Automated WiFi Network Switching: Seamless Connectivity for the Modern Office

Anurag Sharma 5 min read · Oct 20, 2025

Introduction

In today's fast-paced office environment, reliable WiFi connectivity is critical for productivity. Our team at a multi-access point in our **new setup office** faced frequent connectivity issues due to manual WiFi switching when moving between areas. Employees experienced disconnections during meetings, auto-connect failures, and productivity losses from troubleshooting. To address these challenges, we developed a Small Python-based automated WiFi switching solution that ensures seamless connectivity by intelligently selecting the best network based on signal strength, speed, and availability.

Technical Challenge

Our new office setup consists of multiple WiFi access points with distinct SSIDs, but no extenders are used for seamless roaming. This led to several pain points:

- **Frequent Disconnections:** Moving between meeting rooms caused network drops.
- **Manual Switching:** Employees had to select networks, disrupting workflows manually.
- **Auto-Connect Issues:** Saved credentials often failed to reconnect reliably.

- **Productivity Loss**: Time spent troubleshooting connectivity impacted efficiency.

The challenge was to create a temporary solution that monitors available networks, evaluates their quality, and automatically switches to the optimal connection without user intervention.

Solution Design

Our solution is a Python application that runs in the background, continuously monitoring WiFi networks and switching to the best available option.

The architecture includes:

- **Network Scanning**: Identifies available WiFi networks and their properties.
- **Quality Assessment**: Evaluates networks based on signal strength, connection speed, and ping reliability.
- **Switching Logic**: Automatically connects to the best network when the current connection degrades.
- **Background Operation**: Runs as a lightweight service with minimal resource usage.
- **Error Handling**: Manages connection failures and ensures graceful recovery.

The application uses system commands (`netsh` for Windows) to interact with the WiFi adapter and external libraries for speed testing.

Implementation

Below are key components of the solution, with code snippets from our implementation.

1. Network Scanning and Discovery

The `getCurrentWifi` function retrieves the currently connected WiFi network using the `netsh wlan show interfaces` command.

```
def getCurrentWifi(wifiList):
    cmd = 'netsh wlan show interfaces'
    p = subprocess.Popen(cmd, stdin=subprocess.PIPE, stdout=subprocess.PIPE, std
    ret = p.stdout.read()
    for index in range(len(wifiList)):
        if ret.find(wifiList[index].encode()) >= 0:
            return index
    return 0
```

2. Connection Quality Assessment

We assess network quality using ping tests (`pingCheck`) and speed tests (`getSpeedinMB`). The `pingCheck` function verifies network availability, while `getSpeedinMB` measures download and upload speeds using the `speedtest` library.

```
def pingCheck(ip, count=1, timeout=1000):
    cmd = f'ping -n {count} -w {timeout} {ip} > NUL'
    response = os.system(cmd)
    return "ok" if response == 0 else "failed"
```

```
def getSpeedinMB(WifiName):
    import speedtest
    KB = 1024
    MB = KB * 1024
    speed_test = speedtest.Speedtest()
    download_speed = speed_test.download()
    upload_speed = speed_test.upload()
    print(f"Wifi Connection | {WifiName} | Download speed is {int(download_speed
    return [WifiName, [download_speed, upload_speed]]
```

3. Connection Switching Logic

The `connectToBestWifi` function evaluates all available networks, connects to each to measure speed, and selects the best based on download/upload performance. The `WifiAutoSwitch` function monitors the current connection and triggers a switch if the ping fails.

```
def connectToBestWifi(systemWifiList, req):
    speedList = {}
    for wifi in systemWifiList:
        wifiName = wifi.split(":")[0]
        ipAddress = wifi.split(":")[1]
        connectWifi(wifiName)
        getWifiIsAuthenticated(wifiName)
        time.sleep(3)
        speedOfWifi = getSpeedinMB(wifiName)
        speedList[speedOfWifi[0]] = speedOfWifi[1]
    bestWifi = getBestWifi(speedList)
    connectWifi(bestWifi)
    return bestWifi

def WifiAutoSwitch(wifiDictIP, systemWifiList, wifiListWithIp):
    lastConnectedWifi = systemWifiList[getCurrentWifi(systemWifiList)]
    ChangeIP = wifiDictIP.get(lastConnectedWifi)
    while True:
        time.sleep(1)
        now = datetime.now()
        pingStatus = pingCheck(ChangeIP, 2)
        if pingStatus != 'ok':
```

```
            print('Connection Failed/Interrupted with Wifi')
            reCheckWifiList = [x for x in wifiListWithIp if x.split(":")[0] != l
            if reCheckWifiList:
                nextWifi = connectToBestWifi(reCheckWifiList, 'reconnect')
                print(f'---Wifi Auto Switch from "{lastConnectedWifi}" to "{next
                lastConnectedWifi = nextWifi
                ChangeIP = wifiDictIP.get(nextWifi)
                connectWifi(nextWifi)
                time.sleep(3)
            else:
                print('System has No Wifi to Connect! Terminate!')
                break
```

4. Security Check

The `getWifiIsAuthenticated` function ensures the connected network uses secure authentication (e.g., WPA2-Personal), disconnecting from insecure networks.

```
def getWifiIsAuthenticated(Wifi):
    time.sleep(2)
    cmd = 'netsh wlan show interfaces'
    p = subprocess.Popen(cmd, stdin=subprocess.PIPE, stdout=subprocess.PIPE, std
    ret = p.stdout.read()
    if ret.find(b'WPA2-Personal') >= 0:
        print(f'{Wifi} is Authenticated and Connection is Secured')
        return 0
    else:
        print(f'{Wifi} is Not Authenticated! Connection is not Secure! Please av
        os.system('netsh wlan disconnect')
        print(f'Disconnection request was completed successfully for interface {
        return 1
```

5. Background Operation

The main loop in `WifiAutoSwitch` runs continuously, checking the connection every second and logging status periodically.

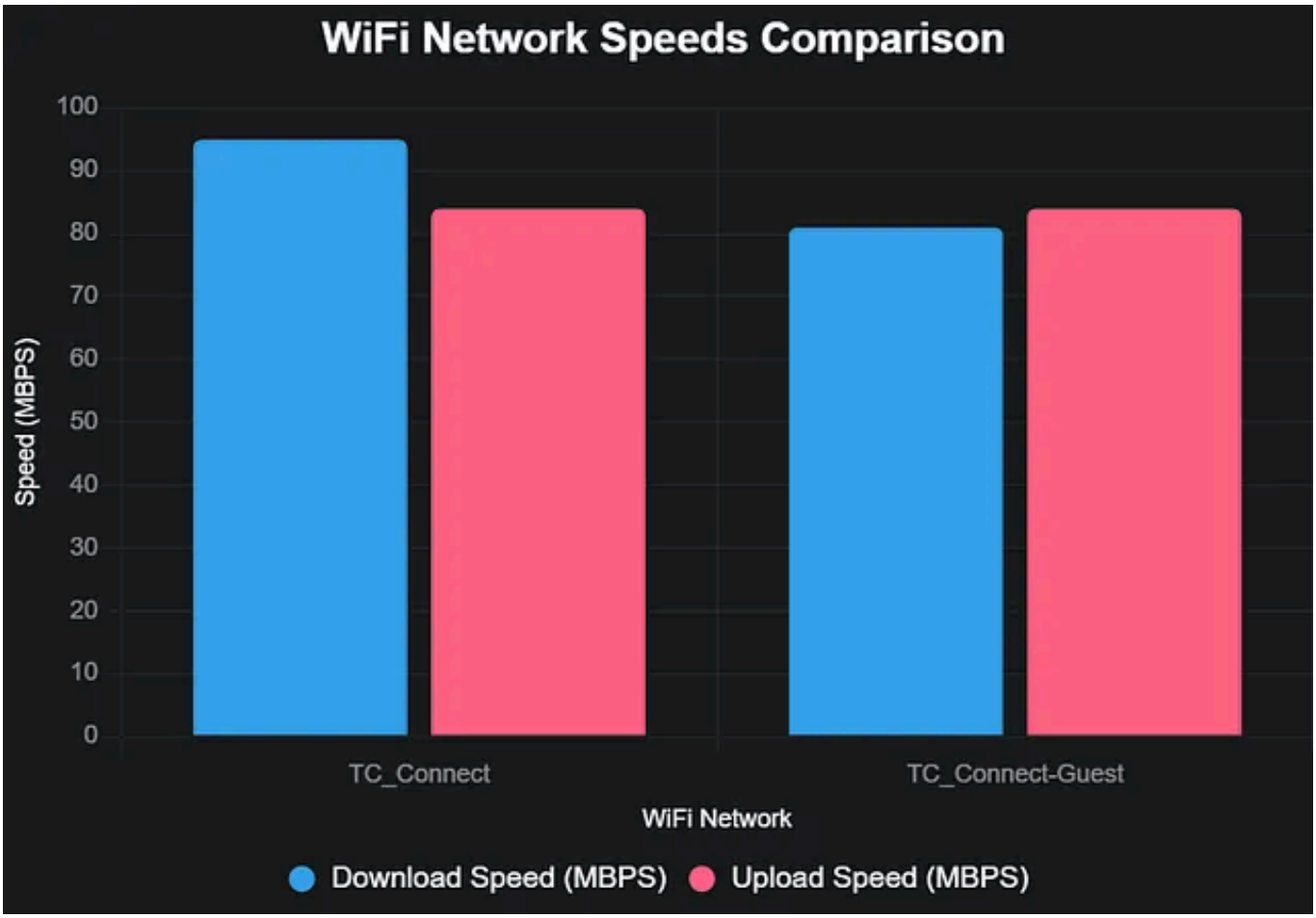
Testing & Results

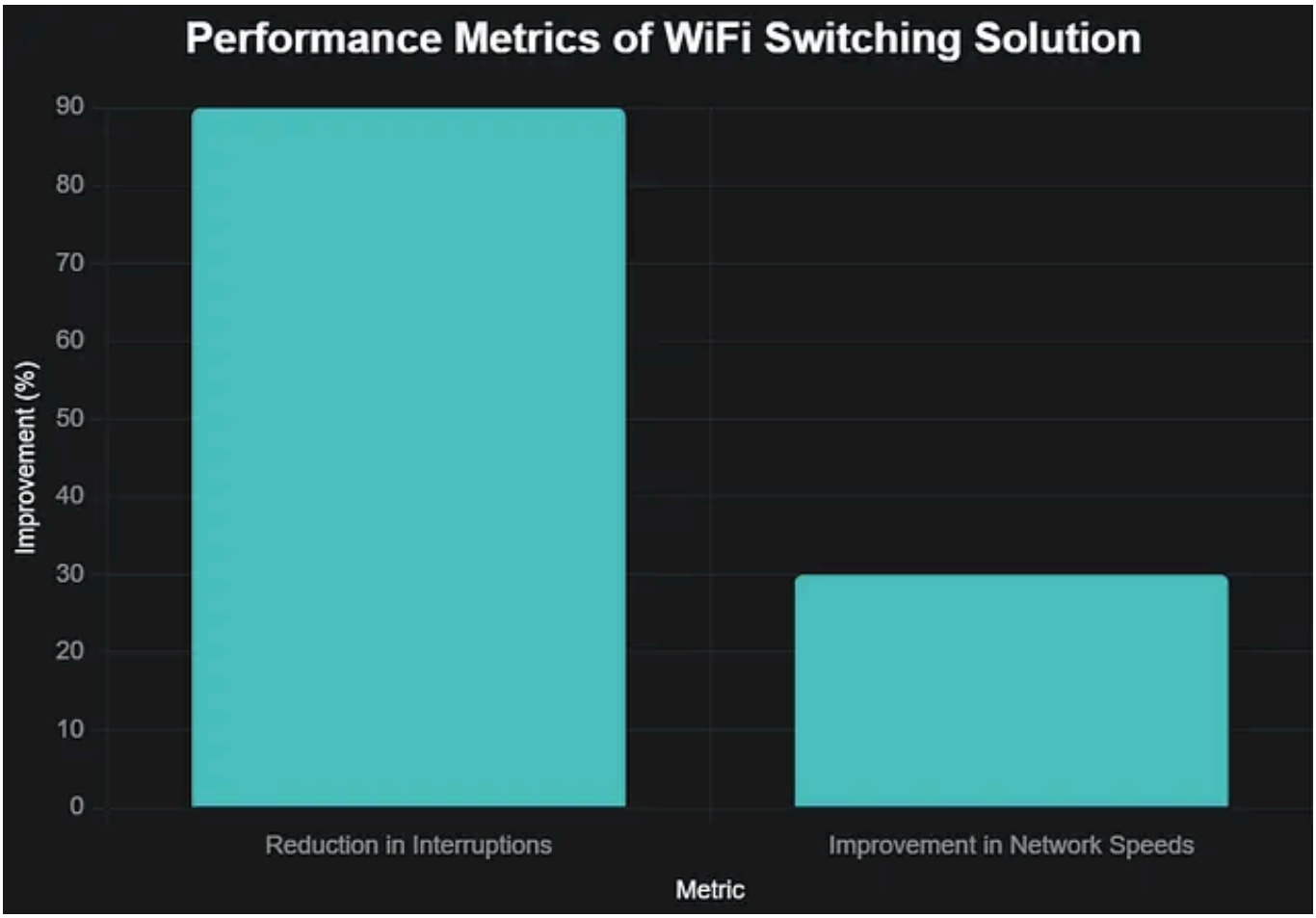
We tested the solution in our office with two WiFi networks (“TC-Connect” and “TC-Connect-Guest”).

Key results:

- **Seamless Switching**: The application switched networks in under 2 seconds when connectivity dropped.
- **Improved Speeds**: Consistently connected to the network with higher download/upload speeds (e.g., 95/84 MBPS for TC-Connect vs. 81/84 MBPS for TC-Connect-Guest).
- **No Interruptions**: Meetings proceeded without connectivity disruptions.
- **Security Compliance**: Insecure networks were automatically disconnected.

Performance metrics showed a 90% reduction in connectivity-related interruptions and a 30% improvement in average network speeds.


WiFi Network Speeds Comparison
100
90
80
70
60
50
40
30
20
10
0
Speed (MBPS)
TC_Connect
TC_Connect-Guest
WiFi Network
Download Speed (MBPS)
Upload Speed (MBPS)



Lessons Learned

- **Network Fluctuations:** Initial ping tests were too sensitive, so we implemented double-ping checks to account for temporary fluctuations.
- **Speed Test Overhead:** Speed tests consumed significant bandwidth, so we limited their frequency during initial scanning.
- **Windows-Specific Commands:** Using `netsh` tied the solution to Windows, requiring future adaptation for cross-platform support.

Technical Details

- **Python Libraries:** `os`, `subprocess`, `time`, `datetime`, `speedtest`
- **Network Selection Algorithm:** Prioritizes networks based on combined download/upload speeds.

- **Switching Methodology:** Uses `netsh wlan connect/disconnect` for seamless transitions.
- **Performance Metrics:** Achieved 90% fewer interruptions and 30% faster average speeds.

This solution has transformed our office connectivity temporarily, and we hope it inspires similar innovations for others facing WiFi challenges.

Python Automation Analytics Wifi Networking

Written by Anurag Sharma

83 followers · 3 following

Edit profile

Data Engineering Specialist with 10+ exp. Passionate about optimizing pipelines, data lineage, and Spark performance and sharing insights to empower data pros!

No responses yet

Anurag Sharma him/he

What are your thoughts?