nrlsmf User's Guide

Abstract

The NRL Simplified Multicast Forwarding (*nrlsmf*) project includes software for a user-space IP Multicast forwarding engine. This software 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 IP Multicast packets in dynamic, wireless networks such as Mobile Ad-hoc Networks (MANETs).

Table of Contents

1. Overview

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. *nrlolsrd*) may issue run-time commands to *nrlsmf* to dynamically 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. The code base also supports working simulation models of SMF in the *ns-2* and OPNET discrete event simulation environments. In the future it is hoped that the SMF functionality might be included as part of operating system kernels.

2. Theory of Operation

The *nrlsmf* program acts as a user-space packet forwarding engine by promiscuously "sniffing" packets specified interfaces or intercepting packets using operating system firewall facilities and then retransmitting packets (using its local network interface MAC address) according to set forwarding rules. Packets can be received and possibly forwarded on a single, specified network interface for operation within a "routing area" corresponding to that single (generally wireless) interface. And, *nrlsmf* also provides for packet reception and forwarding across multiple interfaces to allow for configurable gateway operation and/or multicast distribution across a set of interfaces. Only incoming (non-locally generated), non-link-local IP Multicast packets with time-to-live (TTL) greater than one are considered for forwarding. 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.

3. Usage

The *nrlsmf* program provides a command-line syntax with which it can be launched. Many of the same commands (see Commands) available via the command-line may be applied during run-time using the "remote control" inter-process control interface (see Remote Control Interface). To launch *nrlsmf*, use the following command-line syntax:

```
nrlsmf [ipv6][resequence {on|off}][nhdp <ifaceList>]
       [add [<group>,]{cf|smpr|ecds|push|rpush|merge|rmerge},<ifaceList>]
       [remove {<group> | [<group>,]<ifaceList>}
       [cf <ifaceList>][smpr <ifaceList>][ecds <ifaceList>]
       [push <srcIface>,<dstIfaceList>][rpush <srcIface>,<dstIfaceList>]
       [merge <ifaceList>][rmerge <ifaceList>]
       [clear {<ifaceList>|all}]
       [unpush <srcIface>,<dstIfaceList>][unmerge <srcIface>,<dstIfaceList>]
       [forward {on|off}][relay {on|off}]
       [unicast {unicastPrefix | off}]
       [dscpCapture <dscpValue>,<dscpValueList>]
       [hash <algorithm>][idpd {on | off}][window {on|off}]
       [firewallForward {on|off}][firewallCapture {on|off}]
       [tap <tapInstanceName>]
       [instance <instanceName>][debug <debugLevel>][log <logFile>

nrlsmf [version][ipv6][firewallForward {on|off}][firewallCapture {on|off}
       [add [<group>,]{cf|smpr|ecds|push|rpush|merge|rmerge},<ifaceList>]
       [remove {<group> | [<group>,]<ifaceList>}
       [cf <ifaceList>][smpr <ifaceList>][ecds <ifaceList>]
       [push <srcIface>,<dstIfaceList>] [rpush <srcIface>,<dstIfaceList>]
       [merge <ifaceList>][rmerge <ifaceList>]
       [forward {on|off}][relay {on|off}][delayoff <value>]
       [device <vifName>,<ifaceName>[,<addr>[/<maskLen>][,<addr2>[/<maskLen>] ...]]]
       [rate [<iface>,]<bits/sec>][queue [<iface>,]<queueLimit>]
       [unicast {unicastPrefix | off}]
       [dscpCapture <dscpValue>,<dscpValueList>]
       [dscpRelease <dscpValue>,<dscpValueList>]
       [ihash <algorithm>][hash <algorithm>]
       [idpd {on | off}][window {on | off}]
       [instance <instanceName>][smfServer <serverName>]
       [resequence {on|off}][ttl <value>][boost {on|off}]
       [debug <debugLevel>][log <debugLogFile>]
```

3.1. Usage Examples

To run *nrlsmf* to modify outbound, locally-generated multicast packets for duplicate packet detection *and* to use "classical flooding" (CF) forwarding rules for packets received/forwarded on MANET interface "eth1":

```
nrlsmf resequence on cf eth1
```

Note that a MANET SMF forwarding rule like "`cf`" (or the others including "`smpr`", "`ecds`", etc) can be applied on a single interface or a group of interfaces. A single instance of *nrlsmf* will distribute traffic among all interfaces. If independent interfaces (or groups) are desired, then the user should invoke multiple instances of *nrlsmf*. For example, the syntaxes:

```
"nrlsmf cf eth0 cf eth1"
```

and

```
"nrlsmf cf eth0,eth1"
```

are equivalent. Both result in flooding of packets received on interfaces "eth0" and "eth1" to/among both of these interfaces. If two *separate* instances of *nrlsmf* are launched (one per interface) using the syntax

```
"nrlsmf cf eth0; nrlsmf cf eth1"
```

then the flooding is limited to each respective interface (i.e. packets received on "eth0" are possibly retransmitted only on "eth0"). Future versions of *nrlsmf* _might_ provide the ability to set up multiple interface groupings with a single *nrlsmf* instance.

Using *nrlsmf* as a "gateway" between a typical network interface (e.g. Ethernet) and a MANET interface (e.g. 802.11 in ad hoc mode) can be accomplished using the "push" or "merge" commands (or their "r" (resequencing) variants). For example if it is desired to "push" multicast traffic from a non-MANET interface (perhaps connected to a multicast router) "eth0" to a MANET interface "eth1", resequencing (or adding IPv6 SMF_DPD option header), run "classical flooding" on the MANET interface, _and_ push multicast traffic originated on the MANET area to the non-MANET interface, the following syntax could be used:

```
nrlsmf rpush eth0,eth1 push eth1,eth0 cf eth1
```

Here the "`rpush the0,eth1`" command forces relay of traffic from eth0->eth1 and labels the packets (via the IPv4 ID field or IPv6 SMF_DPD option) for proper SMF duplicate packet detection. The "`push eth1,eth0`" command directs traffic from the MANET interface "eth1" to the non-MANET "eth0" interface, assuming that any traffic on MANET interface has already been "sequenced" (or tagged) appropriately for SMF duplicate packet detection. Note that *nrlsmf* maintains duplicate packet detection state for the "rpush" associate so that packet relayed from eth0 to eth1 are not reintroduced back to eth0 from eth1 even when other neighboring MANET nodes may retransmit those packets to support SMF data dissemination. Finally, the "`cf eth1`" command provides "classical flooding" on the presumably MANET eth1 interface (i.e. retransmitting received multicast packets so that other neighbors out-of-range of the heard transmission may receive the packets). Note that the "`smpr`" or "`ecds`" commands could be used here instead of "`cf`" for more efficient group data distribution if supporting neighborhood discovery protocol (NHDP) (or equivalent) information is operating and controlling *nrlsmf* state. Note that it is planned to integrate NHDP support into *nrlsmf* in a future version.

4. Commands

The *nrlsmf* program supports a variety of commands that can be invoked via command-line at startup or at run-time using the "remote control" interprocess communication interfaces that is provided. There a four rough categories of commands. These include:

1. "Operating Mode Commands" to control or enable general *nrlsmf* capabilities and features.

2. "MANET Interface Commands" to enable SMF operation on or among sets of presumably wireless network interfaces.

3. "Gateway Commands" to enable relaying of multicast to/from sets of interfaces. These commands can allow injection/extraction of multicast traffic for other interfaces to/from MANET interfaces to help create gateways between SMF and other multicast routing domains.

4. "Remote-only Commands" for dynamic, run-time control of specific aspects of *nrlsmf* operation. These commands are not intended for use via the command-line interface.

It should be noted that all of the commands listed here can be invoked via the run-time remote control interface (see Remote Control Interface). However, some commands, particularly the "Operating Mode Commands" are not intended to be invoked after initial startup. And, in general, the "Operating Mode Commands" SHOULD be invoked *before* other commands at startup. In some cases (firewallForward, etc), it is *required* that these commands precede "Interface" or "Gateway" commands.

Table 1. Operating Mode Commands

`ipv6`	If the `resequence` or various `firewall` interface commands are used, this enables setup of IPv6 instantiations of those options. This is an option since not all operating systems provide complete user space process access to the firewall chain for IPv6 traffic.

`forward {on | off}`	Global control of all packet forwarding. If the "`forward off`" command is applied, all packet forwarding is disabled. This command may be invoked at run-time. (default = "on")
`hash {MD5 | CRC32 | SHA1 | NONE}`	When I-DPD is disabled ("`idpd off`") and a hashing algorithm is enabled via this command (i.e., <`algorithm`> not equal to "NONE" (case-insensitive)), *nrlsmf* uses a hash-based duplicate packet detection (H-DPD per the SMF specification) approach to uniquely identify packets for forwarding. If this "`hash`" command is used in conjunction with "`idpd on`", I-DPD is performed for all packets except non-fragmented, non-IPSec IPv6 packets. For those excepted IPv6 packets, H-DPD is used and the SMF "Hash Assist Value" (HAV) is applied as needed when the "`resequence on`" option or "`rpush/rmerge`" operation is configured. Note that, depending upon the hash algorithm selected, this technique may be somewhat processing-intensive compared to the default approach that assumes an embedded unique identifier ("packetId") in packet headers. The "`hash`" and "`ihash`" commands are mutually exclusive and their use supercedes any prior hashing configuration. If I-DPD is disabled and no hash algorithm is specfiied, no duplicate packet detection occurs. (default = "NONE")
`ihash {MD5 | CRC32 | SHA1 | NONE}`	This command is similar to the "`hash`" command described above in that a hashing <`algorithm`> type is specified for *nrlsmf* use. However, H-DPD operation is not invoked as a result of this command. Instead, the specified hash algorithm is used to calculate an "internal hash" value that is concatenated with the explicit I-DPD identifier for duplicate packet detection. This mode of operation enhances I-DPD operation, making it less vulnerable to denial-of-service attacks against SMF where spoofed packets with misleading packet identifiers may be generated by bad-behaving nodes. Note that *nrlsmf* "`ihash`" behavior when I-DPD is disabled (i.e., "`idpd off`") is undefined but will generally result in essentially H-DPD operation for forwarded flows. The "`hash`" and "`ihash`" commands are mutually exclusive and their use supercedes any prior hashing configuration. If I-DPD is disabled and no hash algorithm is specfiied, no duplicate packet detection occurs. (default = "NONE")
`idpd {on | off}`	When enabled ("on"), the ID-based Duplicate Packet Detection (I-DPD per the SMF specification) is performed. Note that *nrlsmf* allows a hybrid of I-DPD and H-DPD to be used for packet identification via the "`ihash`" command. If I-DPD is disabled and no hash algorithm is specfiied, no duplicate packet detection occurs. (default = "on")
`window {on | off}`	When enabled ("on"), *nrlsmf* will use a windowed, sequence-number approach for duplicate packet detection. When disabled, *nrlsmf* uses a table-based approach that only requires unique packet identification without any requirement of ordered, sequenced packet identifiers. When "`hash`" mode is enabled, *nrlsmf* always uses the table-based approach to duplicate packet detection regardless of the "`window`" setting. Also when window is enabled, the `hash` type is automatically set to "`NONE`" and `idpd` operation is enabled. (default = "off")
`resequence {on|off}`	Enables/disables interception and resequencing of locally-generated, outbound IP multicast packets. For IPv4 the packet ID field is modified and for IPv6 the SMF_DPD option header is applied. Packets that subject to IPSec are not modified. The `resequence` option leverages user space access to the operating system firewall chain to intercept outbound IP traffic.
`ttl <hopCount>`	When `<hopCount>` is greater than ZERO, this command causes nrlsmf to intercept outbound, locally-generated IP multicast packets and set their TTL to the fixed hop-count specified by the `<hopCount>` value. If the `<hopCount>` value is less than or equal to ZERO, the TTL modification is disabled. This is provided for experimental use only and is NOT recommended for general use. (default = disabled).

`firewallForward {on | off}`	When enabled, packets are forwarded through the host's IP stack, as opposed to the default method of generating raw MAC layer frames. With "`firewallForward on`", forwarded packets should be subject to IP firewall and queuing rules set up for applicable outbound interfaces. The default mechanism of raw MAC layer frame generation essentially bypasses the IP stack and any such configured IP policies or rules. When used, the "`firewallForward`" command *MUST* precede any "Interface" or "Gateway" commands on the command-line. (default = "off")
`firewallCapture {on | off}`	When enabled, packets are captured using operating system IP firewall capabilities instead of the default raw MAC layer frame capture. It should be note that when this is enabled, that packets are captured after any re-assembly of fragmented IP packets has occurred due to MTU limits. The default mechanism of capturing raw MAC layer frames will not properly forward any IP packets that have been fragmented in the current *nrlsmf* implementation. When used, the " `firewallCapture`" command *MUST* precede any "Interface" or "Gateway" commands on the command-line.(default = "off")
`tap <tapInstanceName>`	This command instructs *nrlsmf* to pass packets that would be forwarded to an external process instead of forwarding them. Note, however, that the external process may use the interprocess "`smfPkt`" command to return these packets back to the nrlsmf process for actual forwarding. The external process can use the "`smfPkt`" command and its format to send interprocess messages with packet content to *nrlsmf* instances identified by their `<instanceName>`. See the "`smfPkt`" command description for further information. This allows an external process to "tap" into the SMF forwarding plane and perform additional, custom filtering of forwarded packet flows. A simple, pass-through example application is included in the *nrlsmf* source code distribution in the file "tapExample.cpp".
`instance <instanceName>`	Assigns a name to the running *nrlsmf* process. This name is used to name the ProtoPipe(MESSAGE) mechanism (Unix-domain socket on Unix, mailslot on Win32, registry-entered host-local UDP socket on WinCE) for interprocess communication that is monitored for "remote control" commands (including most of the command-line options listed in this table). (default = "nrlsmf")
`smfServer <processName>`	Specify a compatible, external process that *nrlsmf* will inform of its status. (*nrlsmf* will signal that process with a "`smfClient <instanceName>`" message via interprocess communication). It is expected that such an external process may wish to have dynamic control of *nrlsmf*. This signaling allows the *nrlsmf* and external processes to be started independently in any order. (default external process name = "nrlolsr")
`unicast {unicastPrefix | off}`	Specify an IP prefix of unicast packets that will be intercepted by *nrlsmf*. These packets are captured through an *iptables* rule and follow the regular processing path of multicast packets, including DPD checks, and ECDS. This command assumes that the unicast packets will be processed by a separate process that interacts with <emphasize>nrlsmf</emphasize> through the <emphasize>tap</emphasize> mechanism.
`dscpCapture <dscpValue>,<dscpValueList>`	Specify a list of DSCP values for unicast packets that will be intercepted by *nrlsmf*. This command assumes that a <emphasize>unicast</emphasize> command is also being used.
`debug <debugLevel>`	Sets the *nrlsmf* debug level. The higher the <debugLevel> value, the more verbose the debug output. (default = 0)
`log <logFile>`	Redirects *nrlsmf* debug output to a file instead of `stderr`.
`boost {on | off}`	Enables/disables "priority boost" (process priority) for the *nrlsmf* process. It may be useful to disable "priority boost" for debugging purposes. (default = 'on')

The "MANET Interface Commands" apply to configuring specific, usually wireless, interfaces for MANET SMF operation. As such, packets inbound on such "MANET" interfaces may also be forwarded by *nrlsmf* back out the same interface as well as possibly other interfaces if so configured. The "`resequence`" command listed here should be used to ensure that packets generated are properly labeled using the IPv4 ID field or the IPv6 SMF_DPD header option for SMF duplicate packet detection.

Table 2. MANET Interface Commands

`add [<group>,]{cf|smpr|ecds|...},<ifaceList>`	Creates a new interface group or adds interfaces to an existing group and configures the group for a specific relay algorithm (e.g., `cf`, `smpr`, `ecds`) or a "`push`", "`rpush`", "`merge`", or "`rmerge`" gateway interface group can also be managed instead of an SMF relay algorithm. Note if the `<group>` name is omitted, an implicit group name based on the relay algorithm or given gateway command is used.
`remove {<group> | [<group>,],<ifaceList}`	Removes either an entire `<group>` or the given `<ifaceList>` interfaces from a specific `<group>` or from all groups if the `<group>` name is omitted. The `<group>` name should be that of an explicit group name previous created with the **add** command or the implicit relay algorithm or gateway mode.
`cf <ifaceList>`	Enables classical flooding (CF) with duplicate packet detection among one or more listed interfaces. Note packets received on a given interface may be forwarded out the same interface as well as on other listed interfaces. The `<ifaceList>` is a comma-delimited (no spaces) list of interface names.
`smpr <ifaceList>`	Enables S-MPR relaying with duplicate packet detection among one or more listed interfaces. Note packets received on a given interface may be forwarded out the same interface as well as on other listed interfaces. The `<ifaceList>` is a comma-delimited (no spaces) list of interface names.
`ecds <ifaceList>`	Enables E-CDS relaying with duplicate packet detection among one or more listed interfaces. Note packets received on a given interface may be forwarded out the same interface as well as on other listed interfaces. The `<ifaceList>` is a comma-delimited (no spaces) list of interface names.
`relay {on | off}`	This command controls the *nrlsmf* status as a relay for interfaces that have E-CDS forwarding enabled. Note that S-MPR relay status is controlled by the selector MAC address list supplied to *nrlsmf* during run-time. (default = "off")
`device <vifName>,<ifaceName>[,<addrList ...>]`	This command ...
`rate [<ifaceName>,]<bits/sec>`	This command ...
`queue [<ifaceName>,]<queueLimit>`	This command ...

The "Gateway Commands" apply to configuring *nrlsmf* to support a forced relay of packets to/from MANET/non-MANET interfaces. This can allow creation of gateways from infrastructure IP networks (e.g., the Internet) with IP Multicast routing support to dynamic MANET areas that rely upon SMF for group data distribution. The commands provided here are designed to have a great deal of flexibility, so some care must be taken to avoid loops with the modes (i.e., "`rpush`" or "`rmerge`" commands) that "resequence" (or tag) packet flows for duplicate packet detection. The *nrlsmf* application has some logic in its command-parsing to avoid some of these scenarios, but the flexibility allowed permits some configurations that may produce undesirable effects.

Table 3. Gateway Commands

`push <srcIface>,<dstIfaceList>`	Force relay of packets from `<srcIface>` to listed destination interfaces. The `<dstIfaceList>` is a comma-delimited (no spaces) list of interface names. Packet forwarding is subject to duplicate packet detection and TTL/hop limit constraints. Useful for setting up an SMF "gateway" to inject packets to/from a MANET SMF area.

`rpush <srcIface>,<dstIfaceList>`	Resequence (modify IPv4 ID field or add IPv6 SMF-DPD option header) and force relay of packets from `<srcIface>` to listed destination interfaces. The `<dstIfaceList>` is a comma-delimited (no spaces) list of interface names. IPSec marked packets are not modified. The potential for reverse relaying of packets is avoided by keeping DPD state on the `<srcIface>` using the resequenced identifiers. Useful for setting up an SMF "gateway" to inject packets from a non-MANET (e.g., infrastructure Internet) interface where sources may not be applying proper sequencing to support SMF duplicate packet detection. Note that the packet identifier "resequencing" (or "tagging") performed here is for "forwarded" (i.e. relayed) packets and is distinct from the "`resequence`" command that applies to locally-generated IP Multicast packets.
`merge <ifaceList>`	Force relay of packets from any listed interface to all other listed interfaces. The `<ifaceList>` is a comma-delimited (no spaces) list of interface names. Packet forwarding is subject to duplicate packet detection and TTL/hop-limit constraints. Useful for setting up an SMF "gateway" to bridge together different networks using essentially classical flooding among the interfaces, but not ever retransmitting the packet back to the inbound interface.
`rmerge <ifaceList>`	Resequence and force relay of packets from any listed interface to all other listed interfaces. The `<ifaceList>` is a comma-delimited (no spaces) list of interface names. Note the potential for packet ping-pong due to reverse relaying from any MANET interfaces in the `<ifaceList>`. Thus, this command should be used with caution and forethought, or maybe never. Note that the packet identifier "resequencing" (or "tagging") performed here is for "forwarded" (i.e. relayed) packets and is distinct from the "`resequence`" command that applies to locally-generated IP Multicast packets.

The "Remote-only Commands" listed here can be invoked only via the run-time, interprocess remote-control interface (see Remote Control Interface). The principal reason these commands are available only via inter-process communication is that the commands have a partially binary format to convey lists of MAC addresses to control aspects of *nrlsmf* operation. In each of these commands, an ASCII prefix is given to indicate the command type, followed by an ASCII "space" character, and then the binary list of concatenated 6-byte Ethertype MAC addresses (in Big Endian order). The length of the datagram command sent to *nrlsmf* implies the length of the MAC address list. The "selectorMac" and "neighborMac" apply to control of S-MPR forwarding. The "mneBlockMac" command is provided to support proper SMF behavior in the NRL Mobile Network Emulation (MNE) environment.

Table 4. Remote-only Commands

`selectorMac <binary macAddrArray>`	This command can be used by external processes (e.g., *nrlolsrd*) to control *nrlsmf* S-MPR forwarding. This command sets the list of MAC addresses that have selected the local *nrlsmf* node as a Multi-Point Relay (MPR). The format of the `<binary macAddrArray>` is a list of 6-byte Ether-type MAC addresses. Thus, the length of this "array" is a multiple of 6 bytes. An ASCII "space" character delimits the literal "`selectorMac`" command string from the binary array of MAC addresses.
`neighborMac <binary macAddrArray>`	This command can be used by external processes (e.g., *nrlolsrd*) to control *nrlsmf* S-MPR forwarding. This command sets the list of MAC addresses that have been identified as symmetric, 1-hop neighbors of the local *nrlsmf* node. The format of the `<binary macAddrArray>` is a list of 6-byte Ether-type MAC addresses. Thus, the length of this "array" is a multiple of 6 bytes. An ASCII "space" character delimits the literal "`selectorMac`" command string from the binary array of MAC addresses.
`mneBlockMac <binary macAddrArray>`	This command enables the *nrlsmf* process to be compatible with the NRL Mobile Network Emulator (MNE) system that uses MAC-based blocking to emulate the connectivity of mobile network topologies. The command sets the list of addresses that should be blocked (i.e., are not reachable as 1-hop neighbors) of the local *nrlsmf* node. The NRL MNE processes use this command as needed. This com-

	mand is only available when *nrlsmf* is compiled with the MNE_SUPPORT macro enabled.
`smfPkt <binary packet info>`	This interprocess message is generated by *nrlsmf* and sent via "remote control interface" interprocess communication to the process identified by the "tap" command, if applicable. Additionally, this same message can be sent to *nrlsmf* to pass packets back to *nrlsmf* for forwarding. The format of the `<binary packet info>` is described below.

4.1. The "smfPkt" Command Format

The "`smfPkt`" command is used to pass packets to/from *nrlsmf* and external processes. This allows for external processes to insert themselves into the SMF forwarding process and provide additional filtering, etc of packets that SMF would forward. The "argument" of the "`smfPkt`" command is actually a binary format that includes a header with forwarding information and an entire Ether-type frame structure (i.e., 14-byte Ether-type header and IP packet payload). Since the "smfPkt" string and its delimiting ASCII "space" character is a total of 7 bytes in length, the binary content begins at the 8th byte of the "smfPkt" command message.

The binary content begins with 1 byte that is an "indexCount", that indicates the length of a list of interface indices that immediately follows. Each interface index is one byte. The first interface index in the list identifies the the network interface upon which the subject packet arrived (i.e., the "inbound interface index"). The remaining interface indices in the list (the "outbound interface index list") are the interfaces on which the subject packet will be (or would have been) forwarded by the *nrlsmf* process. Again, note that the "indexCount" is one greater than the number of outbound interface indices (i.e., the count includes the inbound interface index).

Immediately following the list of interface indices, the packet, in Ether-type frame format, is included. This includes a 14-byte Ethernet frame header and then the frame payload that is presumably a complete IP Multicast packet.

The format of <binary packet info> field of the "smfPkt" is summarized as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   indexCount  | inboundIfIndex|       outboundIfIndex list    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               +-+-+-+-+-+-+-+-+
|                       ...                     |               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               |
|                                                               |
|                Ether-type frame w/ IP packet                  |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5. Run-Time Remote Control

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 <*instanceName*>. 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-domain 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 UDP socket provides equivalent functionality.

6. Current Limitations

There are limitations in the use of some *nrlsmf* options. Many of these limitations are a result of *nrlsmf* being a cross-platform, user-space implementation. Many of these subtleties could be overcome with a kernel implementation of the code.

For example, to take advantage of configured firewall rules and/or quality-of-service (QoS) policies for multicast data transmission and forwarding, it may be desirable to use the *nrlsmf* "`firewallForward`" and/or "`firewallCapture`" options. There are some limitations associated with the user-space mechanisms available to capture and "forward" packets in this manner.

1. The "`rpush`" and "`rmerge`" commands must be used very carefully when used in combination with the "`firewallCapture on`" option:
   - The "`firewallCapture`" option doesn't necessarily provide the source MAC address of received packets properly:
     - On Linux, locally generated packets have some random source MAC address from the 'ip_queue' capture mechanism (thus can’t detect it is receiving packets it previously sent and the resequencing bypasses DPD and a packet "cyclone" to TTL=0 results)
     - On BSD/MacOS, ProtoDetour doesn’t get the source MAC address at all for the "firewallCapture" mode (“layer 2” firewall rule would be needed).
   - So only use "`firewallCapture on`" when absolutely necessary.
2. For IPv4 resequencing, the ID value of ZERO is avoided since some operating systems (e.g., BSD) will automatically re-ID packets that have an ID value of ZERO when "`firewallForward on`" is used.
3. If large multicast packets are sent by hosts that require IP fragmentation, "`firewallCapture on`" must be used for SMF forwarding to work (SMF duplicate packet detection doesn't like fragments and the default Ethernet frame capture mode gets individual fragments while the "`firewallCapture on`" mode gets fully re-assembled IP packets).
4. IPv6 operation is more limited and creates a challenge to apply queuing rules or traffic shaping to forwarded IPv6 traffic:
   - IPv6 raw sockets don’t allow full control of IP packet header as IPv4 raw sockets do.
   - Thus the current “firewallForward” using raw socket for forward doesn’t work.
   - BSD firewall “divert” option evidently work with IPv6 evidently (so no BSD/MacOS “firewallCapture” or “firewallForward

Some of these limitations may be addressed in future versions of *nrlsmf* if possible. It may be possible in some cases that system configuration (perhaps using virtual interface capabilities) may provide work-around solutions for some of these issues.

7. Future Plans

There are a number of additional features and refinements planned for the *nrlsmf* implementation. Some of these (in roughly priority order) include:

1. Built-in support for the Neighborhood Discovery Protocol (NHDP)
2. Option to apply "taggerID" field as described in SMF Internet Draft.
3. Option to load a "config" file for complex configurations

4. Replication of intercepted outbound locally-generated multicast packets to multiple interfaces (i.e., instead of the usual host transmission of multicast on a single specified interface.

5. Filters for applying per-group (destination multicast address) forwarding rules (i.e., to support use of "nrlsmf" as a forwarding engine for experimental forms of group-specific multicast routing).

6. Alternative packet capture/forwarding mechanisms (e.g., virtual interface mechanisms).