Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

Smart IoT Switching System

Tanuj Saini*, Dr. Jaimeel Shah†, Nisarg Chaudhary*, Aayush Brahmbhatt*, Harsh Gajjar*
*Department of Computer Science & Engineering, Parul University, Vadodara, Gujarat, India
†Associate Professor (Guide), Parul University, Vadodara, Gujarat, India

***Abstract*—Smart-home switches are usually controlled through a cloud service, which makes basic operation depend on Internet access and adds delay that is rarely quantified for low-cost designs. This paper presents a four-channel IoT switching system in which a single ESP32 microcontroller drives a relay module from three input paths: physical push buttons, the ESP RainMaker mobile application, and voice commands through Amazon Alexa and Google Home. All paths share one relay-control routine, and the firmware reports every relay change to the cloud so that the remote interfaces follow the physical state. The prototype was evaluated in 80 timed trials across all four channels. All 80 commands switched the intended channel, with no wrong-channel operations. Median latency was 9 ms for push buttons, 565 ms for the mobile application, and 1.90 s and 1.95 s for Alexa and Google Home, measured until the client showed the updated state. With the Internet link removed, the push buttons continued to operate the relays while application and voice control stopped, and the device was back online 6.7 s on average after the link was restored. The results quantify the cost of each cloud stage and support keeping a local control path in low-cost smart switches.**

***Index Terms*—Internet of Things, home automation, ESP32, ESP RainMaker, relay switching, voice assistants, latency**

I. Introduction

The Internet of Things (IoT) connects microcontrollers, sensors, and actuators to network services so that physical devices can be monitored and controlled remotely [1]–[Home automation is one of its most common applications. Earlier systems used fixed or dedicated control links, while current designs rely on Wi-Fi, mobile applications, cloud platforms, and commercial voice assistants [4], [5].

Moving control into the cloud brings convenience but also new dependencies. Smart-home studies identify reliability, interoperability, privacy, and dependence on external services as key concerns [5]–[A switch that can only be operated through a cloud service stops working when the Internet connection fails, and every additional service in the command path adds delay. For low-cost student and hobbyist designs these effects are usually described qualitatively, and the delay of each control path is rarely measured.

This paper addresses that gap with a four-channel switching system built around one ESP32. Physical push buttons give local control, ESP RainMaker gives mobile control, and Amazon Alexa and Google Home give voice control. All inputs operate the same four relay outputs. The contributions are:

- a low-cost design that combines a cloud-independent local path with managed cloud control and two commercial voice assistants through one shared relay-control routine and synchronized device state;
- a timed evaluation of all four control methods across the four channels;
- a measured comparison of latency, success rate, offline behaviour, and state consistency across the local, cloud, and voice paths.

Section II reviews related work. Sections III and IV describe the design and implementation. Section V gives the test method, Section VI the results, and Section VII the limitations and future work. Section VIII concludes the paper.

II. Related Work

Internet and GSM control. Alkar and Buhur built an Internet-based wireless home automation system for multi-functional devices [8]. Yuksekkaya et al. combined GSM, Internet, and speech commands in one system [9]. Both established remote and speech control as practical, but relied on their own network and application structure rather than a managed cloud platform.

Wireless and sensor-based systems. Gill et al. used a ZigBee network with a home gateway [10]. Darianian and Michael proposed RFID-based smart-home services [11], and Vujovic and Maksimovic used a Raspberry Pi as a sensor web node [12]. These works emphasise sensing and wireless coverage rather than low-cost multi-path switching.

ESP32 and MQTT. Wi-Fi microcontrollers reduced the hardware needed for IoT switches. Thirupathi and Sagar implemented home automation with an ESP32 and MQTT [13], a publish–subscribe protocol designed for constrained devices [3], [16]. Haroon et al. combined ESP32 Wi-Fi control with environmental monitoring [14]. Communication options for such systems are reviewed in [15].

Gap. Table I compares these systems. Most focus on one remote protocol or interface. Few combine a local push-button path, a managed cloud platform, and more than one commercial voice assistant on one low-cost controller, and none of them report the measured latency of each path or the behaviour of the system when the Internet link fails.

III. System Design

A. Architecture

The system is organised into three planes (Fig. 1). The *interaction plane* holds the user inputs: the push buttons, the RainMaker application, and the two voice assistants.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

TABLE I
COMPARISON OF RELATED WORK

System	Local	Remote path	Voice	Latency
Alkar & Buhur [8]	Yes	Internet	No	No
Yuksekkaya et al. [9]	Yes	GSM/Internet	Custom	No
Gill et al. [10]	Yes	ZigBee	No	No
Thirupathi & Sagar [13]	Yes	MQTT	No	No
Haroon et al. [14]	Partial	Wi-Fi	No	No
This work	**Yes**	**Cloud**	**Alexa, Google**	**Yes**

The *coordination plane* holds the RainMaker cloud and the ESP32, which carry commands and manage device state. The *actuation plane* holds the four-channel relay module and its loads (fan, light, TV, and AC). The two resulting command paths are

Local: Button → ESP32 → Relay,

Remote: App/Voice → Cloud → ESP32 → Relay.

Voice commands add the assistant's own service in front of the RainMaker cloud, so they contain one more network stage than application commands.

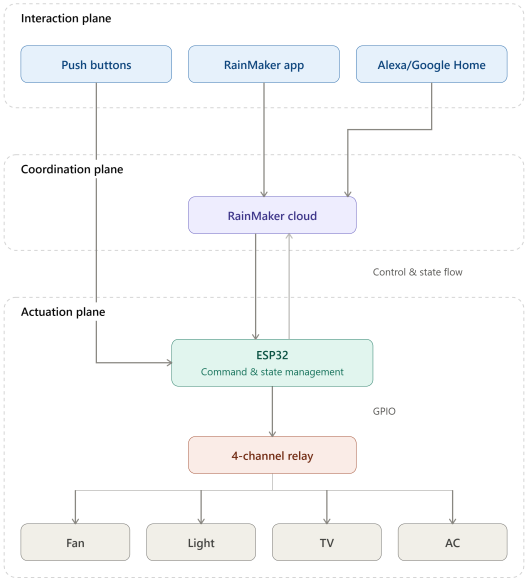


Fig. 1. Functional architecture of the proposed switching system.

B. Hardware

The ESP32 provides the GPIO lines for the buttons and relay inputs and integrated 2.4 GHz Wi-Fi [20]. A four-channel relay module isolates the low-voltage controller from the AC loads; each channel has its own control input, so channels switch independently. A 5 V supply powers the controller and relay module. Low-voltage wiring is kept physically separate from the AC side, and each load must stay within the relay's rated voltage and current. Four push buttons, one per channel, connect to GPIO inputs and give local control that does not depend on Wi-Fi or the cloud. The design uses eight GPIO pins; the ESP32-WROOM-32 has about 22 usable general-purpose pins, which leaves room for 8–12 channels before an I/O expander is needed.

C. Firmware and Protocols

The firmware initialises the GPIO pins, connects to Wi-Fi, and registers the board as a RainMaker node in which each relay is a device with a power parameter [21], [22]. Button events and remote commands call the same relay-control routine, so there is one switching path and one stored state per channel.

Device-to-cloud communication uses MQTT [16] over TLS [18], which protects commands against eavesdropping and modification; this matters because a forged command can change a physical load [19]. The mobile application and voice integrations reach the cloud through HTTP-based REST interfaces [17], so the ESP32 handles only the device side of the connection. Speech recognition is done by Alexa or Google Home; the ESP32 receives only the resulting device command, which keeps processing on the microcontroller small.

D. State Synchronisation

A button press changes a relay without any cloud action, so the remote interfaces could show a stale state. To prevent this, the firmware publishes the new parameter value after every relay change, whether the change came from a button or a remote command. The application and voice assistants therefore read the state reported by the device rather than the last command sent.

IV. IMPLEMENTATION

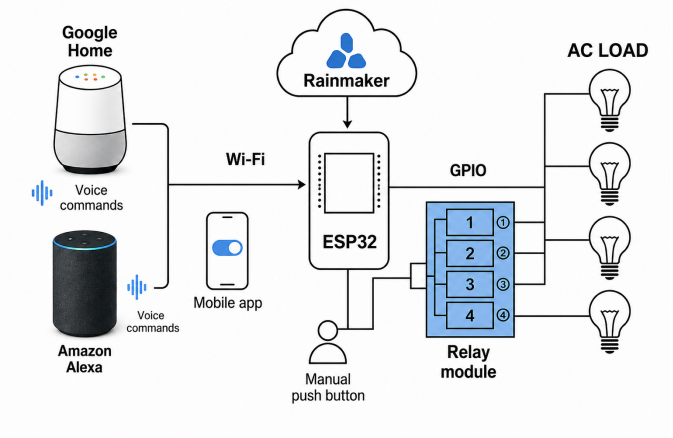


Fig. 2. Implementation overview: control inputs, ESP32, relay module, and loads.

Fig. 2 summarises the implemented system and Fig. 3 shows the prototype. The ESP32 drives the four relay inputs from four GPIO outputs, and the push buttons connect to four GPIO inputs. During provisioning the board joins the 2.4 GHz network and is linked to the user's RainMaker account; the four channels then appear as devices in the

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

application. Alexa and Google Home were linked through the RainMaker integrations, after which each channel could be addressed by name. Jumper wiring on a breadboard was used for the low-voltage circuit. The bill of materials cost about INR 500–800, compared with roughly INR 5,000 for commercial four-channel smart switches.

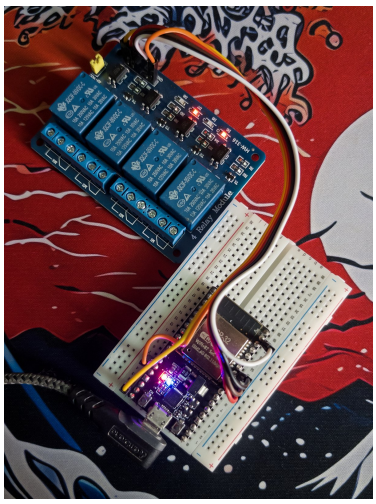


Fig. 3. Assembled prototype: ESP32 and four-channel relay module.

V. Test Method

A. Setup

Table II lists the test conditions. All tests used the same network, device placement, and voice phrases.

TABLE II
Test Conditions

Parameter	Value
Controller	ESP32 DevKit (ESP-WROOM-32, 38-pin)
Relay module	5 V, four-channel
Wi-Fi band / RSSI at device	2.4 GHz / −30 to −60 dBm
Internet connection	200 Mbit/s
Alexa client	Echo Dot (4th gen) and Alexa app
Google client	Nest Mini and Google Home app
Trials per method	20 (all four channels)

B. Latency Measurement

Latency was recorded with a software timer for 20 trials of each control method, spread across the four channels. The start and end points depend on the control path:

- *Push button:* from the button press to the connected light turning on.
- *RainMaker application:* from tapping the switch in the application to the updated status appearing in the application.
- *Alexa and Google Home:* from issuing the command to the updated status appearing in the Alexa or Google Home application.

For the three remote methods, the end point is the state shown by the client. The measured time therefore covers the complete round trip: the command travelling to the device through the cloud, the relay switching, and the new state being reported back to the client.

C. Success Rate, Offline Operation, and State Consistency

A trial counted as successful when the intended channel, and only that channel, changed state. Offline behaviour was tested by disconnecting the router's Internet (WAN) link while keeping the local Wi-Fi network active; 10 button presses and 5 commands per remote method were issued, and the controller was watched for resets over 20 such tests. Recovery time was measured with a stopwatch from reconnecting the WAN link until the device showed as online in the RainMaker application (3 repetitions). State consistency was checked by pressing the physical buttons 10 times and comparing the application state with the physical relay, and the button-to-application update delay was timed for 3 of these presses.

VI. Results

A. Functional Operation and Success Rate

All four channels responded to all four control methods. All 80 timed commands (20 per method) switched the intended channel, and no command changed a different channel (Table III).

TABLE III
Success Rate and Latency by Control Method

Method	n	Success	Median (ms)	Mean (ms)	SD (ms)	P95 (ms)	Max (ms)
Push button	20	20/20	9	8.9	2.1	12	13
RainMaker app	20	20/20	565	574	220	891	892
Amazon Alexa	20	20/20	1900	1908	381	2476	2492
Google Home	20	20/20	1946	1912	380	2428	2491

B. Latency

Table III and Fig. 4 summarise the measured latency. Button control was the fastest, with a median of 9 ms and a maximum of 13 ms, because the command never leaves the device. The RainMaker application had a median round-trip latency of 565 ms (range 247–892 ms), which reflects the Wi-Fi and cloud stages in both directions. Voice control was the slowest: Alexa had a median of 1900 ms and Google Home 1946 ms, and both ranged from about 1.26 s to 2.49 s. The two assistants performed almost identically, which suggests that the extra delay comes mainly from speech processing and the assistant-to-RainMaker hand-off rather than from either assistant specifically. The cloud paths also varied far more than the local path: the standard deviation was 2.1 ms for the buttons, 220 ms for the application, and about 380 ms for each voice assistant.

C. Offline Operation and Recovery

Table IV shows the results with the Internet link removed. The push buttons operated the relays in all 10 attempts because the local path does not use the network. Application and voice

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

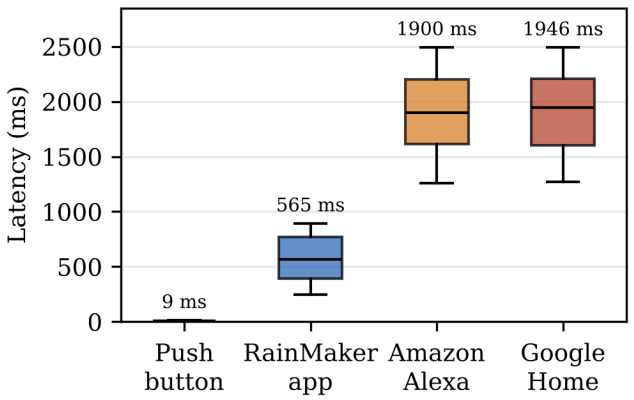


Fig. 4. Latency distribution for each control method (20 trials each; labels show the median).

commands failed in every attempt. The ESP32 did not reset or crash in any of the 20 offline tests. After the Internet link was restored, the device appeared online again after 5, 9, and 6 s (mean 6.7 s), and remote control resumed without a reboot.

TABLE IV
OPERATION WITHOUT INTERNET ACCESS (WI-FI ACTIVE)

Method	Successful commands
Push button	10/10
RainMaker app	0/5
Amazon Alexa	0/5
Google Home	0/5
Controller resets in 20 offline tests	0
Recovery time after WAN restored	5, 9, 6 s (mean 6.7 s)

D. State Consistency

After all 10 button presses, the RainMaker application showed the same state as the physical relay. The application updated 1.0, 1.3, and 1.2 s after the button press (mean 1.17 s).

E. Discussion

Each network stage adds a clear and measurable delay. Moving from local to cloud control raised the median latency from 9 ms to 565 ms, and adding a voice assistant raised it again to about 1.9 s. Even so, all methods responded within about 2.5 s, which is acceptable for switching household loads. The main practical difference appeared during the outage test: only the local buttons kept working. Keeping a direct local path alongside cloud and voice control therefore preserves basic operation when the Internet or an external service fails, at no extra hardware cost.

VII. LIMITATIONS AND FUTURE WORK

Mobile and voice control depend on the Internet and the RainMaker service. The measurements come from one network and location with 20 trials per method, so latency may differ under other conditions. The mechanical relays have limited contact life and can only switch loads within their rating. The system does not measure current or energy and cannot detect a failed load, and the breadboard wiring is not suitable for permanent installation.

Future work will add a local network interface (for example, a local web server or ESP-NOW) so that the application works without the Internet, current sensing for energy monitoring and fault detection, scheduling and sensor-based automation, and a custom PCB with an enclosure. Latency will also be measured across different networks and times of day.

VIII. CONCLUSION

This paper presented a four-channel IoT switching system in which one ESP32 combines push-button, mobile, and voice control through a shared relay-control routine with synchronised state. All 80 timed commands switched the intended channel. Median latency rose from 9 ms for local control to 565 ms for the mobile application and about 1.9 s for Alexa and Google Home, and only the local path kept working without the Internet. The device recovered remote operation 6.7 s on average after the link returned, and the application followed button-initiated changes within about 1.2 s. These measurements show the delay that each cloud and voice stage adds and support including a local control path in low-cost smart switches.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 2015.
[4] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, "A review of smart homes—Past, present, and future," *IEEE Trans. Syst., Man, Cybern. C*, vol. 42, no. 6, pp. 1190–1203, 2012.
[5] B. L. R. Stojkoska and K. V. Trivodaliev, "A review of Internet of Things for smart home: Challenges and solutions," *J. Cleaner Production*, vol. 140, pp. 1454–1464, 2017.
[6] C. Wilson, T. Hargreaves, and R. Hauxwell-Baldwin, "Smart homes and their users: A systematic analysis and key challenges," *Personal and Ubiquitous Computing*, vol. 19, no. 2, pp. 463–476, 2015.
[7] C. Wilson, T. Hargreaves, and R. Hauxwell-Baldwin, "Benefits and risks of smart home technologies," *Energy Policy*, vol. 103, pp. 72–83, 2017.
[8] A. Z. Alkar and U. Buhur, "An Internet based wireless home automation system for multifunctional devices," *IEEE Trans. Consum. Electron.*, vol. 51, no. 4, pp. 1169–1173, 2005.
[9] B. Yuksekkaya, A. A. Kayalar, M. B. Tosun, M. K. Ozcan, and A. Z. Alkar, "A GSM, Internet and speech controlled wireless interactive home automation system," *IEEE Trans. Consum. Electron.*, vol. 52, no. 3, pp. 837–843, 2006.
[10] K. Gill, S.-H. Yang, F. Yao, and X. Lu, "A ZigBee-based home automation system," *IEEE Trans. Consum. Electron.*, vol. 55, no. 2, pp. 422–430, 2009.
[11] M. Darianian and M. P. Michael, "Smart home mobile RFID-based Internet-of-Things systems and services," in *Proc. Int. Conf. Advanced Computer Theory and Engineering*, 2008, pp. 116–120.
[12] V. Vujovic and M. Maksimovic, "Raspberry Pi as a sensor web node for home automation," *Computers and Electrical Engineering*, vol. 44, pp. 153–171, 2015.
[13] V. Thirupathi and K. Sagar, "Implementation of home automation system using MQTT protocol and ESP32," *Int. J. Engineering and Advanced Technology*, vol. 8, no. 2C2, pp. 111–113, 2018.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 11 , November - 2026

[14] A. L. Haroon P. S., M. Shafiulla, S. M. Nawaz, S. M. Naveed, and S. Ahmed, "Home automation using Wi-Fi: ESP32-based system for remote control and environmental monitoring," in *Proc. 3rd Int. Conf. Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, 2024.
[15] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of Things (IoT) communication protocols: Review," in *Proc. 8th Int. Conf. Information Technology (ICIT)*, 2017, pp. 685–690.
[16] OASIS, *MQTT Version 5.0*, OASIS Standard, Mar. 2019.
[17] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message syntax and routing," RFC 7230, IETF, Jun. 2014.
[18] E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.3," RFC 8446, IETF, Aug. 2018.
[19] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.
[20] Espressif Systems, *ESP32 Series Datasheet*.
[21] Espressif Systems, *ESP RainMaker Documentation*.
[22] Espressif Systems, "Provisioning," *ESP RainMaker Documentation*.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)