

M

The UniK - OLSR plugin library

Andreas Tønnesen, Andreas Hafslund, and Øivind Kure

Abstract—In this paper we present a *plugin* for OLSR. This plugin is used for adding an interface for between the olsrd daemon and other applications. Using the plugin, other applications, ranging from network services such as DNS, to broadcast voice traffic can use the broadcast mechanisms from the OLSR protocol. This means that the applications can flood the ad hoc network using the optimized MPR-functionality, and thus reducing the network load.

Index Terms—Mobile Ad Hoc Network, OLSR, dynamically linked libraries, service broadcasting

I. INTRODUCTION

As mobile ad hoc networks (MANETs) [1] are an area for research and development, the ability to add extensions or change normal operation in implementations of routing protocols for such networks, provides a great way of testing new solutions.

The MPR flooding and default forwarding algorithm used in OLSR [2] makes this protocol very interesting to extend. Normal MANET routing suffers from lack of broadcast and multicast solutions. By letting OLSR carry traffic, one can provide a broadcast solution that is optimized. The OLSR daemon will then work as a flooding relay agent for local applications. Other interesting extensions can be updating OLSR parameters at runtime (for instance changing MPR willingness based on power consumption), based on traffic analysis or creating visualizations of the network topology.

Already existing services that require a broadcast mechanism can be used in a MANET routed by olsrd if using an olsrd plugin to flood broadcasted traffic. Such services include DNS, service discovery mechanisms, key distribution schemes etc. Utilizing such protocols in MANETs have been studied in [2] and [3].

As modularity was one of the main goals when designing and implementing the UniK olsrd, the idea of easily extending the protocol led to the design of a *plugin* interface. In this paper this interface, areas of usage and some example plugin implementations are covered.

II. OVERVIEW OF OLSRD PLUGINS

Olsrd supports loading of dynamically linked libraries, called plugins as explained later, for generation and processing of private package-types and any other custom functionality. There are two main reasons for using the OLSR plugins; (i) for sending broadcast traffic in the MANET using OLSR, (ii) for changing the OLSR functionality using the plugin interface. The former is for other applications than the

routing protocol. The latter is for instance running QoS routing, secure routing, or changing the TC message flooding parameters.

A. Dynamically linked libraries

A dynamically loadable library (DLL) is a piece of executable code that contains functions and data. Unlike normal executables, DLLs allows the actual linking to take place at runtime. An application can load and run functions from a DLL dynamically.

DLLs are typically libraries of functions shared by many processes. An example would be a Graphical User Interface (GUI) library. This is, however, not something taken advantage of when using DLL as *plugins*. Plugins provide new functions to an existing application without altering the original application. An illustration of this is shown in Figure 1. Olsrd uses DLLs in this fashion.

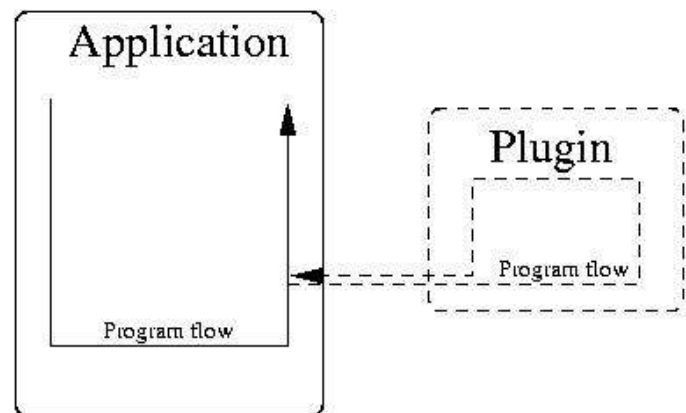


Figure 1: Example of how a plugin intercepts an applications program flow and adds its own.

DLL functionality exists for all common operating system. In Linux they are known as *.so* files while in Microsoft Windows they are known as *.DLL* files.

B. Why use plugins?

The plugin design was chosen for, amongst others, the following reasons:

- No need to change any code in the OLSR daemon to add custom packages or functionality.
- Users are free to implement olsrd plugins and license them under whatever terms they like. Olsrd is GPL licensed meaning that any alteration of the olsrd code itself must be publicly released.
- Plugins can be written in any language that can be compiled as a dynamic library.
- No need for people using extended OLSR functionality to rely on heavy patching to maintain functionality, when new olsrd versions are released. The plugin interface will *always* be backwards

¹Manuscript received July 1st, 2004. This work is supported by THALES Communications AS and UniK – University Graduate Center, Norway.

Andreas Tønnesen and Prof. Øivind Kure is with UniK – University Graduate Center, Norway (e-mails: {andreto, okure}@unik.no).

Andreas Hafslund is with Thales Communications AS, Norway (e-mail: andreha@unik.no).

compatible.

OLSR provides a default forwarding algorithm, that allows for forwarding of OLSR messages of unknown types. This means that even if only a subset of the nodes in the network actually know how to interpret a certain message-type, all nodes will forward them according to the MPR scheme. A wide variety of services designed for wired network environments rely on net-wide broadcasts. Services that needs to broadcast/multicast data can encapsulate data in a private OLSR message-type using an olsrd plug-in as illustrated in Figure 2.

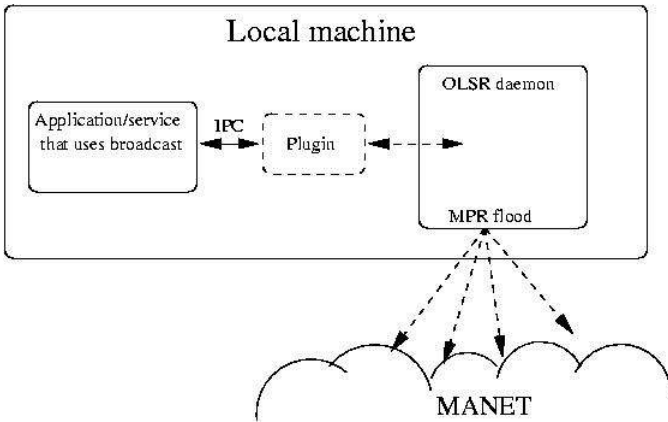


Figure 2: Example of how a plugin can enable the OLSR daemon to work as a relay for broadcasting. The Local application and the plugin communicate using the interprocess communication.

The design of the various entities of olsrd allows one to easily add special functionality into most aspects of the program. One can both register and unregister functions with the socket parser, packet parser and scheduler, and one can update many variables, manipulate all outgoing traffic and more. This opens up for possibilities like intercepting current operation and replacing it with custom actions. As an example, a plugin can provide its own HELLO message generation and parser functions. The plugin can then unregister the default functions used by olsrd and replace them with its own. This means that the OLSR tables can be freed, and a new set of operations can be executed on the tables. This relationship is illustrated in Figure 3.

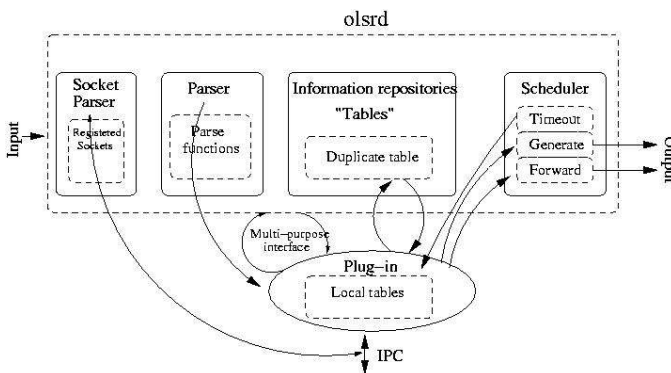


Figure 3: A plugin can manipulate virtually every part of the olsrd daemon.

The modular design of olsrd really shows its strengths when dealing with plugins. A plugin can do things like establishing blocking sockets for communication of its own without blocking olsrd operation. This is because the plugin can register its sockets with the socket parser in olsrd where the socket will be part of the main *select(2)* set.

III. THE OLSRD PLUGIN INTERFACE

For a plugin scheme like this to work, one needs a well-defined easy-to-expand interface for communication between the OLSR daemon and plugins. The interface should be well defined so that a plugin always knows what to expect from the daemon, and the daemon always knows what to expect from the plugin within some given set of functions. The design should be flexible enough to allow for extending the functionality while keeping backwards compatibility.

The actual data that must be set up between the application and the plugin are pointers to variables and functions. The olsrd plugin interface is mainly based upon the function:

```
int olsr_plugin_io(int cmd, void *data, size_t size)
```

This function is similar to the *ioctl(2)* function in syntax. One passes a command and a pointer to some allocated memory and the size of the allocated memory area. The return value indicates success or error, while actual data is put or read from the memory buffer **data*. This function is implemented in *src/plugin.c*, while all the commands are defined in *src/olsr_plugin_io.h*.

Let us look at an example of how a plugin can use a function implemented in olsrd. The function *get_msg_seqno* returns the next message sequence number for OLSR to use when transmitting an OLSR packet. If we want to be able to use this function in our plugin we will typically execute something like:

```
/* Define this function-pointer somewhere */
olsr_u16_t (*get_msg_seqno)();

/* Retrieve the function pointer */
if(!olsr_plugin_io(GETF_GET_MSG_SEQNO,
&get_msg_seqno, sizeof(get_msg_seqno)))
{
    get_msg_seqno = NULL;
    return 0;
}
```

To be able to access the *olsr_plugin_io* function, the plugin needs to be initialized from olsrd. The file *src/plugin_loader.c* implements the plugin loader code. For the plugin loader to be able to set up the needed pointers, the plugin must provide the following function (in addition to some variables and other functions):

```
int register_olsr_data(struct olsr_plugin_data *data)
```

This function is called from the olsr plugin loader passing a pointer to a struct *olsr_plugin_data*, which contains the pointers to olsrd functions that the plugin needs to be able to set up all needed data-pointers. After this the plugin is responsible for fetching all needed pointers from the olsr daemon. The process of initializing a plugin is illustrated in Figure 4.

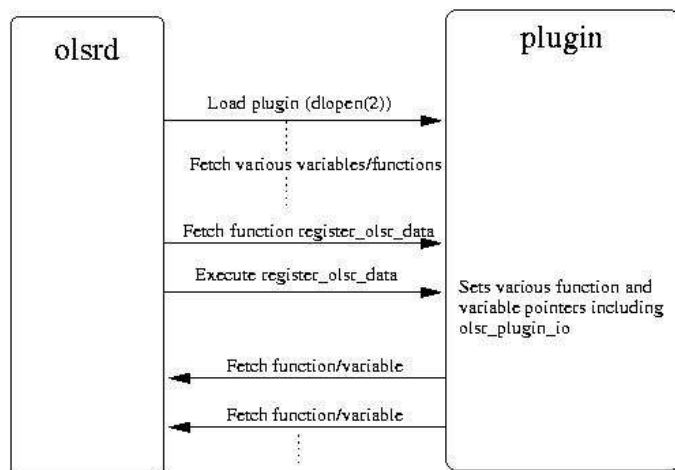


Figure 4: The plugin initialization process.

To make olsrd try to load a plugin at startup, the `LOAD_PLUGIN` directive is used in the configuration file.

IV.TWO EXAMPLE PLUGINS

Two plugins that perform trivial tasks are implemented as example code. Here follows a brief explanation of what they do and how they are designed. Both plugins are part of the olsrd source code package available for download from <http://www.olsr.org>. The example plugin source code resides in the `lib/` directory relative to the olsrd source code root directory.

A. The powerstatus plugin

This plugin is to provide a solution where the powerstatus of nodes running the plugin in the MANET are distributed and registered. This information is made available to the user and other processes through IPC using a TCP socket. The plugin should not affect nodes running without this functionality. The reason we use a TCP socket for the communication, is only to show how the application can communicate with the plugin with standard TCP. This could of course be changed using other IPC techniques.

A node is to periodically flood the network with a custom packet containing the following information:

- Whether or not the node is battery powered.
- Estimated lifetime left on the battery, if battery powered.
- Percentage of power left on the battery, if battery powered.

This should result in a scenario where all powerstatus-enabled nodes have an up-to-date understanding of the powerstatus of all other powerstatus-enabled nodes. Even though the powerstatus-enabled nodes might only be a subset of the nodes in the MANET, the default forwarding algorithm will ensure diffusion of the information. The following is implemented in the plugin:

A message format to carry the power information:

To take advantage of the default forwarding scheme in OLSR, the power information, extracted from `/proc/apm`, has to be transmitted as an OLSR message. This message format is displayed in Figure 5.

The message is encapsulated in a regular OLSR message header with a message type from the OLSR private message types (128-255).

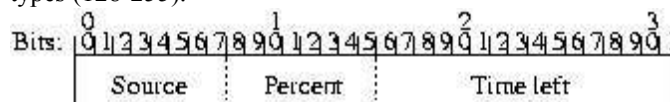


Figure 5: The message format used by the powerstatus plugin. This data is sent as the data part of a regular OLSR message.

An information repository:

To keep an up-to-date database of information of power, an information repository similar to those used in olsrd, is implemented. This is based on hashed linked lists with statically allocated root elements.

Periodic generation of powerstatus messages based on powerstatus:

To transmit messages about the status of power on a periodic interval, a generation function is implemented. This function is registered with the olsrd scheduler at plugin initialization. The function polls the `/proc/apm` file for information about the power status, and builds a message based on the information. This message is then flooded through olsrd.

Parsing of powerstatus message:

A message parse function is registered with the olsrd message parser at plugin initialization to receive all incoming powerstatus messages. This function updates the information repository based on the contents of incoming packages. The function is also responsible for forwarding the message.

Timeout of the repository:

A function that traverses the information repository and removes timed out entries is registered with the olsrd scheduler to run at a given interval. Since it is not critical that timed out entries are removed as soon as possible, this function is not registered to be executed at every olsrd scheduler poll.

IPC functionality:

IPC is done over a TCP socket via the loopback interface. Since this plugin example is not to listen for input, but just to do IPC output, there seems to be no need to register the IPC socket with the olsrd socket parser. However, as we want to be able to connect via this interface to the plugin at any time, the plugin must register the server socket with olsrd to listen for incoming connections. The socket listens for connections on TCP port 8888 and only accepts connections from the local host (127.0.0.1).

B. The dynamic Internet gateway plugin

Nodes in a MANET might dynamically obtain and lose Internet connectivity through interfaces not participating in the MANET routing. A typical scenario would be a laptop that might be connected to the Internet through an Ethernet link for a limited time.

A plugin that dynamically updates the HNA information announced by the local node has been implemented. This plugin checks if the local node has an Internet-connection and updates the local HNA set based on this.

This plugin is a good example of using plugins for other

tasks than packet transmission. Combining this plugin with an automatic network cable detection daemon such as [5] would be a good idea. Only IPv4 is supported by the plugin as of now. This plugin has been used in our recent paper [6].

Detecting Internet routes:

The main object of this plugin is to poll for an Internet route and add or remove such a route from the local HNA set if a change is detected. An Internet connection is identified by a default gateway with a hop count of 0. So a route to 0.0.0.0/0 with metric 0 is considered an Internet route. Since OLSR uses a hop count/metric bigger than 0 on all routes, this plugin will not react to Internet gateways added by olsrd.

To poll for route updates a function that searches the kernel routing table for a default gateway is registered with the olsrd scheduler to be executed regularly on a given interval. If a new Internet route with metric 0 is discovered, the plugin will add this entry to the local HNA set by calling the function:

```
void add_local_hna4_entry(union olsr_ip_addr *net, union
hna_netmask *mask)
```

This function has been fetched from olsrd through the plugin interface.

Whenever such a registered Internet route is removed from the kernel routing table the local HNA entry is also removed using the function:

```
void remove_local_hna4_entry(union olsr_ip_addr *net,
union hna_netmask *mask)
```

This enables nodes to act as Internet gateways whenever they have some Internet connectivity not set up by OLSR itself.

IPC:

The dynamic Internet gateway plugin also offers IPC to read output. Just like with the powerstatus plugin all communication is outbound, but since clients should be able to connect at any time, the IPC server socket is registered with the socket parser of olsrd.

The IPC socket listens on TCP port 9999 and only allows connections from the local host (127.0.0.1).

V. CONCLUSION AND FURTHER STUDY

We have designed and implemented a plugin for the UniK-OLSR routing daemon. The plugin functionality has been used for adding extended functionality to the OLSR. Also, the plugin has been used for sending broadcast traffic different than the regular routing messages. We have used the plugin for:

- A secure extension to the OLSR protocol [7].
- An autoconfiguration mechanism [8].
- Multicasting/broadcasting of Voice using OLSR, to be released this autumn.
- Distributed DNS using OLSR, to be released this autumn.

There are several aspects with the OLSR plugin we wish to improve. One such thing is making the plugin even more modular with respect to the applications. This means that applications using broadcast traffic, could function without any changes with regards to the OLSR plugin. This is not the case today, but will be included in later releases.

ACKNOWLEDGMENT

We would like to thank Roar Bjørgum Rotvik and Jon Andersson at Thales Communications AS for their valuable help during our discussions.

REFERENCES

- [1] J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations". IETF RFC 2501, January 1999.
- [2] T. Clausen, P. Jacquet, A. Laotit, P. Minet, P. Muhlethaler, A. Qayyum, L. Viennot, "Optimized Link State Routing Protocol". IETF RFC 3626 October 2003.
- [3] L. Cheng, "Service advertisement and discovery in mobile ad hoc networks". Technical report, Leigh University, 2002.
- [4] R. Handorean, G. Roman, "Service provision in ad hoc networks", Technical report, Washington University, 2002.
- [5] B. O'Sullivan, "Key netplug". Technical report <http://freshmeat.net/projects/key-netplug/>, 2004.
- [6] P. Engelstad, A. Tønnesen, A. Hafslund, G. Egeland "Internet Connectivity for Multi-Homed Proactive Ad Hoc Networks" In Proceedings of the 2004 International Conference on Communication (ICC 2004), Paris, June 20-24, 2004.
- [7] A. Hafslund, A. Tønnesen, J. Andersson, R. Rotvik, Ø Kure "Secure Extension to OLSR" Currently under review for the OLSR Interop and Workshop, 2004.
- [8] A. Tønnesen, A. Hafslund, P. Engelstad "IP Address Autoconfiguration for Proactive Mobile Ad Hoc Networks" Currently under review for the IEEE SECON 2004, Sensor and Ad Hoc Communications and Networks.