Autotree: Connecting Cheap IoT Nodes with an Auto-Configuring WiFi Tree Network

Martin Gergeleit
Hochschule RheinMain -
University of Applied Science
Wiesbaden, Germany
martin.gergeleit@hs-rm.de

***Abstract*—This paper describes a tree routing mechanism called "Autotree" that automatically establishes an IP-based multi-hop routing on a network of WiFi enabled IoT nodes. It uses NAT and a variation of distance vector routing to dynamically establish an efficient tree without any manual intervention. It automatically adapts to topology changes, thus it can react on node failures and (limited) node mobility. It uses standard IEEE 802.11 links with WPA2 security. Also, the mechanism is completely transparent for other connected WiFi devices. Autotree has been implemented and evaluated on ESP8266 boards.**

Keywords—IoT, WiFi, routing, mesh, auto-configuration, ESP8266

I. INTRODUCTION

One important aspect of the IoT is the underlying communication infrastructure. Clearly, in most IoT applications this infrastructure has to fulfill the following five slightly contradicting requirements: it has to be wireless, it should be able cover larger (outdoor) areas, it has to provide Internet connectivity, it should be as energy efficient as possible, and it should be cheap enough to be deployed in the mass market in a huge amount of devices. All available technologies have some pros and especially cons in these five dimensions: 4G mobile networks are neither energy efficient nor cheap. 5G is not yet available or still quite expensive. LoRa works only one-way, has very low bandwidth, and doesn't speak IP protocols. Neither does ZigBee with its IEEE802.15.4-based layer 2. While it can set up mesh networks over a larger area, it still needs a gateway and a protocol translator to talk to Internet services. Other IEEE802.15.4-based solutions like 6LoWPAN can do better here. At least in theory, but due to the very limited bandwidth (and the still missing breakthrough of IPv6 in the consumer market) 6LoWPAN isn't widely adopted now. Finally IEEE802.11 WiFi: one might discuss, whether WiFi with a power consumption of about 100mA during operation is a perfect choice for IoT applications, but it surly is the most widespread available (semi-private) network technology. Also – as long as power saving is not the main concern – its bandwidth and its full support of all IP-based protocols is a major advantage as it allows for a direct interaction with all IoT cloud services. A significant drawback of WiFi however is its limited range. A typical WiFi cell has a diameter of less than 30 m, therefore a number of attempts have been started to enable routing on WiFi nodes in order to extend the range, including IEEE 802.11s for mesh networking[4].

This paper describes the design and a working implementation of a tree routing mechanism called *Autotree* that establishes an IP-based routing on a network of WiFi nodes. It uses NAT and a variation of *Distance Vector* (DV) routing to dynamically establish an efficient tree without any user intervention. It automatically adapts to topology changes, thus it can react on node failures and (limited) node mobility. It uses standard IEEE 802.11 links with WPA2 security, thus it provides the same level of security as normal WiFi cells. Also, the mechanism is completely transparent. I.e. other WiFi nodes that are connected to the routing tree spanned by the devices and do not run the Autotree mechanism can still use the network and its IP connectivity without even knowing about the existence of the internal routing. This protocol has been successfully implemented and tested on a network of very cheap ESP8266[3] MCUs, that are nowadays built into many available commercial IoT devices with WiFi connectivity.

The remainder of the paper is organized as follows: first related work on (mesh) routing in WSNs and especially WiFi and ESP8266-based networks is given in section II. Then section III describes the ESP8266 and the extension of the lwIP stack required for tree routing. Based on this, Section IV then gives an overview on the Autotree protocol, the dynamic configuration of the routing tree, as well as security and power save issues. Finally, section V shows performance results of a working Autotree network.

II. RELATED WORK

Routing is one of the basic problems in any multi-hop network. In general the well-known Internet routing protocols could be used in a network of interconnected WiFi nodes as well. However these protocols were designed for fairly static networks with stable nodes and very few topology changes. Their timing constants are in the order of minutes as they are optimized for WAN traffic and their reaction times are slow. This led to the development of faster and more dynamic protocols for wireless networks.

Ad hoc On-Demand Distance Vector (AODV) is a layer-2 routing protocol for mobile ad hoc networks [6]. It has been proposed already in 2003 and in the meantime it has been adopted e.g. for the ZigBee standard. It combines mechanisms known from DV protocols with additional active, on-demand route discovery, broken link detection and route caching. Its advantage is that it can establish end-to-end links for arbitrary link-layer mesh topologies and that it can cope with (moderate) node mobility. However, AODV requires a fairly complicated state-machine with a lot of timers and it is obviously quite heavy-weight when neither node mobility nor arbitrary traffic patterns and topologies are required.

Since 2011 IEEE 802.11s describes a standard for WiFi mesh networking. While initially slowly adopted it has become more popular in recent times as some of the major network equipment vendors are now offering mesh-enabled APs. These APs are typically 2.4/5GHz dual-band solutions with one band used as backbone and the other one as access

network. The topology of an IEEE 802.11s mesh network is managed by a special layer-2 routing protocol, the *Hybrid Wireless Mesh Protocol* (HWMP.) HWMP uses proactive tree routing for static nodes and an ad-hoc routing approach inspired by the AODV protocol for mobile nodes. While interoperability of devices from different vendors is still an issue, there is nothing wrong with IEEE 802.11s: it is well-suited for establishing robust, high-bandwidth WiFi networks for laptops, tablets, smartphones and AV-devices spanning a whole building or campus without a cabled infrastructure. These mesh APs are even perfect for providing connectivity to WiFi-based IoT devices within their existing coverage. However, they are probably by far too expensive and also too power consuming for being installed just to connect a small number of low-bandwidth IoT-sensors in a wide area of perhaps several 100m diameter. Here a more cost-effective solution would be preferable. Why not using the same cheap sensor boards for routing or – even cheaper – enabling the WiFi application nodes themselves to act as routers?

There is another mesh implementation for the ESP8266: "painlessMesh" (gitlab.com/painlessMesh/.) Similar to the approach presented in this paper it is based on a layer-2 tree, established via the simultaneous WiFi links of the STA and the AP interfaces. In contrast to the solution proposed here it implements routing solely at user-level, i.e. via dedicated neighbor-to-neighbor TCP links and typed JSON messages as payload. This has the advantage that arbitrary traffic patterns can be supported, but this also implies that no IP traffic and thus none of the standard IoT protocols like MQTT or CoAP can be forwarded.

III. Routing on the ESP8266

The Espressif ESP8266[3] has become a quite popular MCU for IoT applications as it combines the power of a Tensilica L106 32-bit core and 96 kB data RAM with a number of configurable GPIOs and especially an integrated WiFi IEEE 802.11bgn interface – and this all at costs of less than 2$ per module. Several different development environments are available: easy-to-use Arduino-based IDE, the original Espressif NONOS SDK, FreeRTOS, NodeMCU and many others. Also all sorts of IoT-enabling libraries for sensor/actor access and cloud integration are provided by the open source community, like e.g. for MQTT, HTTP/S, CoAP, TLS etc. The ESP8266 doesn't support IEEE802.11s, but it can work as IEEE 802.11 STA and AP - even at the same time. In principle, this already enables it to act as an IP router with two connected networks. This is enough for any ESP8266-based board to work as a node is a tree network with one uplink (at the STA interface) and several downlinks (at the AP interface.) While this is not a full mesh structure, it would provide a perfect topology for a data collecting wireless sensor network (WSN) with one central root, e.g. a central gateway with IP connectivity. Also an MQTT based network with an MQTT broker reachable via the root of the tree would work quite well with distributed sensors and actors.

While the IEEE802.11 WiFi driver is closed source, the IP laver is completely open source. It is based on the Adam Dunkel's lwIP TCP/IP stack[2] that has been adapted to run on the ESP8266. In addition, it has been extended by a *Dynamic Host Configuration Protocol* (DHCP) client running on the STA interface and a DHCP server on the AP interface. As buffer space is limited on the ESP8266, the number of concurrent TCP connections is restricted to 5 by default (can be increased to 8) and the window per connection is small. By default, routing is not enabled on the ESP8266. But as the lwIP stack implementation already supports IP routing, it requires only a change of a single option and a recompilation of the library to enable it. Now routing between the two IP subnets on the STA and the AP interface works, but only if the nodes on both sides use the appropriate routing entries. Static routing beyond two hops still doesn't work due to the lack of a routing table in the lwIP stack of the ESP. During the development of this project static routing tables have been added to the library. Thus, a manually configured larger network of several ESP8266s with different IP subnets on each interface would work. However, this is clearly not a desirable state in a dynamic wireless IoT network. It would require at least an IP routing protocol to dynamically populate and adjust the routing tables to avoid too much manual maintenance effort. But this would still require an individual network configuration of each node, as each node must have at least a distinguishable IP network prefix (at least at the AP interface) before it can run any IP traffic.

That is why in this paper another approach has been chosen: *Network Address Translation* (NAT)[8] with auto-generated network addresses. Together with the DHCP neither any node on the downlink network nor any uplink station needs to have any routing entry. Forwarding via a NAT router is completely transparent on both sides. As the router is free to choose any (not even unique) network prefix on the downlink side, completely dynamic and automatic configuration becomes possible.

NAT is well known for converting private IPv4 addresses into public ones and for hiding complete huge networks behind one or a few public IPv4 addresses. Even several levels of NAT in a row are known to work well – it is used every day in mobile networks. In the case of the ESP8266, NAT can be used to hide the complete downlink network of the AP interface behind the address of the STA interface. The ESP8266 becomes the default router for all nodes in the downlink network and forwards any external request with its own STA IP-address to the uplink network. As usual for NAT, different connections from different IPs are translated and encoded into different port numbers and temporarily stored in a NAT table. Inbound traffic is correctly translated back to the originating IP addresses as long as the "connection" has been established from the downlink network. Unsolicited traffic from the uplink network is blocked as long as there is no explicit port forwarding defined in the NAT table. For this project the lwIP stack has been enhanced by a NAT table. The table stores outgoing TCP, UDP, and ICMP "connections" and aids the correct translation of outgoing and incoming packets via the STA interface. During translation the IP header checksums are updated as described in RFC3022. As mentioned above, memory is an issue on a small MCU like the ESP8266. Thus, the size of the NAT table is a limiting factor. Currently, the size is limited to 512 entries by default. In addition the timeout constants for dropping unused table entries have been reduces compared to the proposed values from the RFC (currently 30 min for TCP connections and 2 secs for UDP and ICMP conversations.) TCP connections that are terminated correctly via the FIN flag delete their table entry immediately with the last packet.

With this routing and NAT extension in place, the ESP acts as transparent NAT router that forwards any traffic from the downlink network to the uplink and possibly to the Internet. With this features it can be used a simple and cheap (battery powered) “WiFi range extender” that has been used by many users and works with a bandwidth up to about 5 Mbps.

IV. AUTOTREE

To cover a larger distances or wider areas it might become necessary to use more than one of these “range extenders” in a row. Generally, this can be done without any problems with NAT routers; actually, this will result in several layers of NAT. However, this means connectivity is limited: all nodes can talk uplink to the Internet, but generally there's no direct IP connectivity between the nodes on different routers. Also it is known, that scalability of WiFi mesh networks is limited[1] and the available bandwidth goes down the more hops it takes. But users have reported that even five ESPs in a row work quite well[7]. For typical IoT applications both issues are not a severe limitations. IoT devices typically don’t communicate directly with each other but set up an (encrypted) link to an cloud server. Also bandwidth is not an issue: an IoT device uses typically a few KBps but very rarely some MBps. In such a setup, manual configuration of repeaters is quite a time consuming and error-prone activity.

A. Protocol Design

To simplify the setup "Autotree" has been developed. It is a layer 2/3 mechanism for setting up a tree-like layer 2 network topology and configuring an according layer 3 subnet structure and the appropriate routing. Autotree first needs at least one WiFi AP as root node, typically with Internet access. The root node itself is not aware of Autotree. Autotree is just a special way to automatically configure the interfaces and the network settings of additional relaying WiFi nodes. For this paper, Autotree has been implemented on ESP8266s, but the concept works on any WiFi device that is able to run a STA and an AP interface at the same time and to do NAT routing in between them. In the following, such a single device will be called an Autotree node (“ATnode”). An ATnode acts as a network router that provides WiFi access to other devices, but it may also be an IoT device itself, i.e. perform as sensor/actor in an application.

As network configuration an ATnode solely needs the desired SSID and the WPA2 password of the common WiFi network. Afterwards the ATnodes can be distributed in the area that has to be covered. After power on each ATnode will walk through the following steps:

1. Turn the AP interface off.
2. Search for the best other AP with the same SSID in reach. This is the one which is closest to the root AP (or the root itself) and has the best signal strength (RSSI.)
3. Connect with the STA interface to that AP.
4. Start a WiFi network on the AP with the same SSID/password as the one it is connected to. Clients can use the same WiFi settings for any network, the original network or the repeated ones.
5. If uplink is lost, go back to step 1.

In order to find the best other AP in step 2, the signal strength is easy to measure with a scan. But which is the one closest to the original WiFi network, when it finds several APs with the same SSID in a scan? Therefore the protocol use a somewhat dirty trick on the ESP8266: an ATnode mode manipulates its BSSID, i.e. the MAC address of its AP interface (actually, according to the IEEE 802.11 standard this is the "ESSID" as it is an AP, but the ESP8266 SDK calls it "BSSID".) The BSSID is sent out with every beacon frame, about ten times per second. It uses the format: 24:24:mm:rr:rr:rr. "24:24" is just the unique identifier of a ESP (there is a minimal probability that this collides with the real APs MAC, but one can neglect this, as one can change that prefix if really required.) "mm" means the "mesh level", this is the distance in hops to the original WiFi network. The last three "rr:rr:rr" are just random numbers to distinguish the various ESPs. The original AP keeps its BSSID, i.e. the one without the prefix "24:24" is recognized as root, called mesh level 0. Manipulating the BSSID is not the most elegant way of implementing additional information in a beacon frame, but it works. Conceptually it would be much better to use a user-defined tag with the same information in the beacon frame, which is allowed by the IEEE802.11 standard, but the closed source driver prevents beacon manipulations on the ESP8266.

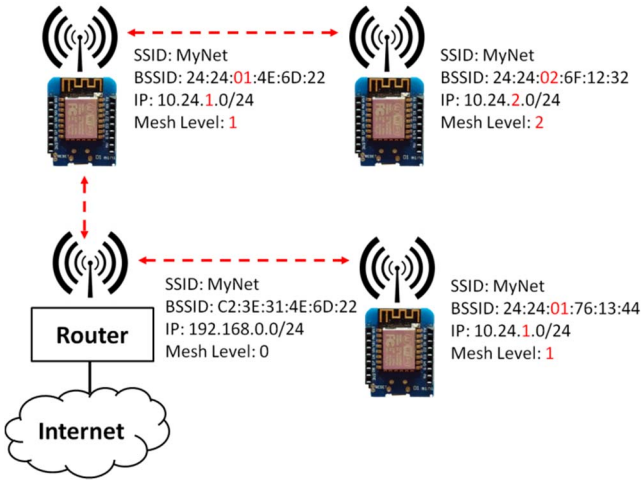

Fig. 1. Example of a simple Autotree Configuration.

With this information each ATnode can learn which other ATnode is the closest to the original WiFi network, can connect to that, and choose its own BSSID accordingly. Also the IP address of the STA interface is adjusted to the mesh level: 10.24.m.0. This creates a tree with the original WiFi AP as root and repeating nodes on several mesh levels Actually, it works somewhat similar as the *Spanning Tree Protocol* (STP) on the link layer or routing on the network layer using a DV protocol. Seen as a DV protocol the metric it uses is the distance to the root. A common problem in DV protocols are (temporary) routing loops that occur, when a link goes down and an alternative route is established via a node that is also disconnected by this link loss, but doesn’t yet know about this. These routing loops are completely avoided in Autotree, because the routing tree is always constructed in the direction from the root downwards. As soon as an uplink link loss is detected, an ATnode restarts configuration in step 1, immediately disables the AP interface, disconnects its STAs, and stops beacon sending. This in turn means that all downstream ATnodes in the tree

connected via this ATnode also recursively disconnect and start re-configuration.

Figure 1 shows an example of a simple Autotree configuration with one central router and three ATnodes. All nodes in this network have the same SSID and use the same shared password. The central router with Internet access (mesh level 0, root of the tree) can be detected by all nodes as it has a BSSID not starting with the magic prefix “24:24”. The other ATnode encode their mesh level in their BSSID, so that other can detect the best reachable node, which is the closest to the root. In general, there can be more than one root with Internet access. The mechanism also works when there are other mesh level 0 routers. In that case it will create more than one tree. Each ATnode simply selects the best tree, the one accessible at the lowest mesh level.

B. Security

As Autotree is based on normal WiFi AP-to-STA links, it uses at link level the standard WPA2-PSK security with a shared password. This means only authenticated nodes can join the network and each link is individually encrypted by a temporal key. Link level sniffing is as difficult as it would be in a single cell WiFi network. Of course, distributing a password in weakly protected IoT devices is a security issue, but this is true for any WiFi network (and many others), no special problem of Autotree, and beyond the scope of this paper. However the ESP8266 is powerful enough to use public key encryption for authenticating individual devices with TLS at the application layer.

C. Protocol Parameters

If there are more than one ATnode in range, there might be a trade-off in the end-to-end performance between a shorter "bad" path and a longer "good" path (good and bad in terms of link quality.) The parameter *am_threshold* determines what a bad connection is: if the RSSI in dB in a scan is less than this threshold, a connection is bad and a path with one more hop is preferred. I.e. given *am_threshold* is -75 dB and there are two Autotree nodes detected in the scan: ***A*** with level 1 and RSSI -78 dB and ***B*** with level 2 and RSSI -60 dB, then a link to ***A*** is considered as too bad (-78 dB < am_threshold) and ***B*** is preferred. The new node will become a level 3 node with uplink via ***B***.

D. Management and Power Save

For setup and re-configuration Autotree doesn’t need any additional management layer, just the SSID and the WPA2 password is enough. However, in a larger network additional aspects of network management might become important: state of nodes and links, bandwidth bottle necks or a central overview of the current network topology. As AMnodes are not directly accessible due to the use of NAT, a centralized management interface has been implemented via the MQTT protocol at layer 7. All ATnodes can be configured to connect to a globally accessible MQTT broker (via the root node.) With publications and subscriptions on dedicated topics a network management console can monitor and control either a single node or all participating router in the network at once. To get more insight into the topology of an Automesh network, all nodes periodically publish on a "TopologyInfo" topic. The network management console subscribes on this topic and will get all the node and link information including the RSSI (of connected ATnodes) required to reconstruct the complete graph and to detect weak links in the mesh. The “TopologyInfo” topic contains the following JSON structure like the one depicted in figure 2.

```
{
"nodeinfo" {
      "id":"ESP_07e37e",
      "ap_mac":"24:24:01:72:c7:f9",
      "sta_mac":"60:01:bc:07:e3:7e",
      "uplink_bssid":"00:1a:54:93:23:0a",
      "ap_ip":"10.24.1.1",
      "sta_ip":"192.168.178.33",
      "rssi":"-66",
      "mesh_level":"1",
      "no_stas":"2"
},
"stas":[
      {"mac":"5c:cf:45:11:7f:13","ip":"10.24.1.2"},
      {"mac":"00:14:22:76:99:c5","ip":"10.24.1.3"}
]
}
```

Fig. 2. Example of status info of an ATnode.

This allows also for a global power save, which can mitigate a problem of all relaying/routing IoT nodes: the power supply. Power save is difficult as these nodes have to be active at least as long as any of the application nodes sends traffic across them. Ideally, they are awake all the time. In some application scenarios a time slotted approach with global sleep times of the whole network might relax the problem. The MQTT management console can send all ATnodes in a synchronized deep sleep for a certain amount of time (on the ESP8266 up to 71 minutes.) When deep sleep ends, the nodes wake up and after a few seconds automatically re-establish the routing tree.

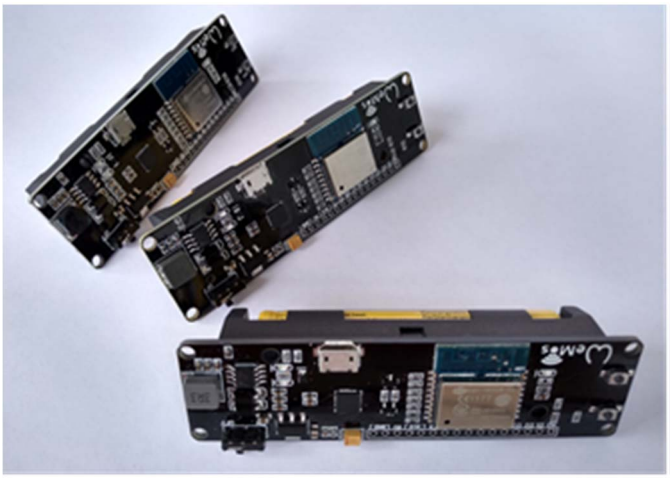

Fig. 3. Battery-powered ESP8266 prototype boards.

V. Evaluation

A test scenario has been set up with the battery powered nodes depicted in Figure 3. A root AP and three ATnodes were placed in a row with a distance of about 7 meters and an average RSSI of about -72 dB, forming an Autotree “tree” with levels 0 to 3. The whole network used IEEE 802.11g, the maximum an ESP can reach in STA and AP mode. Performance was tested with two additional nodes: a Raspberry Pi connected to the root AP via Ethernet as responder and an Android smartphone as mobile test device.

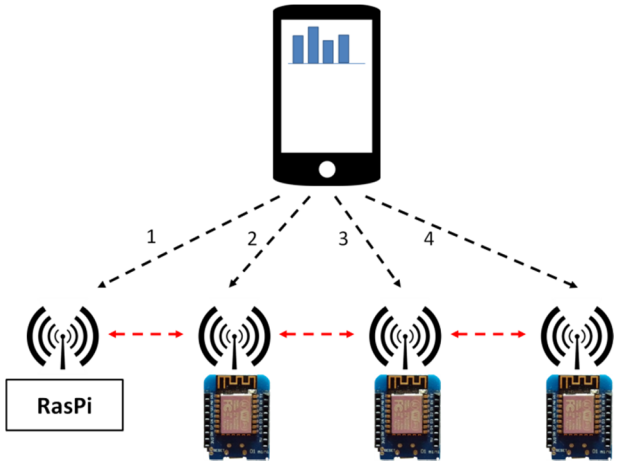

Fig. 4. Test setup with ATnodes up to mesh level 3.

As depicted in figure 4 the smartphone was first connected to the root AP (1) and then successively to the EPSs at the different mesh levels (2-4) resulting in 1 to 4 wireless hops per packet. Performance was then tested by "ping"ing the Raspberry Pi from the smartphone for latency evaluation and using "iperf" with default parameters for TCP bandwidth measurement. The results are shown in figure 5: basically, each additional mesh level halves the available bandwidth and adds about 2ms of round trip latency.

Hops	RSSI (dB)	Bandwith (Mbps)	Latency (ms)
1	-69	12	4,1
2	-74	5,5	5,6
3	-71	2,4	7,6
4	-75	1,6	11,2

Fig. 5. Battery-powered ESP8266 prototype boards.

Independent experiments[7] of other users have also shown, that even a setup over 5 hops and a distance several streets works well and provides sufficient bandwidth for IoT applications.

VI. Conclusion

Autotree is a simple and robust distributed algorithm that constructs a tree network of WiFi nodes using IP routing an NAT. It is well suited for IoT applications as it provides direct IP connectivity to the Internet and sufficient network performance. It can be implemented on the IoT nodes themselves and can cover areas with a radius up to 5 times larger than a single WiFi cell. A fully functional implementation of the Autotree protocol for ESP8266 can be downloaded and tested at https://github.com/martin-ger/esp_wifi_repeater. In the near future efforts are planned to extend the concept into a generally available component for IoT applications. An obvious next step will be the implementation of the protocol for other IoT-platforms like the ESP32 MCU. In order to allow for a full end-to-end IP connectivity of ATnodes also a new project has been started to tunnel IP traffic in MQTT and use a central MQTT broker as switching component.

References

[1] Chaudet, C., Dhoutaut, D. and Guérin Lassous, I., "Performance issues with IEEE 802.11 in ad hoc networking", IEEE Communication Magazine, volume 43, number 7, July 2005

[2] Dunkels, Adam, "Design and implementation of the lwIP TCP/IP stack", Swedish Institute of Computer Science. 2, 2001.

[3] Espressif Systems, "ESP8266EX Datasheet", Version 5.8, 2018, https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[4] Hiertz, G. R. et al., "IEEE 802.11s: the WLAN mesh standard", IEEE Wireless Communications, Vol. 17, No. 1, pp. 104-111, 2010.

[5] Ivan Grokhotkov, "ESP8266 Arduino Core Documentation, Release 2.4.0", May 14, 2017.

[6] Perkins, C., Belding-Royer, E., Das, S., "Ad Hoc On-Demand Distance Vector (AODV) Routing", Jul. 2003.

[7] Spranger, M., "ESP8266 WLAN-Mesh über 4 Nodes", Jul 2018, https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/esp-als-w-lan-router

[8] Srisuresh, P., and Egevang, K., "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, RFC Editor, 2001.