Wireless Mesh Networks under FreeBSD

Rui Paulo
rpaulo@FreeBSD.org
The FreeBSD Project
AsiaBSDCon 2010 - Tokyo, Japan

Abstract

With the advent of low cost wireless chipsets, wireless mesh networks became much more attractive for both companies, governments, and the general consumer. Wireless mesh networks are being used extensively since the popularization of the 802.11 wireless technologies, but usually they worked with the help of layer 3 routing technologies. Since 802.11 didn't provide any kind of support for wireless mesh networks, in 2004, IEEE created the Task Group s (TGs) to develop a new amendment to 802.11 which would define the operation of a wireless mesh network using existing 802.11 hardware and having a routing protocol work at layer 2. Later, the amendment also included provisions for mesh authentication, encryption, link management, bridging mesh networks with other types of networks, and channel reservation. This paper will talk about the FreeBSD implementation of 802.11s that's available in version 8.0 and beyond. This work was sponsored by The FreeBSD Foundation.

1 Introduction

A wireless mesh is a group of inter-networked wireless nodes which forward messages between themselves to create a mesh network. A wireless mesh network is similar to a wired/wireless network, but with some big advantages, namely: automatic global connectivity (if one mesh node is connected to the Internet, all the others mesh nodes can take advantage of this without any additional administration procedure), no single point of failure, it's a self healing network and each mesh node is able to do path discovery. Every wireless node has the ability to route packets, so the network software on each node runs one instance of the routing protocol software. Regarding to routing protocols, there are dozens (almost one hundred) of different routing protocols available today that can be used on wireless mesh networks. This number clearly shows that the best routing protocol to use depends on the conditions, the size and the usage scenario of the wireless mesh network.

Wireless mesh networks are becoming increasingly popular today. Notable examples of wireless mesh networks are: Mesh Dynamic's Military Mesh Network[5] which allows the military forces to communicate without any central node; electricity meters are being deployed with wireless mesh technologies so they can communicate their readings from one to another and finally reaching the the central station; the Iridium satellite constellation has 66 satellites (excluding spares) that form a mesh network and provide voice and data coverage for the entire Earth's surface; the OLPC[6] project builds a laptop suitable for children that has mesh capabilities built in, allowing its users to reach the Internet in places where it would be very hard to do so; the Sonos Multi-Room Music System creates a wireless mesh network so you can listen to music in every room of your home. There's a high probability that your everyday data can be traveling through a

wireless mesh network without you knowing it.

In 2004, IEEE created the Task Group s (TGs)[3] to develop a new amendment to 802.11 which would define the operation of a wireless mesh network using existing 802.11 hardware. In 2008, Linux gained initial support for 802.11s[1]. In 2009, FreeBSD also gained initial support for 802.11s[2] and in late 2009, interoperability problems were resolved between Linux and FreeBSD. These problems existed because Linux had been following an early version of the draft specification while FreeBSD used a more recent one and they were incompatible frame format wise.

This paper will describe the 802.11s amendment and the FreeBSD implementation.

2 802.11s overview

A mesh network based on 802.11s is called a MBSS, Mesh Basic Service Set. MBSS networks are identified by a mesh ID, similar to the well known SSID (Service Set ID) used on regular 802.11 networks.

All the mesh nodes must be using the same radio channel. If one wants to collocate a MBSS network and a BSS network (access point), using non overlapping channels is a good idea. Even better is using the 2.4Ghz band for the one network and the 5Ghz band for the other network, if the local radio regulations allow this.

After a node setups itself and starts broadcasting beacons, it will passively listen for beacons from other mesh nodes. The beacons contain a mesh configuration information element that announces which routing protocol is being used, which authentication protocol is required to peer, which protocol measures the link metric and other mesh parameters. If the mesh ID and the mesh configuration parameters in a beacon from another mesh node match the ones we are using, we can initiate peering with this mesh node. Peering allows the mesh network to accept a new node in its network topology. This new node usually also forwards packets, but this behavior can be disabled. Peering is a 4-way handshake between two mesh nodes. Each mesh node will have a peering ID that it will share with the other mesh node. These IDs are only meaningful on the context of peering. Upon successful peering, the new mesh node is now allowed to exchange packets with the other mesh node.

Routing protocols on 802.11s work at the OSI layer 2, in contrast with many of the current ad-hoc wireless routing protocols which work at the OSI layer 3. The routing protocol to use is agreed between mesh nodes and it must be the same for all the mesh nodes. The 802.11s standard requires implementations to support the HWMP (Hybrid Wireless Mesh Protocol), but is extensible enough that you can even implement your own wireless mesh routing protocol. The “Hybrid” name comes from the fact that HWMP works both in on-demand and proactive modes. When in on-demand mode, HWMP is pretty similar to AODV (Ad-Hoc On-Demand Distance Vector) which is a routing protocol used extensively over the years on MANETs (Mobile Ad-Hoc Networks) and other types of wireless ad-hoc networks. AODV (and HWMP) work by sending path requests when a destination is not found on the routing table. The destination node, if reachable, will send back a path reply. This path exchange will come through all the possible paths between the source and the destination nodes, but the two nodes will only pick the best path (heuristics are based on the number of hops and the path metric). On the proactive mode, a mesh node using HWMP is continuously trying to find paths to all the other mesh nodes by broadcasting path request packets. A node setup to proactively find mesh paths is usually called a root node. The advantage of this mode is that nearby nodes that just joined the mesh can find a path to all mesh nodes by contacting the root node. Although this path setup may be faster, the path it finds may not be the best. The

best path will eventually be found because the reactive mode is being used at the same time. Count-to-infinity problems are avoided using sequence numbers. Radio Aware Optimized Link State Routing Protocol (RA-OLSR) is an example of another routing protocol that can be used with 802.11s. All of this routing happens using 802.11 management action frames and it's transparent the user.

802.11s networks can be bridged with other wireless networks or with wired networks. The node that bridges a 802.11s network with another network is called a Portal. The job of the Portal is to proxy traffic and to announce itself on the 802.11s network.

All this happens without major frame format changes, so the current 802.11 networks won't be disturbed by the present of a 802.11s network. In order to cope with the necessary addition of a Mesh Header in data packets, this header was not added to the MAC header, but instead it was added to the frame body. 802.11s networks will expect this header to be present but regular 802.11 networks will discard its contents. This new field includes some mesh flags, a mesh TTL, a mesh sequence number, and mesh address extensions. These are used respectively for handing path loops or count-to-infinity problems, discarding duplicate unicast/multicast/broadcast frames, and for external address support.

In order to avoid contention on the wireless medium, the 802.11s defines a controlled channel access mode (Mesh Coordinated Channel Access) that restricts the usage of the wireless medium to specific times for each wireless mesh node. This means that, when transmitting, wireless mesh nodes will experience much lower contention. In contrast with the regular CSMA/CA mode used on regular 802.11 networks, this gives much better network performance. This is not yet implemented in FreeBSD.

The 802.11s amendment is under draft status and the final version is expected sometime in 2010 or 2011.

3 Implementation details

The FreeBSD implementation of 802.11s started in April 2009 and was sponsored by The FreeBSD Foundation[7]. The code is around new 4500 LOC including the routing protocol (HWMP) and was committed to the FreeBSD Subversion repository in July the same year. The project had the following goals:

- Implement a mesh VAP;
- Implement mesh peering;
- Implement HWMP;

Encryption and authentication were purposely left off, but will be implemented in the future.

We extended the `net80211` protocol stack to include mesh networking support. The source code to support most of the wireless mesh networking is under `sys/net80211/ieee80211_mesh.c` and the source code that implements HWMP is under `sys/net80211/ieee80211_hwmp.c`. Every wireless driver must have small modifications in order to support mesh networks. As a rule of thumb, any card that can operate as an access point (`hostap`) can also operate as a mesh node.

As we started the development on the FreeBSD 8.0 code base, we could take advantage of the virtual access point (VAP) support. This functionality allows the user to create several wireless system interfaces with just one physical network interface (one radio), allowing the collocation of one mesh interface and one access-point interface using one wireless card. The most significant drawback is that they must operate on the same channel, degrading network performance. In order to create a mesh VAP, the root user must type: `ifconfig wlan0 create wlandev ath0 wlanmode mesh`, provided that `ath0` is the parent network device. After that, the user should pick a channel and a mesh ID:

`ifconfig wlan0 channel 1 meshid freebsd-mesh up`. At this stage, the interface is operational.

Whenever a user requests a mesh VAP, the mesh code will initialize the necessary structures to create one. After that, beacons will begin to be transmitted. If there are other mesh nodes on the same channel and the mesh parameters (mesh ID and mesh configuration) match our own configuration we are ready to start peering. When peering, each mesh node sends one "peer link open" action frame and one "peer link confirm" action frame. When the link is meant to be terminated, the mesh node is supposed to send a "peer link close" action frame. The protocol defines a detailed FSM but explaining it is out of the scope of this paper. More information can be found on the 802.11s amendment.

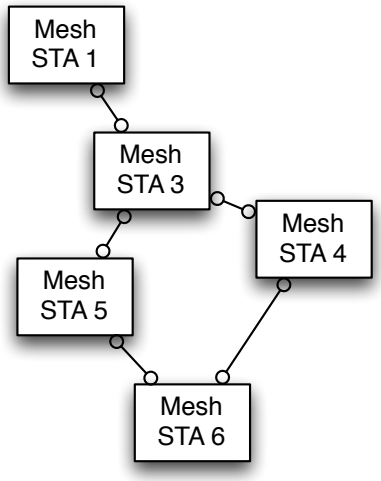
Peering handling source code is located at `ieee80211_mesh.c`. On FreeBSD, you can list the mesh node peers with the command: `ifconfig wlan0 list sta`.

The mesh code is also responsible for forwarding packets and we did this by changing the `mesh_input()` function that checks if the data packet is supposed to be forwarded. `mesh_input()` is a large function that deals with many details of the mesh protocol. The function that injects a forwarded packet (`mesh_forward()`) asks the routing module which is the next hop for this packet. If the packet was forwarded, we don't pass it to `bpf(4)`, otherwise pass it to `bpf(4)` and deliver it to the next layer protocol. Finally, we should note that all forwarding takes place inside the kernel.

If you want to understand the protocol packet format more clearly, we highly recommend installing Wireshark[8] with the 802.11s patches[9].

3.1 Routing implementation details

The routing code finds paths on the mesh network. Whenever a data packet reaches our input function and that function calls the routing code, we need to deliver the packet to the next hop if we know which node is the next hop, or we need to find a path and queue the data packet.



To find a path with HWMP we broadcast an 802.11 action frame containing the Path Request (PREQ) information element. This element contains the source address, the destination address, a sequence number, a TTL and HWMP flags (discussion about the flags is outside the scope of this paper). If the message reaches a node that's the destination address it will send back a 802.11 action frame containing a Path Reply (PREP) information element. This frame is now unicast and goes through the same intermediate node as the PREQ. When the PREP reaches the source address a bidirectional path is established and the intermediate nodes will also have a path to both the source and the destination addresses. Data transmission can now begin and we dequeue the previously queued packets.

When a peer link goes down, all the nodes on the mesh network must be informed so that

they can updated the routing table. HWMP accomplishes this by broadcasting an action frame containing the Path Error (PERR) information element. Intermediate nodes will receive this management frame, update their routing tables and re-broadcast the packet if necessary.

Since each HWMP message has a sequence number associated, we can discard duplicates and refrain doing any action (parse and/or forward) on packets that we have processed previously.

On FreeBSD, you can inspect the mesh routing table with the command: `ifconfig wlan0 list mesh`.

3.2 Implementing your own routing protocol

As we already mentioned, the FreeBSD code is modular enough that you can implement your own routing protocol. To achieve this, you need to initialize your own `struct ieee80211_mesh_proto_path` and then call `ieee80211_mesh_register_proto_path()`. The `ieee80211_mesh_proto_path` structure is detailed below. For brevity we omitted function parameters.

```
struct ieee80211_mesh_proto_path {
  uint8_t  mpp_active;
  char     mpp_descr[...];
  uint8_t  mpp_ie;
  sutruct ieee80211_node *
           (*mpp_discover)(...);
  void     (*mpp_peerdown)(...);
  void     (*mpp_vattach)(...);
  void     (*mpp_vdetach)(...);
  int      (*mpp_newstate)(...);
  const size_t
            mpp_privlen;
  int       mpp_inact;
};
```

The most important elements of this structure are:

- `mpp_descr` which describes the protocol name;
- `mpp_ie` which defines the protocol number included on the mesh configuration information element (IE);
- `mpp_discover` is a callback that should return a pointer to the next hop of a given destination address;
- `mpp_peerdown` is a callback allows `net80211` to report to your routing algorithm that a neighbor mesh node (peer) disconnected itself from the mesh network;
- `mpp_vattach`, `mpp_vdetach`, `mpp_newstate` are callbacks that provide initialization, de-initialization and state change notifications, respectively;

4 Further work

The remaining items required to fully support 802.11s are:

- finish the "external address" support;
- implement authentication and encryption;
- implement congestion signaling;
- implement the neighbor offset protocol;
- implement channel reservation;

As the standard changes volatility regarding to these items, we expect to start working on their implementation when the draft standard reaches the final development phases.

References

[1] *802.11s Linux implementation* `http://www.open80211s.org`

[2] *802.11s Wireless Mesh Networking - FreeBSD Wiki* `http://wiki.freebsd.org/WifiMesh`

[3] *Status of Project IEEE 802.11s - Mesh Networking* `http://www.ieee802.org/11/Reports/tgs_update.htm`

[4] *RFC 3561 - Ad hoc On-Demand Distance Vector (AODV) Routing* http://tools.ietf.org/html/rfc3561

[5] *Robust Mesh Networks for Military and Public Safey* http://www.meshdynamics.com/military-mesh-networks.html

[6] *One Laptop Per Child (OLPC)* http://laptop.org/

[7] *The FreeBSD Foundation* http://www.thefreebsdfoundation.org/

[8] *Wireshark network protocol analyzer* http://www.wireshark.org/

[9] *Open80211s Wireshark patches* http://o11s.org/patches/wireshark/