T.E.N.T.
Temporary Easy Network Technology

Orion Collins
Department of Computer Science and Networking
Wentworth Institute of Technology
Boston, MA 02115, USA
{collinso}@wit.edu

Abstract- In areas with limited to low infrastructure there needs to be a way to setup an area network allowing for various uses including unified communications and a decentralized webserver. The Temporary Easy Network Technology or T.E.N.T that was created allows for a lightweight Wireless mesh network to be setup. By utilizing raspberry pi's and batman advanced I created a low cost and easily scalable wireless network that can be configured to fit the need of each situation.

Keywords- Limited Infrastructure, Lightweight, Raspberry Pi, B.A.T.M.A.N, Decentralized, Portable Wireless Mesh Network

I. INTRODUCTION

In disaster relief scenarios, communication with others is key. Being able to get up to date news on the disaster and what efforts are being taken to recover would be very helpful to have, and in some cases, save lives. Doing all of this will be possible if a small, lightweight wireless mesh network (WMN) could be deployed during disaster relief efforts. These instant area networks (IAN) would be temporarily set up to supplement damaged infrastructure.[2]

The answer to the problem is TENT. Temporary easy network technology. A TENT can be placed in an area that has recently been affected by a catastrophic natural disaster. It will be implemented, using raspberry Pis acting as the nodes of the wireless mesh network. There wouldn't need to be a connection to the internet, as this network will be using ad-hoc networking. An ad-hoc network is a network that does not have a main connection, such as a router. It has multiple end-devices that can communicate to one another using a variety of different protocols.

The TENT will be configured as a mesh network, a network topology where there is not a central node to direct and administer all content. Instead, each device directs traffic for each other device on the network. When a new device is discovered its nearest neighbor updates it with the correct routing information. Then relays that information to surrounding nodes who continue to propagate across the network a new device has joined. Mesh networks are much more stable than a traditional centralized network typically known as star topologies. Mesh networks are categorized as proactive, reactive and hybrid. A proactive protocol uses tables that tracks the routes for destinations that are updated at each topology change. Reactive protocols state that each node only keeps track of its neighbors when there is the need for it to communicate, a bigger delay is only generated if a new path is necessary. Hybrid protocols use the features of both proactive and reactive protocols, such that for each set of nodes, only some of them do a periodic update of the possible traffic destinations.[1] Our wireless mesh network will be using a proactive protocol B.A.T.M.A.N. Thanks to the ease of expansion in WMNs, a server administrator can easily deploy more nodes onto the network at any given time increasing the range of coverage.

Raspberry Pis are used for the nodes because of the low cost and, the operating system, Raspbian. Given an intermediate level of Linux experience a user should be able to get the network up and running. The version of Raspbian that was used was Raspbian Stretch and Jessie displaying cross compatibility between different operating systems. Raspberry Pis are a very light piece of hardware, but it can run an operating system with a full GUI alongside other software you put on the device. On the Raspberry Pis, I have taken advantage of using the onboard wireless cards. These wireless cards have been switched onto ad-hoc mode and can be able to maintain connection up to ten meters away.

The protocol to be introduced is B.A.T.M.A.N, or, Better Approach to Mobile Ad-Hoc Networking. Most of the time, wireless routing protocols operate on the network layer 3. This way, they exchange routing information by transferring UDP packets to one another. Batman-adv is mesh networking protocol that utilizes layer 2 communications encapsulating and forwarding all traffic of participating nodes.[4] Therefore, all nodes appear to be on the same LAN and are unaffected by topology changes. Batman-adv operates in the kernel limiting the amount of CPU power required to manage the connections. B.A.T.M.A.N. is a proactive protocol that identifies only the best next hop instead of discovering the complete route. Therefore, there is no need for the global knowledge of all the changes in the network topology. Batman determines for each destination in the mesh one single hop

neighbor, which can be utilized as the best gateway to communicate with the destination node.[4] A node detects the presence of Batman originators to/from a Batman originator which are used for link quality and path detection. Every batman node broadcast an originator message (OGM) which informs neighboring nodes about their existence. An OGM packet has a field for: its version, a field to inform if the node is a direct neighbor or not, a unidirectional flag, a desired value for the Time-To-Live (TTL), a gateway flag (to inform if it is a node with Internet access), a sequence number used for the packet identification and an originator address. In my case the <172.28.x.x/16> IPv4 address of the B.A.T.M.A.N. interface on which the OGM has been generated.[1],[7] To avoid control traffic overload, the overall number of messages that flood the mesh topology is limited. Considering the intended scenario, a communications networks for humanitarian aid in post disaster environments. B.A.T.M.A.N. seems as one of the best possibilities for a routing protocol with its performance improved given the use of a limited number of mesh nodes for temporary coverage of an area. Internet connection is not an issue for the sake of the depicted scenario if all they needed was to communicate supplies, or transfer files between devices. Besides, a high node density limits the networks ability to cope with a large number of hops in the transmission path. Therefore, the relatively short number of hops for this kind of deployment favors the use of B.A.T.M.A.N.[1]

Nginx, a software created in 2004 was utilized for the web server for distributing news and content globally. We used the Nginx web server to be able to communicate with others in the network. We wanted a very basic web server for posting news and broadcast communications. Then Rocket.chat was employed for coordination purposes this helps people communicate and collaborate. Rocket chat comes equipped with a simple GUI, so it was easy to use and setup, with a built-in account system to manage users on the chat server.

II. PROBLEM

The problem I aimed to solve was the need for an easy to deploy, fast acting and low-cost network in areas with low/limited infrastructure. Our project must be flexible and easily expanded while also being decentralized and pliable in-case of other issues.

-low cost
-light weight
-scalable
-easy to implement
-provide platform for typical network services

III. SOLUTION

Step 1: flash the sd cards with Raspbian lite enable ssh on boot, login and change default passwords and hostnames. Enabling SSH will allow for headless configuration but is not necessary to getting the network up and running. Changing the password will improve the overall security of the devices. Altering the hostnames will also improve clarity for optional services like DNS.[9]

```
$sudo systemctl enable ssh
$sudo passwd pi
$sudo raspi-config
```

Fig. 1 basic configuration options(*raspi-config* is a cli graphical program for managing various features on the pi)

Step 2: Update Pi and install required packages for batctl, nginx, and snap for Rocket.chat. Snap is an additional package manager recommended for installing large packages like Rocket.chat-server. Then install Rocket.chat. Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.[11] This step assumes that the users are connecting at least one node to the internet to download the packages. If internet connection is not up, precompiled .iso images can be distributed containing required software. [5],[6]

```
$sudo apt-get update
$sudo apt -y install libnl-3-dev libnl-genl-3 dev nginx git snapd
$sudo snap install rocketchat-server
```

Fig. 2 update and install (update is required before installing any new packages onto the pi using the *apt* package manager.)

Step 3: Download and install batctl in home directory (/home/pi/). *batctl* is the command line interface (CLI) tool used to manage the batman-adv kernel module. It offers a variety of features that will allow users to configure batman-adv, display debug information such as originator tables, translation table and the debug log. In combination with a bat-hosts file *batctl* allows the use of host names instead of MAC addresses. Since B.A.T.M.A.N operates on layer 2, *batctl* includes layer 2 equivalents of commands *ping*, *traceroute*, *tcpdump*, and *throughputmeter*.[4]

```
$git clone https://git.open-mesh.org/batctl.git
$cd batctl
$sudo make install
```

Fig. 3 *git clone <target URL>* (clones a directory from the internet into the current directory)

Step 4: Create a script to configure wlan adapter as meshpoint and configure and turn on bat0 inside the home directory. Which is batman-adv's virtual interface (bat0) were all batman packets will be directed. The commands on the script can be input manually into the terminal to achieve the same result. Line 1-2 of the script

will enable the B.A.T.M.A.N-adv kernel module which allows the device to transmit OGM's. Lines 3-8 performs a variety of commands. First is turns off the wlan0 interface which is the interface which is going to be configured to use B.A.T.M.A.N. Next the maximum transmission unit size(MTU) is set to 1532 which is the recommended size for B.A.T.M.A.N packets [4]. Then the project SSID is set to being "projMesh" using a randomly generate cell id(ap) on channel 11. After that the meshpoint is configured and the wlan0 interface can be turned back on, line 10. After that, the script bridges wlan0 with the bat0 interface utilizing *batctl* which is the virtual interface responsible for managing traffic. Finally, the script turns on the bat0 interface and assigns the ip address 172.28.0.1/16. After the script has been created the user needs to grant the script executable privileges so that once it has been added to the boot path it is able to execute properly. Note that if so desired a user manually run the script by moving to the home directory and running the script like so: $*sudo ./batstartup.sh* [10]

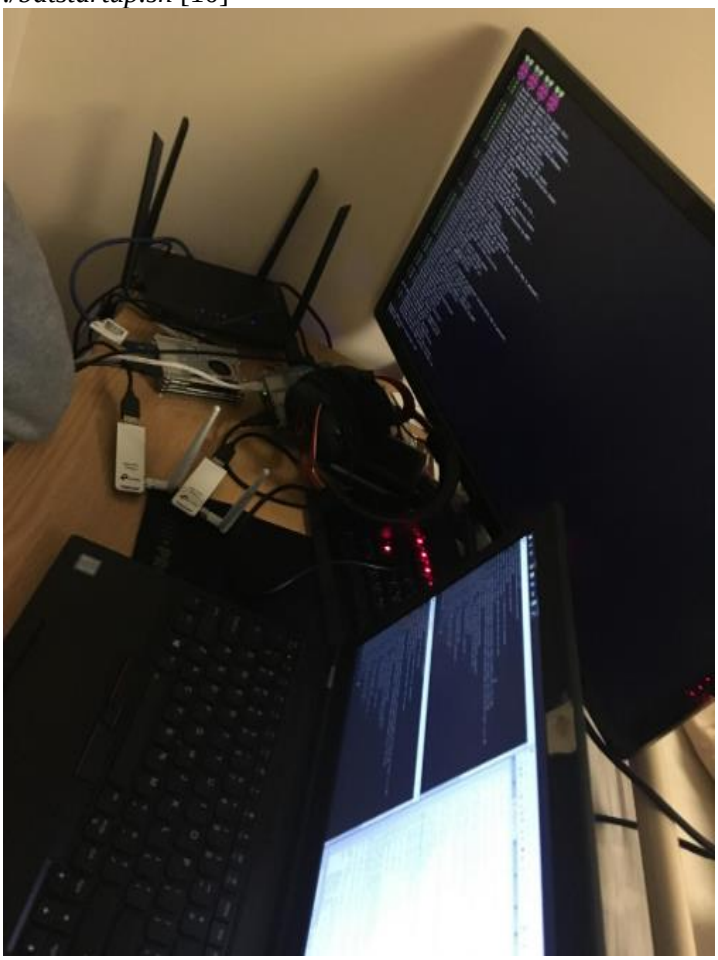

Fig. 4 raspberry Pi's running script. (Computer ssh to node 2 while node 1 comes up on the display.)

```
$sudo nano batstartup.sh
#Paste this inside
sudo modprobe batman-adv
sleep 5s
sudo ip link set wlan0 down
sudo ifconfig wlan0 mtu 1532
sudo iwconfig wlan0 mode ad-hoc
sudo iwconfig wlan0 essid projMesh
sudo iwconfig wlan0 ap any
sudo iwconfig wlan0 channel 11
sleep 2s
sudo ip link set wlan0 up
sleep 2s
sudo batctl if add wlan0
sleep 2s
sudo ifconfig bat0 up
sleep 5s
#netID: 172.28.x.x/16 | change x for each node
sudo ifconfig bat0 172.28.0.1/16
echo Done!!!
$sudo chmod +x batstartup.sh
```

Fig. 5 startup script for batstartup.sh (in *nano* use CTRL+O, Y to save the script. *chmod +x <target>* is a tool to make files executable from the command line)

Step 5: Make a quick edit to the dhcpcd config file so that everything in my script runs correctly. From experimenting it was discovered that the dhcpcd service was causing the startup script to fail. This was amended by adding the "denyinterfaces wlan0" to the end of the dhcpcd configuration file. Which default denied any dynamic ip address leasing to wlan0 so that it can properly act as a meshpoint.[9]

```
$sudo nano /etc/dhcpcd.conf
# something else is here
# .
denyinterfaces wlan0
```

Fig. 6 etc/dhcpcd.conf (don't remove any lines from the file CRTL+O, Y will save the file)

Step 6: Add startup scripts to boot path on pi's and reboot. This step is optional but recommended as otherwise the startup script will have to be run manually upon boot after each shutdown. The file rc.local located in the /etc directory executes scripts after normal bootup using *rc*. By adding the path to the batstartup.sh script in the home directory the mesh should start as a custom service.[8] If everything has be done correctly rebooting the pi should have it come online with a valid meshpoint operating B.A.T.M.A.N-adv.

```
$sudo nano /etc/rc.local
# something else is here
#.
/home/pi/batstartup.sh &
exit 0
$sudo reboot
```

Fig. 7 /etc/rc.local (don't remove any lines from the file and make sure to place the path to the startup script before exit 0 or the script will not execute. CRTL+O, Y will save the file and exit)

IV. TESTING

Verify device association

To check and see if both devices are on the same network first check *iwconfig* to see if both devices display the same cell id. If they display the same cell id it means the devices wireless adapters are linked via ad-hoc. Different cell ids would mean that the devices had been improperly configured and the startup script should be edited. A possible solution is replacing *sudo iwconfig wlan0 ap <Node Cell: ID>*. Next a user can check *ifconfig* to view the bat0 and wlan0 interfaces. Both should be up and the bat0 interface should be displaying an ip address assigned by the script in this case using the 172.28.x.x/16 scheme. Note again, that if duplicate ip addresses are assigned the nodes will not be able to communicate.

```
$sudo iwconfig
wlan0     IEEE 802.11  ESSID:"projMesh"
          Mode:Ad-Hoc  Frequency:2.462 GHz  Cell:
5E:CC:EC:C3:CC:39
          Tx-Power=31 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:on
```

Fig 8. *iwconfig* (outputs information about the status of current wireless interfaces)

```
$sudo ifconfig
bat0      Link encap:Ethernet  HWaddr ee:3e:bd:40:14:7b
          inet addr:172.28.0.1  Bcast:172.28.255.255
Mask:255.255.0.0
          inet6 addr: fe80::5ce8:5c3f:c44b:93e2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST
MTU:1500  Metric:1
wlan0     Link encap:Ethernet  HWaddr b8:27:eb:f8:4e:d9
          inet6 addr: fe80::ba27:ebff:fef8:4ed9/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1532  Metric:1
```

Fig. 9 *ifconfig* (outputs information about the statistics regarding all network interfaces. For example, displaying the MTU size for both wlan0 and bat0.)

Check neighbors and originators

Utilizing *batctl* a user can see the nearby OGM originators table. Each node maintains information about the known other originators (bat# interfaces) in the network in an Originator List. The Originator List contains one entry for each Originator from which or via which an OGM has been received within the last PURGE_TIMEOUT seconds. If OGMs from different Originators (bat# interfaces) of the same node are received, then there MUST be one entry for each Originator. In fact, the receiving node does not necessarily know that certain different Originators (and corresponding IP addresses) are belonging to the same B.A.T.M.A.N. node.[7] The command *batctl n* and *batctl tp <target>* will display both the next-hop neighbors and the throughput on the network. The neighbors table can be used to determine who is on the network and their associated mac addresses will be displayed along with when their OGM was last seen. Throughput shows a relatively small transmission rate at only 2.95MB/s but it is suitable for the limited services that need to be run on the network.

```
$sudo batctl o
[B.A.T.M.A.N. adv 2017.3, MainIF/MAC:
wlan0/b8:27:eb:b2:ea:0e (bat0/1a:8c:c4:7d:35:5a
BATMAN_IV)]
   Originator        last-seen (#/255) Nexthop           [outgoingIF]
 * b8:27:eb:f8:4e:d9    0.740s   (250) b8:27:eb:f8:4e:d9
[wlan0]
```

Fig. 10 *batctl* originators table (displays the nodes mac address and the mac address of the transmitting interface)

```
$sudo batctl n
[B.A.T.M.A.N. adv 2017.3, MainIF/MAC:
wlan0/b8:27:eb:b2:ea:0e
(bat0/1a:8c:c4:7d:35:5a BATMAN_IV)]
IF            Neighbor              last-seen
      wlan0     b8:27:eb:f8:4e:d9    0.050s
```

Fig. 11 *batctl* neighbors table

```
$sudo batctl tp b8:27:eb:f8:4e:d9
Test duration 10740ms.
Sent 33214428 Bytes.
Throughput: 2.95 MB/s (24.74 Mbps)
```

Fig. 12 *batctl throughputmeter* (can be used by the network administer to test speed on the network without installing additional tools)

Confirm communication services

Once connection has been established a user should be able to log into any of the services on the wireless mesh network. The website configured was minimal and served only to direct traffic to the chat service on the network where users could communicate needs.

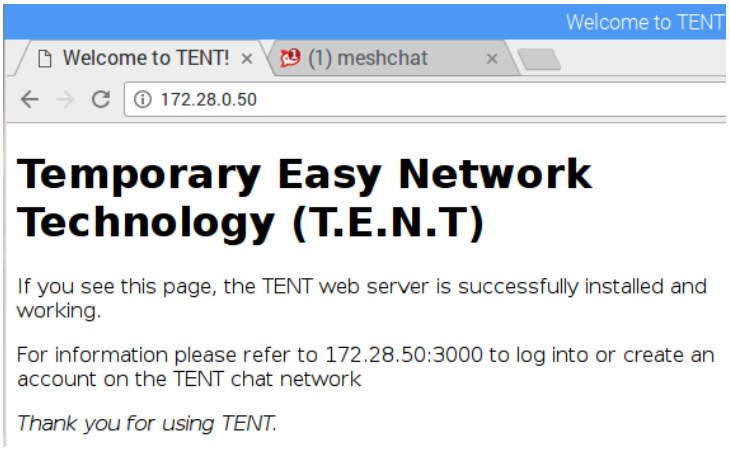

Fig. 13 TENT webpage

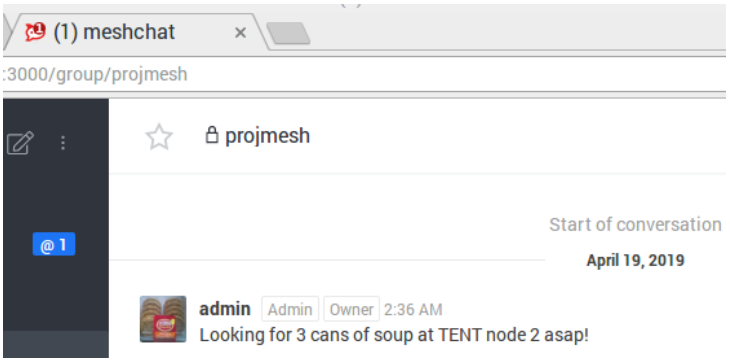

Fig. 14 TENT mesh chat interface

V. CONCLUSION

The main objective of the project demonstrates that a simple lightweight solution, like my T.E.N.T, is not only possible, but also the cost efficient, time efficient and relatively easy to setup solution for any organization needing a quick to deploy network setup. In small camp of humanitarian aid if doctors needed to relay information between temporary infirmaries TENT provides this utility Though not suitable for large mesh networks BATMAN works flawlessly in these conditions. It allows various devices to join and allows for quick expansion, while the overall cost of the network is less than $50 per node. The decentralized aspect of my network allows for it to be easy to repair and very durable to node failure. Overall the T.E.N.T solution is a frame that allows for customization while also allowing for users to solve a multitude of problems. It's because of these reasons that the decentralized Wireless Mesh network is the best solution. Given more experience users could easily enable and configure additional services such as DHCP, DNS, or FTP.

REFERENCES

[1] Sousa, Bachega, et al. "Case study for a highly portable mesh network"

[2] Portman, Marius "Wireless Mesh Networks for public safety and disaster recovery applications" 2006

[3] R. T. do Valle e D. C. Muchaluat-Saade, "MeshAdmin: An integrated platform for wireless mesh network management", *in Proc. Network Operations and Management Symposium (NOMS)*, 2012, pp. 293–301.

[4] Lindner, Marek. *B.A.T.M.A.N. Wiki*. www.open-mesh.org/projects/open-mesh/wiki.

[5] Nginx open source software https://www.nginx.com/resources/wiki/start/topics/tutorials/install/

[6] *Rocket.chat https://github.com/RocketChat/Rocket.Chat.RaspberryPi*

[7] S. Wunderlich https://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00

[8] StackExhange https://unix.stackexchange.com/questions/49626/purpose-and-typical-usage-of-etc-rc-local

[9] Suilji Pi mesh node configuration https://github.com/suiluj/pi-adhoc-mqtt-cluster/wiki/Batman-Adv-and-Batctl

[10] Raspberry Pi https://www.raspberrypi.org/forums/viewtopic.php?t=197035

[11] git https://git-scm.com/