“nrlsmf” Update

(multi-interface support, etc)

68th IETF - Prague

21 March 2007

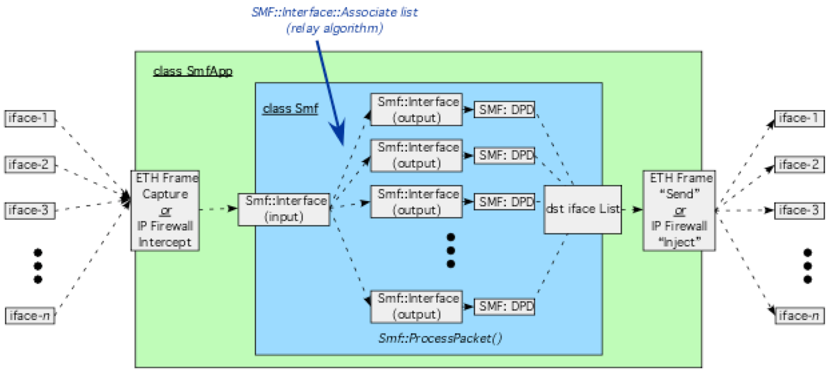
Justin Dean/ Brian Adamson

NRL Code 5522

Overview

- The *nrlsmf* source code has been updated:
  - Multi-interface support, including “gateway” modes of operation.
  - Proper application of IPv6 DPD option header per SMF Internet Draft
  - Proper handling of IPSec packets
- This entails new code architecture and subsequently usage changes
- MANET PacketBB implementation completed (“class ManetMsg”, etc) and NHDP development in progress. (See “protolib” tree for ManetMsg implementation)

The New “nrlsmf” Architecture



Note: A “Smf::Interface” may “associate” with itself for MANET interface forwarding

“nrlsmf” Implementation

- C++ “class SmfApp” is the working application daemon:
  - Uses ProtoCap and/or ProtoDetour classes for packet capture and forwarding.
  - Provides a ProtoPipe for “remote control” of operation
- “class Smf” is the core SMF packet processing module that maintains a list of Smf::Interfaces with associations from “input” interfaces to “output” interfaces
- ProtoPktIP classes (IPv4 and IPv6) created for packet parsing, building, and manipulation (e.g. DPD Option detection/insertion).

Aside: ProtoPkt Classes

- “ProtoPkt” base class for basic C++ wrapper around a buffer (UINT32 aligned)
- ProtoPktETH provided for Ethernet frame parsing/building.
- ProtoPktIP, ProtoPktIPv4, and ProtoPktIPv6 classes for IP packet manipulation.
  - Checksums updated, etc as fields changed
  - Methods for iterating and adding extension headers
- A ProtoPktUDP class is also provided.
  - UDP checksum calculation/validation methods
- ProtoPktESP, ProtoPktAUTH, and ProtoPktDPD classes are provided to set/get fields as needed for SMF DPD.
- ProtoPktRTP is also provided (used in Ivox VoIP app)
- The “ManetMsg” classes (PacketBB) are based on ProtoPkt
- The goal was to provide a consistent, easy-to-use, and efficient (high-performance) mechanism for message/packet parsing/building
  - Other protocol messages (MGEN, NORM) could be based upon ProtoPkt class
  - Abstractions of ns-2 and OPNET packet structures could be created with alternate implementation of ProtoPkt classes …

nrlsmf Functions

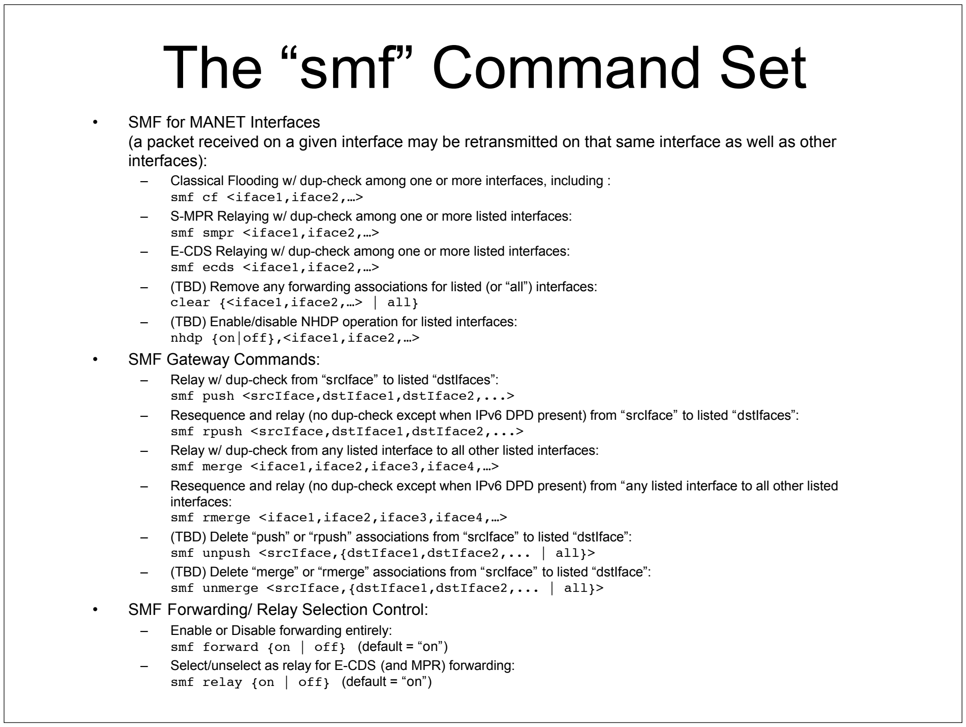
- MANET interface support
  - Duplicate packet detection
  - Supports S-MPR/ECDS/Classical forwarding
  - Multi-interface support
- Gateway support
  - Forced relaying of multicast packets across and among multiple interfaces.
  - Resequencing or packet marking for external flows injected into MANET/SMF areas
- “Remote control” interface allows external processes to control *nrlsmf* forwarding.
- Packet marking and resequencing for source hosts.

SMF-DPD Header Option

- *nrlsmf* resequencing for IPv6 (source and gateway) now uses the format with the optional "taggerID" as described in the current SMF draft.
- *nrlsmf* will correctly process packets received with the "taggerID", but does not <u>yet</u> provide an option to set the "taggerID" as a gateway.
- DPD for packets with "taggerID" is conducted in the context of `<srcAddr::dstAddr::taggerId>` sequence spaces.
  - If a flow is redundantly injected by gateways, it will be redundantly forwarded.
  - Other policies may be explored in the future.
- Note: Gateways will not "tag" flows that are pre-sequenced by sources (SMF-DPD or IPSec)

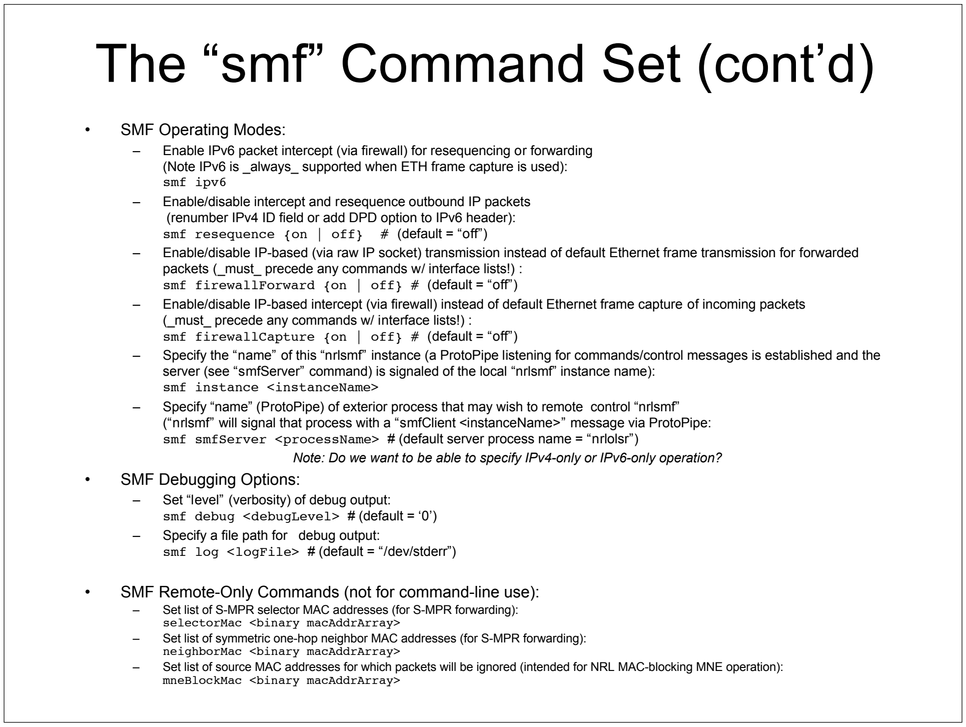
IPSec Duplicate Packet Detection

- *nrlsmf* now detects IPSec treated packets and uses IPSec sequence information for DPD on a `<srcAddr::dstAddr::SPI>` basis.
- IPSec packet flows are <u>not</u> resequenced by *nrlsmf.*
- IPv4 and IPv6, ESP and AH IPSec is supported.



The “smf” Command Set

- SMF for MANET Interfaces
  (a packet received on a given interface may be retransmitted on that same interface as well as other interfaces):
  - Classical Flooding w/ dup-check among one or more interfaces, including :
    `smf cf <iface1,iface2,…>`
  - S-MPR Relaying w/ dup-check among one or more listed interfaces:
    `smf smpr <iface1,iface2,…>`
  - E-CDS Relaying w/ dup-check among one or more listed interfaces:
    `smf ecds <iface1,iface2,…>`
  - (TBD) Remove any forwarding associations for listed (or “all”) interfaces:
    `clear {<iface1,iface2,…> | all}`
  - (TBD) Enable/disable NHDP operation for listed interfaces:
    `nhdp {on|off},<iface1,iface2,…>`
- SMF Gateway Commands:
  - Relay w/ dup-check from “srcIface” to listed “dstIfaces”:
    `smf push <srcIface,dstIface1,dstIface2,...>`
  - Resequence and relay (no dup-check except when IPv6 DPD present) from “srcIface” to listed “dstIfaces”:
    `smf rpush <srcIface,dstIface1,dstIface2,...>`
  - Relay w/ dup-check from any listed interface to all other listed interfaces:
    `smf merge <iface1,iface2,iface3,iface4,…>`
  - Resequence and relay (no dup-check except when IPv6 DPD present) from “any listed interface to all other listed interfaces:
    `smf rmerge <iface1,iface2,iface3,iface4,…>`
  - (TBD) Delete “push” or “rpush” associations from “srcIface” to listed “dstIface”:
    `smf unpush <srcIface,{dstIface1,dstIface2,... | all}>`
  - (TBD) Delete “merge” or “rmerge” associations from “srcIface” to listed “dstIface”:
    `smf unmerge <srcIface,{dstIface1,dstIface2,... | all}>`
- SMF Forwarding/ Relay Selection Control:
  - Enable or Disable forwarding entirely:
    `smf forward {on | off}` (default = “on”)
  - Select/unselect as relay for E-CDS (and MPR) forwarding:
    `smf relay {on | off}` (default = “on”)



The “smf” Command Set (cont’d)

- SMF Operating Modes:
  - Enable IPv6 packet intercept (via firewall) for resequencing or forwarding
    (Note IPv6 is _always_ supported when ETH frame capture is used):
    `smf ipv6`
  - Enable/disable intercept and resequence outbound IP packets
    (renumber IPv4 ID field or add DPD option to IPv6 header):
    `smf resequence {on | off}` # (default = “off”)
  - Enable/disable IP-based (via raw IP socket) transmission instead of default Ethernet frame transmission for forwarded packets (_must_ precede any commands w/ interface lists!) :
    `smf firewallForward {on | off}` # (default = “off”)
  - Enable/disable IP-based intercept (via firewall) instead of default Ethernet frame capture of incoming packets (_must_ precede any commands w/ interface lists!) :
    `smf firewallCapture {on | off}` # (default = “off”)
  - Specify the “name” of this “nrlsmf” instance (a ProtoPipe listening for commands/control messages is established and the server (see “smfServer” command) is signaled of the local “nrlsmf” instance name):
    `smf instance <instanceName>`
  - Specify “name” (ProtoPipe) of exterior process that may wish to remote control “nrlsmf”
    (“nrlsmf” will signal that process with a “smfClient <instanceName>” message via ProtoPipe:
    `smf smfServer <processName>` # (default server process name = “nrlolsr”)

Note: Do we want to be able to specify IPv4-only or IPv6-only operation?

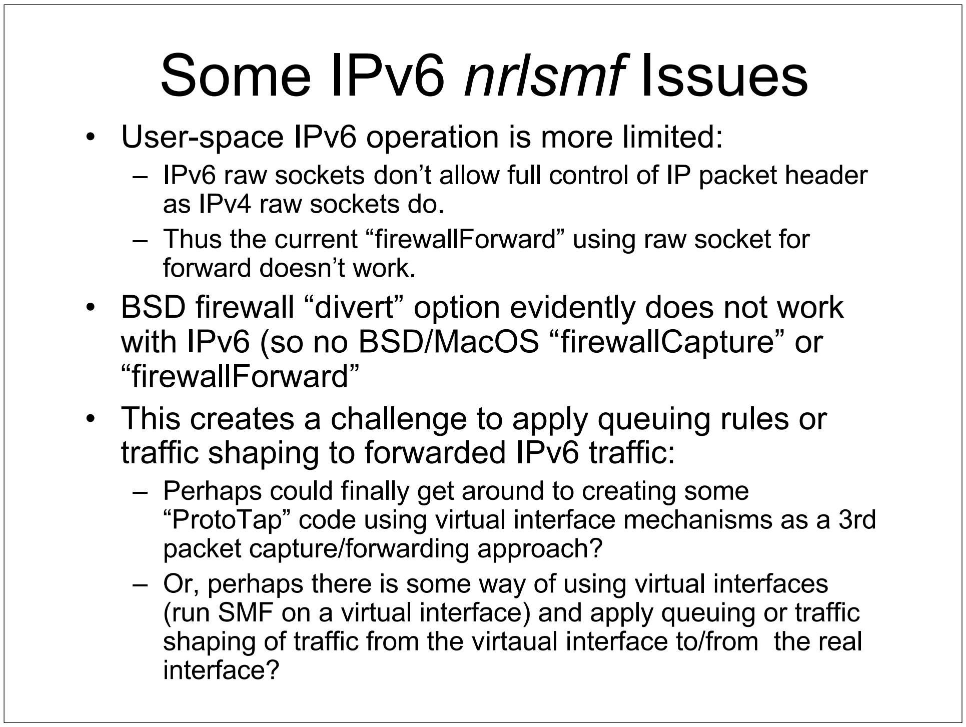
- SMF Debugging Options:
  - Set “level” (verbosity) of debug output:
    `smf debug <debugLevel>` # (default = ‘0’)
  - Specify a file path for debug output:
    `smf log <logFile>` # (default = “/dev/stderr”)
- SMF Remote-Only Commands (not for command-line use):
  - Set list of S-MPR selector MAC addresses (for S-MPR forwarding):
    `selectorMac <binary macAddrArray>`
  - Set list of symmetric one-hop neighbor MAC addresses (for S-MPR forwarding):
    `neighborMac <binary macAddrArray>`
  - Set list of source MAC addresses for which packets will be ignored (intended for NRL MAC-blocking MNE operation):
    `mneBlockMac <binary macAddrArray>`

Some Usage Notes

- IMPORTANT: The "rpush" and `"rmerge"` commands must be used very carefully when when used in combination with the "`firewallCapture on`" option:
  - The "firewallCapture" option doesn't get the *srcMacAddr* properly:
    - On Linux, locally generated packets have some random *srcMacAddr* from the 'ip_queue' capture mechanism (thus can’t detect it is receiving packets it sent and the resequencing bypasses DPD and a packet cyclone to TTL=0 results)
    - On BSD/MacOS, ProtoDetour doesn’t get the *srcMacAddr* at all for the "firewallCapture" mode (“layer 2” firewall rule would be needed).
  - So only use "`firewallCapture on`" when absolutely necessary.
- Also note that raw Ethernet forwarding is _not_ yet supported with "`firewallCapture  on`" (i.e.,"`firewallForward on`" MUST be used w/ "`firewallCapture on`”)
- 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.
- 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).

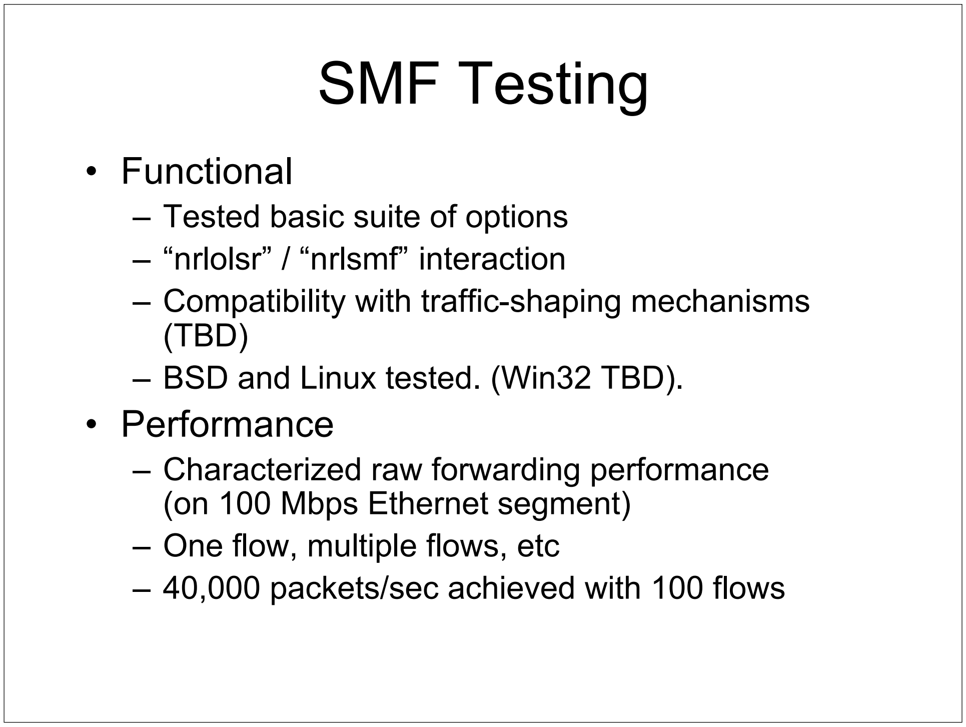
Future Additional Options

- Control of SMF DPD window parameters and prune timeout, perhaps on a per-interface basis.
- Similarly, there will likely be NHDP parameters (e.g. “HELLO” interval, etc) that may be useful to control on a per interface basis.
- Option to load a “config” file for complex configurations.
- 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)



Some IPv6 *nrlsmf* Issues

- User-space IPv6 operation is more limited:
  - 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 does not work with IPv6 (so no BSD/MacOS “firewallCapture” or “firewallForward”
- This creates a challenge to apply queuing rules or traffic shaping to forwarded IPv6 traffic:
  - Perhaps could finally get around to creating some “ProtoTap” code using virtual interface mechanisms as a 3rd packet capture/forwarding approach?
  - Or, perhaps there is some way of using virtual interfaces (run SMF on a virtual interface) and apply queuing or traffic shaping of traffic from the virtaual interface to/from the real interface?



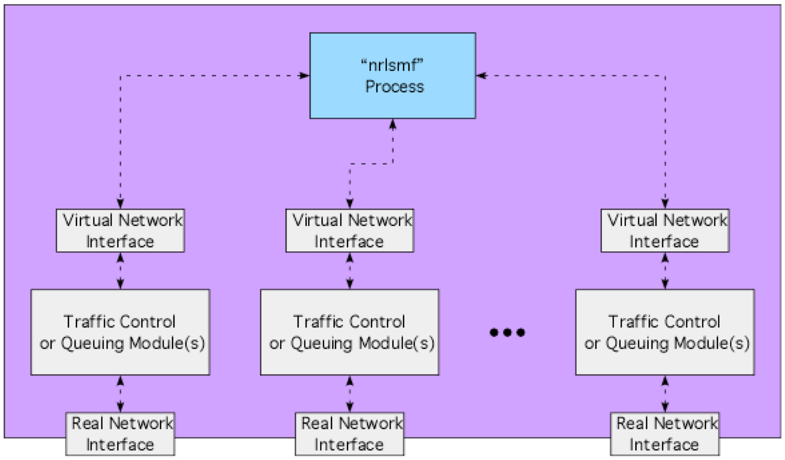
SMF Testing

- Functional
  - Tested basic suite of options
  - “nrlolsr” / “nrlsmf” interaction
  - Compatibility with traffic-shaping mechanisms (TBD)
  - BSD and Linux tested. (Win32 TBD).
- Performance
  - Characterized raw forwarding performance (on 100 Mbps Ethernet segment)
  - One flow, multiple flows, etc
  - 40,000 packets/sec achieved with 100 flows

Future Work Items

- More testing including field tests.
- Incorporate NHDP into “nrlsmf” code base (in progress)
- Explore use of virtual interface mechanisms (e.g., TAPx) for alternative packet capture and forwarding.
- Explore use of virtual interfaces to enable traffic shaping of forwarded IPv6 traffic

Possible Virtual Interface Hack?


“nrlsmf”
Process
Virtual Network Interface
Virtual Network Interface
Virtual Network Interface
Traffic Control or Queuing Module(s)
Traffic Control or Queuing Module(s)
•••
Traffic Control or Queuing Module(s)
Real Network Interface
Real Network Interface
Real Network Interface