NRL Simplified Multicast Forwarding (SMF) Engine Overview and User's Guide

The NRL Simplified Multicast Forwarding (*nrlsmf*) project includes software for a user-space forwarding engine. This software is was developed by the Naval Research Laboratory (NRL) PROTocol Engineering Advanced Networking (PROTEAN) Research Group. The goal of this effort is to provide an implementation of experimental techniques for robust, efficient distribution of broadcast or multicast packets in dynamic, wireless networks such as Mobile Ad-hoc Networks (MANETs). The *nrlsmf* application can be run as a stand-alone application capable of providing "classic" flooding of broadcast and multicast traffic for a specified network interface or can be used in conjunction with a controlling program to perform more sophisticated multicast forwarding. An interprocess communication "remote control" interface is provided so that a compatible program (e.g. *nrlolsr*) may issue real-time commands to *nrlsmf* to control the multicast forwarding process. Both IPv4 and IPv6 operation are supported. Versions of *nrlsmf* can be built for the following operating systems: Linux, MacOS, BSD, Win32, and WinCE.

Theory of Operation

The *nrlsmf* program acts as a packet forwarding engine by promiscuously "sniffing" packets on a specified interface and then retransmitting packets (using its local network interface MAC address) according to set forwarding rules. Currently, *nrlsmf* only receives and forwards packets on a single, specified network interface for operation within the "routing area" corresponding to that (generally wireless) interface. However, future iterations of *nrlsmf* will also allow for packet reception and forwarding across multiple interfaces to allow for configurable gateway operation. Currently, *nrlsmf* will forward incoming (non-locally generated) packets that are IP Multicast with time-to-live (TTL) greater than one.

Duplicate packet detection is an important facet of wireless network multicast forwarding since packets often must be forwarded on the same interface as they are received. Thus, neighbors' subsequent retransmission of forwarded packets will be "heard" and the local forwarding engine must discriminate between new packets and previously-forwarded packets to avoid unnecessary retransmission. Accomplishing duplicate packet detection of native (unencapsulated) IP packets presents several challenges:

1) While the IPv4 packet "id" field (a 16-bit sequence) number is available to discriminate duplicate packets, its implementation varies in different operating systems (e.g. always set to ZERO, incremented on a "per socket" basis instead of global, etc). Furthermore, when incremented on a global basis (with respect to the originating host's IP packet transmission), other data flows (TCP sockets or other multicast or unicast flows) may cause the field to be incremented haphazardly (or wrap quickly) with respect to a specific flow of traffic to be forwarded by an SMF engine. Ideally, a sequence number that incremented on a per-IP destination basis (with respect to an originating source) would be most useful for duplicate packet detection. There are also

further issues to consider with respect to the possibility of IP security (IPSec) operation or network address translation (NAT) presence that complicate matters here. However, regardless of these issues, the current *nrlsmf* implementation boldly uses the IP "id" field for duplicate packet discrimination. Work is in progress to develop mechanisms to "correct" the IP "id" field (or possibly insert an header option) on hosts participating in networks with SMF.

2) In IPv6, there is no equivalent to the IPv4 "id" field, so duplicate detection is further complicated. The current *nrlsmf* implementation has a partially-implemented feature that attempts to mitigate this by adding a hop-by-hop options header for detected, locally-generated IPv6 flows. Then, incoming flows that have the option header are forwarded. This results in two copies of each generated multicast packet on the first hop, but correct forwarding on subsequent hops. A similar approach could be adopted to "correct" IPv4 "id" fields on systems where this is needed in lieu of outbound packet interception. Again, we are investigating approaches to outbound packet interception for supported operating systems to provide a more efficient experimental capability for long-term use.

3) Also under investigation are approaches to using "hash" functions for duplicate detection. Preliminary results indicate these approaches are complex for per-packet processing and imperfect with respect to false duplicate detection which results in undesirable packet loss, perhaps unrecoverable even with end-to-end reliability mechanisms. But further investigation is merited.

4) Encapsulation of packets for forwarding has some advantages in that discrimination for duplicate packet detection can be carefully controlled. The principle disadvantage is that edge wireless systems (not participating in packet forwarding) without explicit SMF support for decapsulation would be unable to receive multicast transmissions they would have otherwise been able to receive had native IP packet formats been used.

The *nrlsmf* currently forwards based on its configuration and possibly the source MAC address of received packets meeting TTL criteria. If "default forwarding" is enabled, *nrlsmf* will forward all non-duplicate, TTL > 1, multicast packets. Otherwise, *nrlsmf* only forwards packets with MAC source addresses matching those in a list provided by a separate controlling program (e.g. *nrlolsr*). Future versions of *nrlsmf* will feature additional options to control the forwarding process and allow experiments with different algorithms and techniques.

Downloads

Source and binary distributions of *nrlsmf* will be made available at http://downloads.pf.itd.nrl.navy.mil/smf.

The source code is also maintained as a project entitled "Simplified Multicast Forwarding" with a CVS repository at http://pf.itd.nrl.navy.mil. Note that *nrlsmf* depends

upon the NRL Protolib project and the Protolib code must be separately retrieved if the *nrlsmf* code is obtained via CVS.

Build Instructions

For Unix platforms, the "smf/unix" directory in the source tree contains Makefiles for different platforms. Type:

```
make Makefile.<ostype> nrlsmf
```

to build the *nrlsmf* binary executable.

For Win32 platforms (TBD) , a distribution of "winpcap" is required to build the *nrlsmf.exe* executable. A Visual C++ workspace (nrlsmf.dsw) and project files are provided in the "smf/win32" directory for building *nrlsmf.exe*.

For WinCE (a.k.a. PocketPC) platforms, the "RawEther" library is required. The Rawether development kit is a commercial product available from http://www.rawether.net. (Note we are able to provide a binary distribution of *nrlsmf.exe* for WinCE platforms including the required Rawether libraries). A workspace (nrlsmf.vcw) and project files are provided for the Embedded Visual C++ compiler.

Usage

The nrlsmf program may be run as a stand-alone application to perform "classic" flooding

```
nrlsmf interface <ifaceName> [ipv6][resequence {on|off}]
       [defaultForward {on|off}][firewallForward {on|off]
       [firewallCapture {on|off}][debug <level>][version]
       [smfServer <serverName>][instance <instanceName>]
```

where

`interface <interfaceName>`	This specifies which network interface will be monitored for packets and forwarded. The <interfaceName> name may be a canonical name (e.g. "eth0") or an IP address that has been assigned to the desired interface.
`defaultForward {on|off}`	When this option is set to "on", *all* received packets will be forwarded (except those from "blocked" sources (see the *mneBlock* command). With duplicate packet detection, this provides an implementation of "classic" broadcast flooding (limited by TTL hop count). Note this command may be toggled "on/off" via the run-time *nrlsmf* remote-control interface to dynamically control forwarding behavior. And "smfServer" process may wish to control *nrlsmf* this way for relay set algorithms where forwarding does not depend upon previous-hop information.

`ipv6`	This option enables IPv6 support. Specifically, it enables interception of outbound IPv6 packets the *resequence* option is enabled.
`resequence {on|off}`	When set "on", this option causes *nrlsmf* to intercept outbound multicast packets and "resequence" the IPv4 "id" header field, or apply an IPv6 hop-by-hop sequence option header.
`firewallForward {on | off}`	When set "on", this option causes *nrlsmf* to transmit any forwarded packets through the IP stack (via the firewall) instead of the default behavior of raw Ethernet frames. This enables firewall rules, QoS filters, etc. to be applied to outbound packets forwarded by *nrlsmf*. (Note the "firewallForward on" behavior _may_ become default for *nrlsmf* in the future.)
`firewallCapture {on | off}`	When set "on", this option causes *nrlsmf* to capture incoming packets to potentially forward via the firewall _instead_ of the default behavior of capturing frames via system "packet sniffing" facilities. The advantage of this option is that inbound packet firewall rules might be applied to packets. The disadvantage is that SMF previous-hop (source-based) multicast relay (e.g., S-MPR) may not be used since the previous-hop MAC source address information is sometimes not available via the system firewall interface. (The current *nrlsmf* implementation always forwards via the firewall when firewallCapture is enabled. This will be remedied in a future version).
`firewall {on | off}`	This is a shortcut command equivalent to invoking both the `firewallForward` and `firewallCapture` commands simultaneously.
`debug <level>`	Increasing <level> values allow additional debugging information to be output while running.
`instance <instanceName>`	This establishes an alternative "name" for the *nrlsmf* process which is used for interprocess communication. The default <instanceName> is "`nrlsmf`".

`smfServer <serverName>`	This instructs *nrlsmf* to inform (via interprocess communication) a specific server process of its presence, implicitly requesting any current configuration from the named process. The default <serverName> name queried at *nrlsmf* startup is "nrlolsr", unless an alternate <serverName> is given. Note that with *defaultForward* set"on", *nrlsmf* can run as a "stand-alone" application without requiring additional routing information.
`version`	This instructs *nrlsmf* to display version information and exit.
`background`	This instructs *nrlsmf* to run as a "background" process without a console or debug window (Win32/WinCE only).

Note the combination of "defaultForward off" and an empty "forwardMac" list (see below) results in no packet forwarding. In this case, the "defaultForward" command may be toggled from "off" to "on" and vice versa via the "Remote Control" interface (see below) to control "classic flooding" operation.

Example Usage:

On Linux, nrlsmf may be instantiated as follows to work in conjunction with nrlolsr and provide resequencing of IP "id" fields:

```
nrlsmf interface eth1 resequence on
```

On WinCE or Win32 the syntax could be:

```
nrlsmf interface 192.168.1.101
```

In the Linux example, the interface named "eth1" is presumed to be a wireless interface for MANET and in the Win32, the interface can be identified by its IP address. Packet "resequencing" is not currently supported on Win32/WinCE platforms, but these platforms generally provide appropriate sequencing in the first place. This will be addressed in future releases.

Remote Control Interface

The *nrlsmf* application provides an interprocess communication "remote control" interface for receiving run-time instructions from other processes. The "remote control" interface is identified by a canonical name. The default name used by *nrlsmf* is, strangely enough, "`nrlsmf`". However, an alternative name can be specified using the *instance* command-line option described above. This might be useful if multiple *nrlsmf* instances are required to cover multiple network interfaces.

On UNIX systems, the <instanceName> corresponds to a Unix-domain datagram socket named "`/tmp/<instanceName>`" that is opened and monitored for commands (thus the default *nrlsmf* Unix-domaind socket would be identified as "`/tmp/nrlsmf`"). On

WIN32 systems, a "mailslot" named "`\\.\mailslot\<instanceName>`" is created and used while on WinCE systems a semaphore is instantiated along with a corresponding registry entry mapping to a locally-bound socket provides equivalent functionality.

Note that all of the command-line options described above can be issued to *nrlsmf* via its remote control interface at run-time. This even includes the instance command which will cause *nrlsmf* to begin listening for commands on a possibly alternate named interface. In addition to the commands described above, some additional commands are available only through the remote control interface. These include:

`smfServerStart <serverName>`	This informs *nrlsmf* that a remote server has started with its own remote control interface of <serverName>. Controlling processes should use this command instead of the "smfServer" command describe above as a command-line option. At this time, *nrlsmf* does not issue any commands to remote servers other than "smfClientStart <instanceName>", which is done in response to the "*smfServer*" command above. Future *nrlsmf* features may require richer interaction with the controller or use this interface for statistics reporting to the controller.
`forwardMac <binaryMacAddrs>`	This command informs *nrlsmf* of the set of source MAC addresses for which it should forward multicast packets. This is applicable only when *defaultForward* is "off". This <binaryMacAddrs> is a concatenated byte array of 6-byte MAC addresses in network (Big Endian) byte order.
`mneBlock <binaryMacAddrs>`	This command is used to provide *nrlsmf* with a list of source MAC address whose transmitted packets it should completely ignore. The intended use of this command is for testing in NRL's original Mobile Network Emulation (MNE) environment, but could be leverage for other purposes. It is only available, if *nrlsmf* is compiled with the C macro "MNE_SUPPORT" defined. As with the forwardMac command, the <binaryMacAddrs> is a concatenated byte array of 6-byte MAC addresses in network (Big Endian) byte order.

This document will be updated as the *nrlsmf* code evolves.