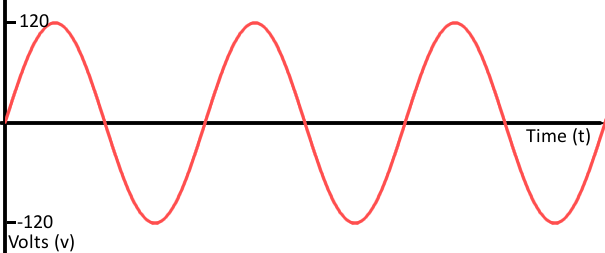
***SET – 03***

* Analog to Digital Converters:

In electronics, an analog to digital converter (ADC) is a system that converts an analog signal such as a sound picked up by microphone or light entering a digital camera to a digital signal.

First we have to understand what analog and digital signals are;

1. Analog Signals:

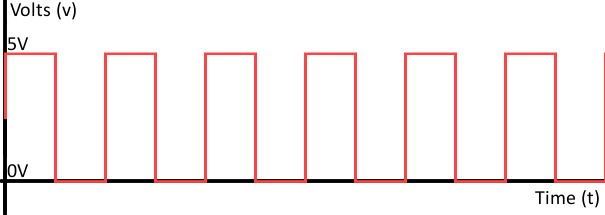


An analog signal is an continuous signal for which the time varying variable of the signal is a representation of some other time varying quantity.

In analog audio signal, the instantaneous voltage of the signal varies continuously with the pressure of the sound waves.

1. Digital Signals:

A digital signal is a discrete time signal for which not only the time but also the amplitude has discrete values. In other words, its samples take on only values from a discrete set. In this kind of a signal there are only two modes of operation, HIGH or LOW.



There are several ADC architectures. Due to the complexity and the need for precisely matched components. All but the most specialized ADCs are implemented as integrated circuits (ICs).

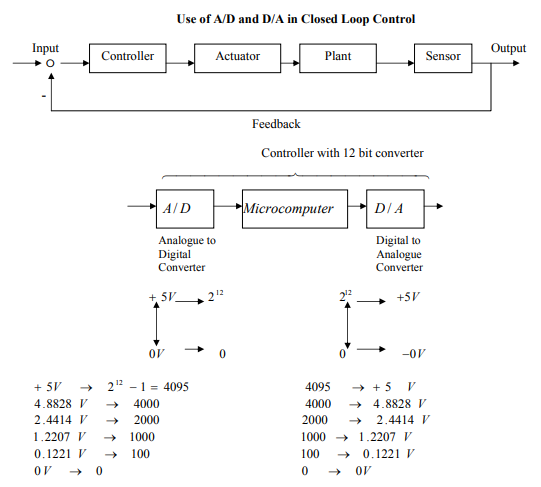
Analog information is transmitted by modulating a continuous transmission signal by amplifying a signal's strength or varying its frequency to add or take away data.

In the real world, most data is characterized by analog signals. In order to manipulate the data using a microprocessor, we need to convert analog signals to the digital signals, so that the microprocessor will be able to read, understand and manipulate data.

Digital information describes any system based on discontinuous data or events. Computers, which handle data in digital form, require analog-to-digital converters to turn signals from analog to digital before it can be read.

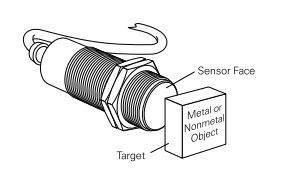
One example is a modem which turns signals from digital to analog before transmitting those signals over communication lines such as telephone lines that carry only analog signals.

The signals are turned back into digital form (demodulated) at the receiving end so that the computer can process the data in its digital format.



* Theory on working of inductive and capacitive proximity sensors:

1. Capacitive Sensors:



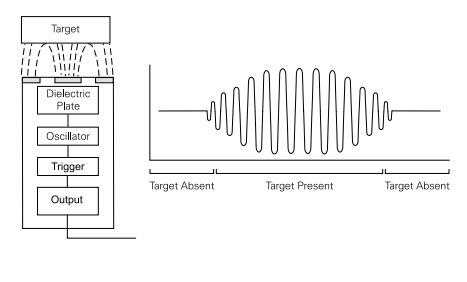
The sensing surface of a capacitive sensor is formed by two concentrically shaped metal electrodes of an unwound capacitor.

When an object nears the sensing surface it enters the electrostatic field of the electrodes and changes the capacitance in an oscillator circuit. As a result, the oscillator begins oscillating.

The trigger circuit reads the oscillator’s amplitude and when it reaches a specific level the output state of the sensor changes.

As the target moves away from the sensor the oscillator’s amplitude decreases, switching the sensor output back to its original state.

The working of a capacitive proximity sensor can be understood by the following diagram.

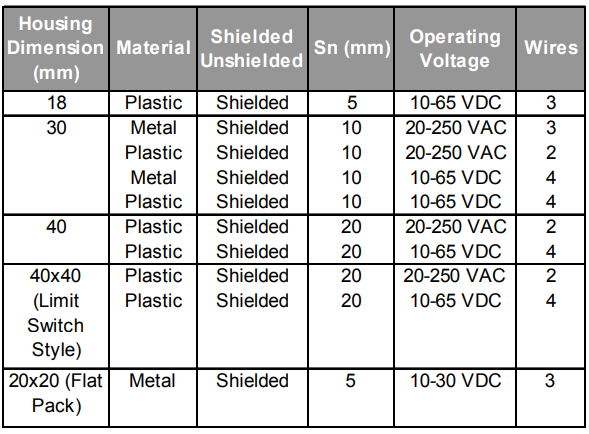


The main advantage of a capacitive proximity sensor is that it can sense a variety of materials like rubber, metal, plastics etc. whereas its counterpart, the inductive proximity sensor can only detect the presence of metallic objects as it sends out electromagnetic fields.

Also, the capacitive sensor has the ability to see through plastics and hence is used to detect the level of water in plastic bottles etc.

The 3RG16 product family identifies the Siemens capacitive proximity sensor. Units are available in DC or AC versions.

Electronic controls such as SIMATIC PLCs or relays can be controlled directly with the DC voltage version.

In the case of the AC voltage version the load (contactor relay, solenoid valve) is connected with the sensor in series directly to the AC voltage. Sensors are available with two-, three-, and four-wire outputs. 

1. Inductive Sensors:

-Available on robu.in and amazon.in for a price range of 250-450 INR

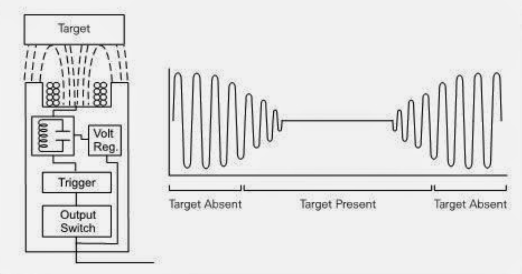
Inductive sensors detect the presence of metal objects—whether they are ferrous or nonferrous. They can be used to detect the presence or absence of parts, to count objects, or in positioning applications. Inductive sensors are often used instead of traditional limit switches because they can operate at higher speeds than mechanical switches. Inductive sensors are also more reliable because they are more robust.

Inductive sensors generate a high-frequency electromagnetic field. They are typically constructed using a coil and a ferrite core. When a target passes through the sensor’s magnetic field, the current induced on the target’s surface changes the characteristics of the oscillator that generates the field, causing it to lose energy. The sensor is designed to detect this energy loss as a transition, which in turn triggers a signal to actuate a solid-state output to either an “on” or “off” state. When the metal object exits the magnetic field, the oscillator regenerates, and the sensor returns to its normal state.

The output stage of a proximity sensor can be either analog or digital. Analog versions can be voltage (typically 0-10 V dc) or current (4-20 mA). They typically provide a linear signal to allow distance measurements of up to nearly 2 in. Digital outputs are designed to be used in dc-only circuits or in ac/dc circuits. Most versions are configured with normally-open outputs, but other versions can be normally closed, or can incorporate both a normally open and a normally closed output. The versions using the Namur output are intended for hazardous locations and must be used with the appropriate interface device for intrinsic safety ratings to apply.

Some advanced inductive sensors use multiple coils that enable them to detect all metals at the same range without requiring adjustments. Instead of a single coil inducing and being affected by eddy currents on a target, these inductive sensors use separate, independent sender and receiver coils. By detecting both ferrous and nonferrous components, these sensors provide an overall longer operating range.

Inductive sensors are well suited for detecting metallic objects in machinery and automation equipment. Some inductive sensors are inherently immune to magnetic field interaction, making them useful in applications where alternative technologies would fail due to interference with magnetic fields, such as welding, lifts, and electric arc furnaces.



***Communication Protocols***

Communication Protocols:

A set of rules and regulations that allow two electronic devices to connect to exchange the data with one and another.

1. Ultrasonic Sensors:

An ultrasonic sensor is a sensor which emits an ultrasonic wave (~40000Hz) which travels through air and then when it hits an obstacle, the wave reflects back to the source.

Once the reflected wave reaches the source again, we calculate the distance of the object by multiplying the speed of sound in air with the total time taken to receive the signal.

Moreover, the total time taken is actually twice the time taken as the wave has to travel to the object and then reflect back again.

Hence the final distance is to be halved and the result is the final distance.

An ultrasonic sensor module consists of 4 pins namely, Gnd, VCC, Trig, Echo.

The Gnd and the VCC pins have to be connected to the Gnd and the 5V pins on the Arduino UNO board respectively.

The Trig and the Echo pins can be connected to any of the specified digital I/O pins of the board.

From the coding point of view, we have to first specify the Trig and echo pins on the board , in this case they are 9 and 10.

***CODE:***

const int trigPin = 9;

const int echoPin = 10;

// defines variables

long duration;

int distance;

void setup() {

pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output

pinMode(echoPin, INPUT); // Sets the echoPin as an Input

pinMode(LED\_BUILTIN, OUTPUT);

Serial.begin(9600); // Starts the serial communication

}

void loop() {

// Clears the trigPin

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

// Sets the trigPin on HIGH state for 10 micro seconds

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

// Reads the echoPin, returns the sound wave travel time in microseconds

duration = pulseIn(echoPin, HIGH);

// Calculating the distance

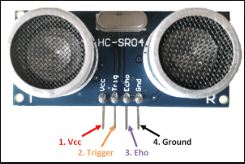
distance= duration\*0.034/2;

// Prints the distance on the Serial Monitor

Serial.print("Distance: ");

Serial.println(distance);

}

 Available online for a range of Rs.100-300

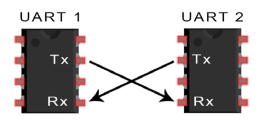
1. UART:

UART – Universal Asynchronous Receiver/Transmitter.

These are used to transmit signals from one UART to another UART and then further to a Data bus.

Unlike SPI, these are one-one communication devices and do not have multiple masters or slaves.

A main advantage of an UART is that it requires only 2 wires to transmit data between two UARTs. Data flows from the Tx pin f the transmitting UART to the Rx pin of the receiving UART.



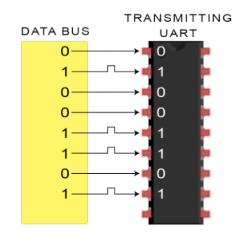
In a UART, there is no clock signal to synchronize the output, hence it is said to be working asynchronously.

So to make this work, a start bit and a stop bit is used to enclose the data so that the UART knows when to start reading the data.

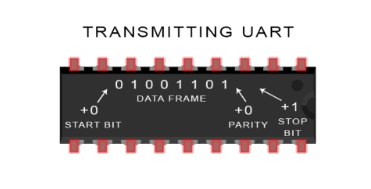
Baud rate is the measure of speed of data transfer in bits per second (bps).

Steps of UART Transmission:

1. The transmitting UART receives data in the parallel from the data bus.



1. The transmitting UART adds the Start bit, Parity bit(optional), and the Stop bit to the data frame/.



1. The entire data packet is then sent serially from the Tx of the transmitting UART to the Rx of the receiving UART.
2. The receiving UART discards the Start, Parity and the stop bit from the data packet and then converts the serial signal into parallel signal and then transmits it to the data bus in parallel

Advantages:

1. Only uses two wires.
2. No clock signal is required.
3. Has a parity bit to allow for error checking.
4. The structure of the data packet can be changed as long as both sides are set up for it.
5. Well documented and widely used method.

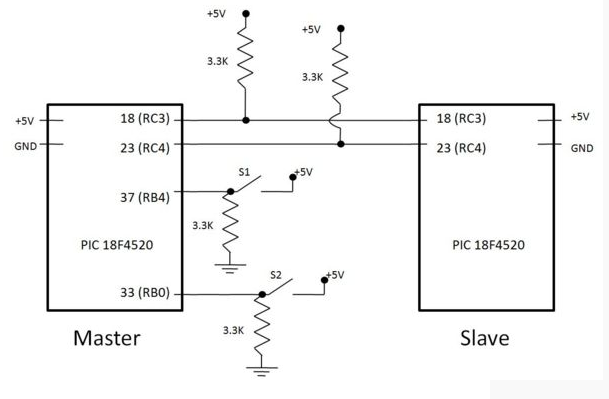
Disadvantages:

1. The size of the data frame is limited to a max of 9 bits.
2. Doesn’t support multiple slave or multiple masters system
3. The baud rates of each UART must be within 10% of each other.
4. I2C:

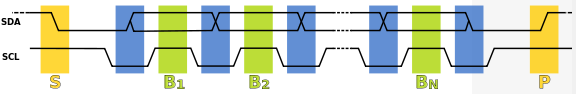
I2C is pronounced "I squared C" and stands for Inter-Integrated Circuit. This protocol was designed by Phillips Semiconductors around 1992 to allow easy communication between components on the same circuit board and can achieve transfer rates of up to 400 kbit/sec.

In this bus, it is possible for a single master to control 112 slaves, while there can also be multiple masters. Hence helping us to build a larger network.

Circuit Diagram:



Timing Diagram:



1. Data transfer is initiated with a *start* bit (S) signaled by SDA being pulled low while SCL stays high.
2. SCL is pulled low, and SDA sets the first data bit level while keeping SCL low (during blue bar time).
3. The data are sampled (received) when SCL rises for the first bit (B1). For a bit to be valid, SDA must not change between a rising edge of SCL and the subsequent falling edge (the entire green bar time).
4. This process repeats, SDA transitioning while SCL is low, and the data being read while SCL is high (B2 ...Bn).
5. The final bit is followed by a clock pulse, during which SDA is pulled low in preparation for the *stop* bit.
6. A *stop* bit (P) is signaled when SCL rises, followed by SDA rising.

In order to avoid false marker detection, there is a minimum delay between the SCL falling edge and changing SDA, and between changing SDA and the SCL rising edge. Note that an I²C message containing *N* data bits (including acknowledges) contains *N*+1 clock pulses.

CODE:

Below is an example of [bit-banging](https://en.wikipedia.org/wiki/Bit_banging) the I²C protocol as an I²C master. The example is written in [pseudo](https://en.wikipedia.org/wiki/Pseudo_code) [C](https://en.wikipedia.org/wiki/C_(programming_language)). It illustrates all of the I²C features described before (clock stretching, arbitration, start/stop bit, ack/nack).[[23]](https://en.wikipedia.org/wiki/I%C2%B2C#cite_note-23)

1 *// Hardware-specific support functions that MUST be customized:*

2 #define I2CSPEED 100

3 void I2C\_delay(void);

4 bool read\_SCL(void); *// Return current level of SCL line, 0 or 1*

5 bool read\_SDA(void); *// Return current level of SDA line, 0 or 1*

6 void set\_SCL(void); *// Do not drive SCL (set pin high-impedance)*

7 void clear\_SCL(void); *// Actively drive SCL signal low*

8 void set\_SDA(void); *// Do not drive SDA (set pin high-impedance)*

9 void clear\_SDA(void); *// Actively drive SDA signal low*

10 void arbitration\_lost(void);

11

12 bool started = false; *// global data*

13

14 void i2c\_start\_cond(void) {

15 **if** (started) {

16 *// if started, do a restart condition*

17 *// set SDA to 1*

18 set\_SDA();

19 I2C\_delay();

20 set\_SCL();

21 **while** (read\_SCL() == 0) { *// Clock stretching*

22 *// You should add timeout to this loop*

23 }

24

25 *// Repeated start setup time, minimum 4.7&nbsp;us*

26 I2C\_delay();

27 }

28

29 **if** (read\_SDA() == 0) {

30 arbitration\_lost();

31 }

32

33 *// SCL is high, set SDA from 1 to 0.*

34 clear\_SDA();

35 I2C\_delay();

36 clear\_SCL();

37 started = true;

38 }

39

40 void i2c\_stop\_cond(void) {

41 *// set SDA to 0*

42 clear\_SDA();

43 I2C\_delay();

44

45 set\_SCL();

46 *// Clock stretching*

47 **while** (read\_SCL() == 0) {

48 *// add timeout to this loop.*

49 }

50

51 *// Stop bit setup time, minimum 4us*

52 I2C\_delay();

53

54 *// SCL is high, set SDA from 0 to 1*

55 set\_SDA();

56 I2C\_delay();

57

58 **if** (read\_SDA() == 0) {

59 arbitration\_lost();

60 }

61

62 started = false;

63 }

64

65 *// Write a bit to I2C bus*

66 void i2c\_write\_bit(bool bit) {

67 **if** (bit) {

68 set\_SDA();

69 } **else** {

70 clear\_SDA();

71 }

72

73 *// SDA change propagation delay*

74 I2C\_delay();

75

76 *// Set SCL high to indicate a new valid SDA value is available*

77 set\_SCL();

78

79 *// Wait for SDA value to be read by slave, minimum of 4us for standard mode*

80 I2C\_delay();

81

82 **while** (read\_SCL() == 0) { *// Clock stretching*

83 *// You should add timeout to this loop*

84 }

85

86 *// SCL is high, now data is valid*

87 *// If SDA is high, check that nobody else is driving SDA*

88 **if** (bit && (read\_SDA() == 0)) {

89 arbitration\_lost();

90 }

91

92 *// Clear the SCL to low in preparation for next change*

93 clear\_SCL();

94 }

95

96 *// Read a bit from I2C bus*

97 bool i2c\_read\_bit(void) {

98 bool bit;

99

100 *// Let the slave drive data*

101 set\_SDA();

102

103 *// Wait for SDA value to be written by slave, minimum of 4us for standard mode*

104 I2C\_delay();

105

106 *// Set SCL high to indicate a new valid SDA value is available*

107 set\_SCL();

108

109 **while** (read\_SCL() == 0) { *// Clock stretching*

110 *// You should add timeout to this loop*

111 }

112

113 *// Wait for SDA value to be written by slave, minimum of 4us for standard mode*

114 I2C\_delay();

115

116 *// SCL is high, read out bit*

117 bit = read\_SDA();

118

119 *// Set SCL low in preparation for next operation*

120 clear\_SCL();

121

122 **return** bit;

123 }

124

125 *// Write a byte to I2C bus. Return 0 if ack by the slave.*

126 bool i2c\_write\_byte(bool send\_start,

127 bool send\_stop,

128 unsigned char byte) {

129 unsigned bit;

130 bool nack;

131

132 **if** (send\_start) {

133 i2c\_start\_cond();

134 }

135

136 **for** (bit = 0; bit < 8; ++bit) {

137 i2c\_write\_bit((byte & 0x80) != 0);

138 byte <<= 1;

139 }

140

141 nack = i2c\_read\_bit();

142

143 **if** (send\_stop) {

144 i2c\_stop\_cond();

145 }

146

147 **return** nack;

148 }

149

150 *// Read a byte from I2C bus*

151 unsigned char i2c\_read\_byte(bool nack, bool send\_stop) {

152 unsigned char byte = 0;

153 unsigned char bit;

154

155 **for** (bit = 0; bit < 8; ++bit) {

156 byte = (byte << 1) | i2c\_read\_bit();

157 }

158

159 i2c\_write\_bit(nack);

160

161 **if** (send\_stop) {

162 i2c\_stop\_cond();

163 }

164

165 **return** byte;

166 }

167

168 void I2C\_delay(void) {

169 **volatile** int v;

170 int i;

171

172 **for** (i = 0; i < I2CSPEED / 2; ++i) {

173 v;

174 }

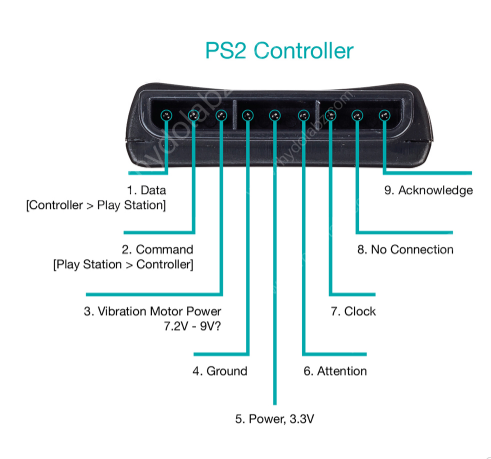
175 }

Advantages:

 •Due to open collector design, limited slew rates can be achieved.   
 •More than one masters can be used in the electronic circuit design.   
 •Needs fewer i.e. only 2 wires for communication.   
 •I2C addressing is simple which does not require any CS lines used in SPI and it is easy to add extra devices on the bus.   
 •It uses open collector bus concept. Hence there is bus voltage flexibity on the interface bus.   
 •Uses flow control

1. PS2 Controller Interfacing:

* The PS2 wireless controller is a standard controller for the PlayStation 2 and is identical to the original DualShock controller for the PlayStation console.
* It features twelve analog (pressure-sensitive) buttons (Χ, O, Π, Δ, L1, R1, L2, R2, Up, Down, Left and Right), five digital button (L3, R3 Start, Select and the analog mode button) and two analog sticks.
* The controller also features two vibration motors, the left one being larger and more powerful than the one on the right.
* It is powered by two AAA batteries. It communicates with the console using 2.4 GHz RF protocol.



1. **DATA:** This is the data line from Controller to PS2. This is an open collector output and requires a pull-up resistor (1 to 10k, maybe more). (A pull-up resistor is needed because the controller can only connect this line to ground; it can’t actually put voltage on the line).
2. **COMMAND:** This is the data line from PS2 to Controller.
3. **VIBRATION MOTOR POWER**
4. **GND:** Ground
5. **VCC:** VCC can vary from 5V down to 3V .
6. **ATT:** ATT is used to get the attention of the controller. This line must be pulled low before each group of bytes is sent / received, and then set high again afterwards. This pin consider as “Chip Select” or “Slave Select” line that is used to address different controllers on the same bus.
7. **CLK:**500kH/z, normally high on. The communication appears to be SPI bus.
8. **Not Connected**
9. **ACK:**Acknowledge signal from Controller to PS2. This normally high line drops low about 12us after each byte for half a clock cycle, but not after the last bit in a set. This is a [open collector output](http://en.wikipedia.org/wiki/Open_collector) and requires a pull-up resistor (1 to 10k, maybe more).

**PS2 Signals:**

PS2 wireless controller communicates with Arduino using a protocol that is basically SPI. The play station sends a byte at the same time as it receives one (full duplex) via serial communication. There’s a clock (SCK) to synchronize bits of data across two channels: DATA and CMD. Additionally, there’s a “Attention” (ATT) channel which tells the slave whether or not it is “active” and should listen to data bits coming across the CMD channel, or send data bits across the DATA channel (Reasonably, only one slave device should be active at a time) . The PlayStation 2 actually uses this plus an additional line that is not specifically part of the SPI protocol – an “Acknowledge” (ACK) line.

The clock is held high until a byte is to be sent. It then drops low (active low) to start 8 cycles during which data is simultaneously sent and received. The logic level on the data lines is changed by the transmitting device on the falling edge of clock. This is then read by the receiving device on the rising edge allowing time for the signal to settle. After each Command is received from the controller, that controller needs to pull ACK low for at least one clock cycle. If a selected controller does not ACK the PS2 will assume that there is no controller present. LSBs (least significant bits) are transmitting first.

1. Basic Wireless Communication Protocols:

* LTE:

 Phone carriers and the industry have invested large sums of money to upgrade cell towers and other network equipment to support 4G, standardizing on a communication protocol called [Long Term Evolution (LTE)](https://www.lifewire.com/definition-of-lte-broadband-817465) that emerged as a popular service starting in 2010.

[LTE technology](https://www.lifewire.com/lte-long-term-evolution-817470) was designed to significantly improve the low data rates and roaming issues with older phone protocols. The protocol can carry more than 100 Mbps of data, although the [network bandwidth](https://www.lifewire.com/what-is-bandwidth-2625809) is normally regulated to levels below 10 Mbps for individual users. Due to the significant cost of equipment, plus some government regulatory challenges, phone carriers have not yet deployed LTE in many locations. LTE is also not suitable for home and other [local area networking](https://www.lifewire.com/local-area-network-816382), being designed to support a larger number of customers across much longer distances (and corresponding higher cost).

* WiFi:

[Wi-Fi](https://www.lifewire.com/what-is-wi-fi-2377430) is widely associated with [wireless networking](https://www.lifewire.com/introduction-to-wi-fi-wireless-networking-818265) as it has become the de facto standard for home networks and public [hotspot](https://www.lifewire.com/definition-of-a-hotspot-816546) networks. Wi-Fi became popular starting in the late 1990s as the networking hardware required to enable PCs, printers, and other consumer devices became widely affordable and the supported data rates were improved to acceptable levels (from 11 [Mbps](https://www.lifewire.com/bits-per-second-kbps-mbps-gbps-818122) to 54 Mbps and above).

Although Wi-Fi can be made to run over longer distances in carefully controlled environments, the protocol is practically limited to work within single residential or commercial buildings and outdoor areas within short walking distances. [Wi-Fi speeds](https://www.lifewire.com/why-do-wifi-connection-speeds-keep-changing-818295) are also lower than for some other wireless protocols. Mobile devices increasingly support both Wi-Fi and LTE (plus some older cellular protocols) to give users more flexibility in the kinds of networks they can use.

[Wi-Fi Protected Access](https://www.lifewire.com/definition-of-wifi-protected-access-816576) security protocols add network authentication and data encryption capabilities to Wi-Fi networks. Specifically, [WPA2](https://www.lifewire.com/what-is-wpa2-818352) is recommended for use on home networks to prevent unauthorized parties from logging into the network or intercepting [personal data](https://www.lifewire.com/protecting-personal-data-on-computer-networks-817992) sent over the air

* Bluetooth:

One of the oldest wireless protocols still broadly available, Bluetooth was created in the 1990s to synchronize data between phones and other battery-powered devices. Bluetooth requires a lower amount of power to operate than Wi-Fi and most other wireless protocols. In return, Bluetooth connections only function over relatively short distances, often 30 feet (10 m) or less and support relatively low data rates, usually 1-2 Mbps. Wi-Fi has replaced Bluetooth on some newer equipment, but many phones today still support both of these protocols.

* 60 GHz Protocols – WirelessHD and WiGig:

One of the most popular activities on computer networks is streaming of video data, and several wireless protocols that run on 60 [Gigahertz (GHz)](https://www.lifewire.com/story-of-hertz-megahertz-and-gigahertz-818308) frequencies have been built to better support this and other usages which require large amounts of network bandwidth. Two different industry standards called WirelessHD and WiGig were created in the 2000s both using 60 GHz technology to support high-bandwidth wireless connections: WiGig offers between 1 and 7 Gbps of bandwidth while WirelessHD supports between 10 and 28 Gbps.

Although basic video streaming can be done over Wi-Fi networks, best quality high-definition video streams demand the higher data rates these protocols offer. The very high signaling frequencies of WirelessHD and WiGig compared to Wi-Fi (60 GHz versus 2.4 or 5 GHz) greatly limit connection range, generally shorter than Bluetooth, and typically to within a single room (as 60 GHz signals do not penetrate walls effectively)

* Wireless Home Automation Protocols- Z-Wave and Zigbee

Various network protocols have been created to support home automation systems that allow remote control of lights, home appliances, and consumer gadgets. Two prominent wireless protocols for home automation are Z-Wave and Zigbee. To achieve the extremely low energy consumption required in home automation environments, these protocols and their associated hardware support only low data rates - 0.25 Mbps for Zigbee and only about 0.01 Mbps for Z-Wave. While such data rates are obviously unsuitable for general-purpose networking, these technologies work well as interfaces to consumer gadgets which have simple and limited communication requirements.