCtrl and Timer interfaces to outside.

→ The figure shows how a larger configuration is built out of two components.



→ Hence using component definition, implementation & connection concepts, Tinyos and nesc form powerful and easy basis to implement core Operating System functions as well as Protocol stacks and application functions.

Overally Tinyos is currently regarded as the standard implementation Platform for WSNs

Other examples: Apart from TinyOS, there are few other execution environments or operating system for WSN nodes. These are Contiki which is ported to various hardware platforms and implements a TCP/IP stack on top of a platform with severely restricted resources. The other examples are ecos & Mantis project.

Some examples of sensor nodes:

The sensor nodes based on different application scenarios have different requirements such as battery life, mechanical robustness of node's housing, size and cost etc. Few such sensor nodes are:

1. The "Mica Mote" family:

The commonly used Mica motes have various versions (Mica, Mica2, Mica2Dot). These sensor nodes are commonly available through the crossbow company. The OS used for these nodes is TinyOS.

→ These boards have microcontroller of Atmel family, a radio modem (TR 1000) with additional connections. Therefore, connecting additional sensors (humidity sensors) enable wide range of applications and experiments. Sensors are connected with controller via I2C or SPI bus.

2. EYES nodes:

These nodes are developed by sponsered Project "Energy efficient Sensor Networks (EYES). It is equipped with a Texas Instrument Msp 430 microcontroller, Infineon radio modem TDA 5250. Along with a filter and transmission Power control. The radio modem also provide measured signal strength to the controller. The node has USB interface to PC along with possibility of sensor/actuator addition.

BT nodes: BTnodes feature an Atmel ATmega 128L microcontroller with 64+128 KB RAM and 128KB FLASH Memory. These nodes use Bluetooth as their radio technology in combination with Chipcon CC1000 operating bet^n. 433 and 915 MHz.

4. scatterweb :

These sensor nodes based on MSP 430 microcontroller and ranger cyfto embedded web servers which has wide range of interconnection Possibilities apart from Bluetooth and low power radio mode. I^2C or CAN connections are available.

Conclusion :

The basic components of sensor and the energy consumption of every component is observed. From this WSNs consist of two parts one part continuously vigilant to detect and report events and requires negligible power. The second part that performs actual Processing and communication has high power consumption and to be operated at low duty cycle. These separate functionalities