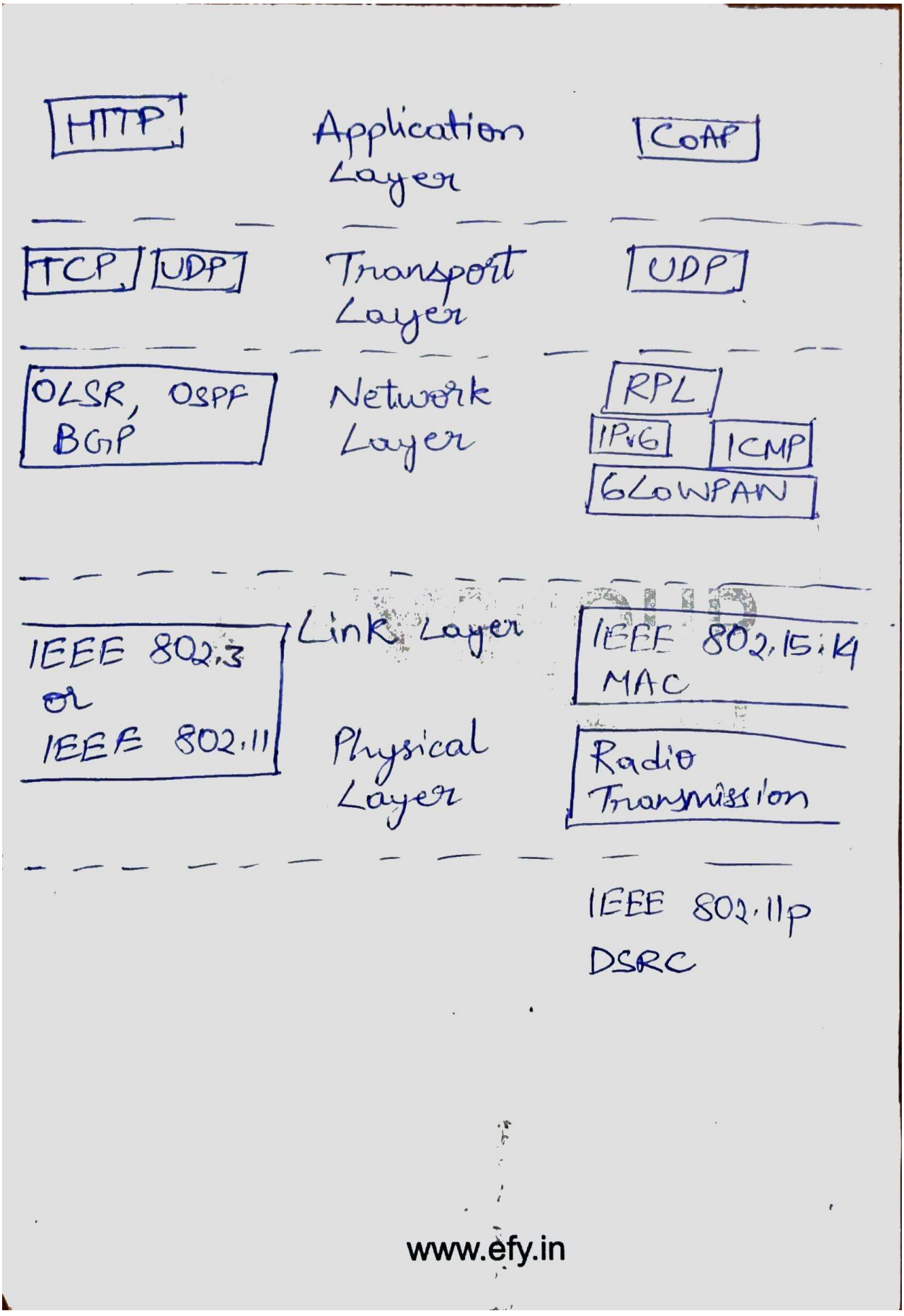
Network Stack

Tuesday, June 30, 2020 10:23 PM


HTTP
Application Layer
CoAP
TCP
UDP
Transport Layer
UDP
OLSR, OSPF BGP
Network Layer
RPL
IPv6
ICMP
6LoWPAN
IEEE 802.3 or IEEE 802.11
Link Layer
IEEE 802.15.4 MAC
Physical Layer
Radio Transmission
IEEE 802.11p DSRC

2020_07_01 3_15 pm Office Lens

Wednesday, July 1, 2020 3:16 PM

21st March, 2020

→ Processor:
Instruction sequences

ASK
FPGA → hardware design.
Image processing, big data analytics.

→ clock control unit.

Direct memory access.
interfaces:

Programming & debugging
JTAG, PDI,
bootloader.

Arduino ← "Shields".
FPGA → Basic logic unit → combinatorial logic
look up tables
MUX
Register (sequencial, clocking)

Xilinx
Altera
Actel.
| Spartan-6 evaluation kit.

Design workflow

Hybrid platforms → SoC.

Metal Programable blocks.

→ Linux kernel development book – 3rd addition: Robert Love.

→ Micro kernel: All functions & services are removed from the kernel space and moved to the user space.

→ Use single microkernel OS for a much wider range of applications rather than a realtime executive.

→ POSIX APIs
  1003.1
  ANSI X3J11 C
  * Realtime Extensions
  * Threads
  * Additional Realtime Extensions
  * Application Environment profiles.

→ microkernels:
  * thread, signal, message-passing, synchronisation, scheduling, timer, process management
  * NO: filesystem, TCP/IP, GUI, network manager, etc.

→ Modular kernel.

→ Contiki: modular
  * Proto threads: event driven model + threads
  - Sequencial.

Cooja network simulator.

Radio networking stack called "Rime".

→ What is a network stack?
TCP/IP stack?
IPV4/IPV6 stack?
Bluetooth stack?
BLE stack?

Base Band Protocol.
FH-SS (79 carriers).
LMP (link manager protocol).

→ Continous variable scope delta.
(CVSD).
PPP — Point to point protocol.
OBEX — Object Exchange.

→ nesC :

structured component base extension of C language.

Components	Interface → Tasks

TinyOS.

Resumé :

24th March, 2020

→ Embedding Sensors and Motors :

- University of Colorado Boulder.
- Developing Industrial Internet of things.
- Sensor and sensor circuit design
- Motors and motor control circuits.

mokutil --test-key MWARE15.der

RiotOS

→ ANSI C

→ Inherited from the Kernel.

Modularity.

→ Scheduler works with periodic events & works as a tickless scheduler

↓

switch to idle thread when no functions

↓

defines the sleeping duration.

→ Small kernel size

⇒ can run of very low clock speeds.

Scheduler designed to minimize thread switching ⇒ reduce overhead by context switching.

→ Interrupt handling

* API ≈ common sensor net programming model.
* Lowest possible interrupt latencies MSP430.

→ Seperation of CPU code & kernel implementation.
⇒ Use complex processor features.

→ Max reliability, strong realtime characteristics!

→ Multithreading.
Distributed systems → kernel messaging APIs

→ Different network protocols like 6LoWPAN & RPL, IPv4, IPv6, UDP,

→ Network stack: modular
⇒ protocol layers at any hierarchy.
† Adaptation layer.

0.00048828

→ Contiki OS.
→ Repair partition.

IPv6
→ Packet header design
→ bootstrapping
→ neighbor discovery,

* Adaptive network stack.

IoT OS requirements:

* Ability to run on a constrained hardware.
* Ability to leverage the capabilities of a powerful board.

/usr/local/src/tinyos.

LLN - Lower power, busy networks.

* TCP: congestion control mechanisms are too complex for LLNs. & Packet loss does not indicate congestion.

IETF - Internet Engineering Task force.

CoAP - Constrained Application Protocol.

Microkernels:

* Can be slow if IPC is suppar
* Cannot handle complex data structures.
* Every piece of OS needs to be modularly rewritten using messaging.

Options:

1. Monolgithic, layered, micro or hybrid?
2. Scheduling & schedulers:
   - real-time
   - different priorities & degrees of interaction.
3. Programming model:
   U-space | Multi-threading
   k-space | multi processing

   ⇒ Programming language.

Contiki - a hybrid operating

→ Event driven
→ Preemptive multi-threading is implemented as application library.
* Per-thread stacks not needed or locking mechanisms.
* Event driven model with state driven programming are difficult.

→ Communication by posting events.
→ No hardware abstraction layer.
⇒ Communicate directly with hardware

1. Event scheduler
2. Polling mechanisms.
3. Single shared stack

RDC: Radio Duty Cycle.

Real-time Systems:

Job → instance of a task.

Period P: $J_{1,1} \xleftrightarrow{P} J_{1,2}$

E: Execution time

D: Deadline.

Feasibility interval

phase, jitter (r^-, r^+)

Hyperperiod H

Precedance graph.

---

compressor clutch.

Resistance based sensor

PWM for compressor ⇒ differential

Resistor control for motors

~~motor~~ blower motor voltage drop

Add on heater (electric).

CoE verticals :

→ All CoE activities should converge to some project and should be implemented.

→ MoP related points.

→ Goal sheet.

Static schedulers

t	T
Scheduling time	tasks

$$U = \sum_{i=1} \frac{e_i}{P_i} \quad \text{(utilization)}$$

Tick scheduling → time sharing

Maximum frame size.

$F = H/p$

* $f \geq \max_{1 \leq i \leq n} (e_i)$
* $f \mid H$.
* if $t' > t \Rightarrow F(k+1) \in [t'; t'+D_i]$
  $\Rightarrow t + 2f \geq t' + D \Rightarrow 2f - (t'-t) \geq D$

1. Application Layer :
which application?
what type of message?
send/ recieve/ control?

2. Presentation layer / syntax layer. (compression, encryption).

3. Session layer

4. Transport layer:
QoS, Congestion control, packets, error correction, reliability.

5. Network layer:
Routing, packets → frames, (IP address), routing protocols, etc.

6. Link layer:

IP → physical, hardware usability, error due to noise correction.

* Ethernet, WiFi, Bluetooth, Zigbee, etc.

7. Physical layer:

Data bits → EM pulses, timing, multiplexing, modulation scheme, etc.

Build a device & assign it an IP address

⇒ It is now part of IoT.
Congrats.

[illegible]/4
[illegible] 78125

Performance, cost, energy

Real-time systems

Web connectivity & security

49, 16, -12851 ?

{104, 101, 108} → {111, 106, 99}

⇒ {8, 5, 12} → {15, 10, 3}

52, 10, -12851

71, 50, 71

64 512/115

6×22

64, 285

Factors fuling IoT

→ Sensor sophistication & prices,

→ Directly connect to the Internet

→ IPv6

→ Cloud computing

IoT for Manufacturing

* Ability to put realistic load and boundary conditions,

→ Digital Prototype,

→ Digital twins using an IoT platform.

⇒ Replace load conditions.

Ajitabh Sarin, Lakshman Shahidhar,

BMA?

→ LIN, CAN, CAN-FD, FlexRay, Ethernet.

Digital Twin:

→ Interference,
→ look like
→ weight.

→ function
→ failure.
→ test,
→ decision,
→ requirement
→ How it works?

System Modelling

System Engineering

Network Protocols
Embedded Software
Electrical Systems

PLM & ALM

S/w → Driver → hardware

Functional Architecture,

↓

Logical Architecture, (Integrator)

↓

Electronic design.

(ICD data "edm" format to PCB),

S/w architecture → NN design

↓

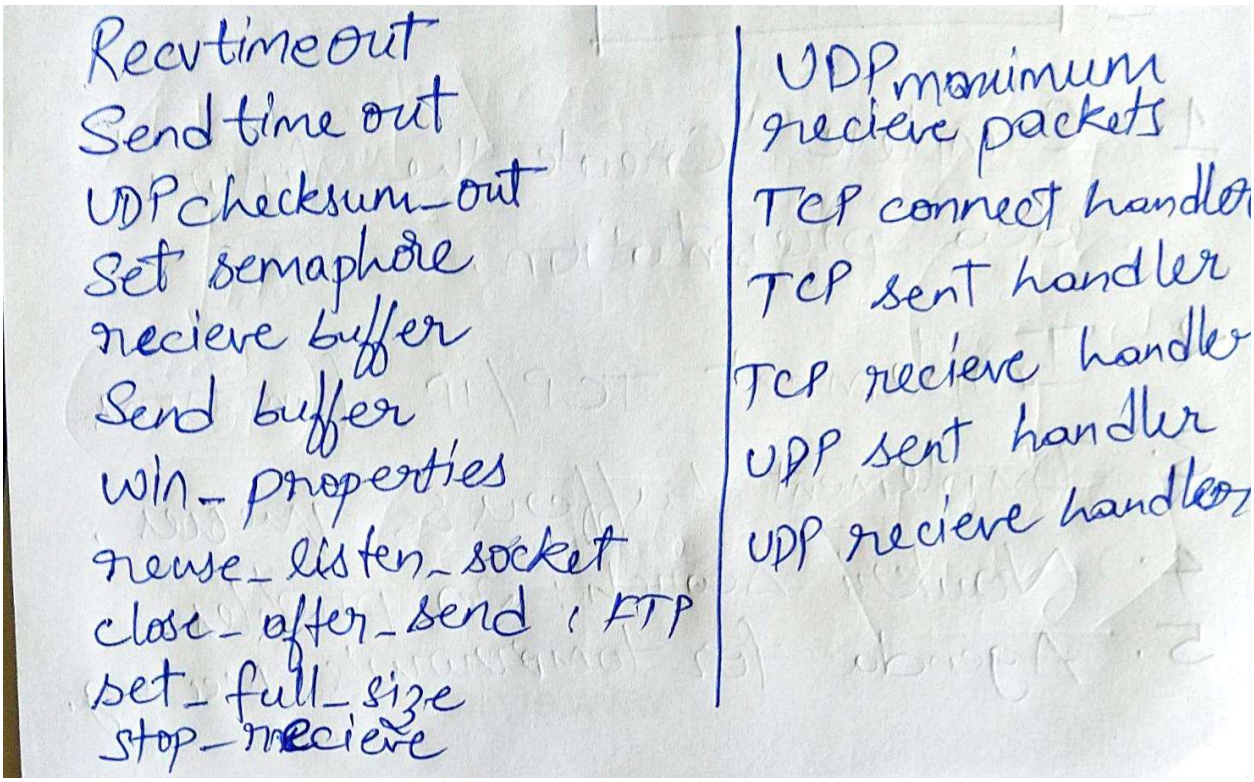
Embedded S/w Implementation,

MIL
SIL
HIL
} Simulations

ECU extracts, network extracts,

Xpedition Hyperlynx

Palarlon


Recvtimeout
Send time out
UDP checksum_out
Set semaphore
reciere buffer
Send buffer
win_properties
reuse_listen_socket
close_after_send ; FTP
set_full_size
stop_reciere
UDP maximum reciere packets
TCP connect handler
TCP sent handler
TCP reciere handler
UDP sent handler
UDP reciere handler

Connect → send
(sendto)
bind → listen → accept.

server: DHCP should be off
⇒ statically assign IP.

Server side:

1. Initialise network ✓
2. Get IP address assigned ✓
3. Create a socket ✓
4. Bind it to a port ✓
5. Put in listen state ✓
6. Accept incoming requests ✓
~~into a buffer~~.
7. Recieve data into a buffer.

PGD ESD at CDAC

TA - Recruiter
HR - Jr
HR - Sr
HM - Jr
HM - Sr.

---

* How to send the file into the TCP socket?