kline-mif-mpvd-api-reqs-00 (and more)

ietf://94/mif

A personal request

Brace yourself, and bear with me, please, because:

- some of this is in the -00 doc
  - which was only sent out Monday morning (sorry)
- some of this comes out of a mif-homenet design team meeting
  - which only just met on Wednesday

Outline

1. Motivation
2. High-level API requirements
3. Demands placed upon the host operating system
4. New functionality required
5. Updates to existing APIs
6. Some “conceptual” PvDs
7. Specified PvD determination
8. Next steps and fruit-throwing

Motivation

- RFC7556#section-6 defines 3 level of app PvD-awareness
  - Basic
    - app is unchanged
    - system controls which PvD it gets to use
  - Intermediate
    - app minimally updated to specify connectivity requirements and/or preferences
    - system selects the PvD to use based on these, rest of app is mostly unchanged
  - Advanced
    - app is updated and able to use all “the firepower of this fully armed and operational battle station”
    - this work is (attempting to be) here
- Claim: basic and intermediate SHOULD be implementable given advanced

High-level requirements for an API

Using the API, an application MUST be able to:

- find accessible PvDs
- get configuration elements of a specific PvD
- restrict networking functionality to a specific PvD
  - ideally: on a per functional element basis
- specify only one PvD (per functional element)
  - iterating through multiple PvDs should be done explicitly...
  - ...at another layer (e.g. in a library wrapping the API)
- use only one consistent programming representation for “a PvD”
  - for all calls, and for both explicit and implicit PvDs

“PvD-aware End System Model”

- Somewhere between Weak and Strong End System models
- Slight variation on RFC1122#section-3.3.4.2:
  1. A host MAY silently discard an incoming datagram whose destination address does not correspond to any PvD associated with the physical (or virtual) interface through which it is received.
  2. A host MUST restrict itself to sending (non-source-routed) IP datagrams only through the physical (or virtual) interfaces that correspond to the PvD associated with the IP source address of the datagrams.

“PvD-aware End System Model” (cont’d)

- source address selection
  - The candidate source addresses MUST be restricted to the set of unicast addresses associated with the specified PvD (modifies RFC6724#section-4)
  - Additionally, source address selection policies from PvDs other than the specified PvD MUST NOT be applied.
- route isolation
  - The set of routes consulted for any routing decision MUST be restricted to the routes associated with the specified PvD
- “passive PvD determination”
  - e.g. for sending TCP SYN-ACKs, and maybe RSTs and ICMP errors
  - need to persist this information and present it to apps

Requirements for new functionality

- PvD availability
  - MUST be a call to get the list of currently accessible PvDs
  - SHOULD be able to receive notifications of updates without polling
- PvD configuration elements
  - MUST be a call to get a configuration element, given a PvD reference
  - SHOULD be extensible
  - SHOULD be able to receive notification of updates without polling
- PvD explicit intent
  - MUST be able to specify the PvD to which functionality is to be restricted
  - SHOULD be able to do this at every degree of programming parallelism
- PvD access policy

Updates to existing APIs

- PvD explicit intent
  - SHOULD update every existing API call that invokes some networking functionality
  - kind of obvious to say, but up to implementations to figure out what makes sense
- A special note about DNS
  - (because RFC3493, “Basic Socket Interface Extensions for IPv6”)
  - All DNS protocol communications with a PvD's nameservers MUST be restricted to use only source addresses and routes associated with the PvD.
  - “If getaddrinfo() is called with the AI_ADDRCONFIG flag specified, IPv4 addresses shall be returned only if an IPv4 address is configured within the specified provisioning domain and IPv6 addresses shall be returned only if an IPv6 address is configured within the specified provision domain. The loopback address is (still) not considered for this case as valid as a configured address.”

Some “conceptual” PvDs

- default PvD
  - “use this PvD unless another is explicitly specified”
  - system as a whole has a default PvD
  - there MUST be a way for app to set this for its entire process
  - system + per-process default PvD capability helps implement “basic PvD-awareness” level
- unspecified PvD
  - generally defers determination of a specific PvD to a resolution algorithm (coming up)
  - can be used with API calls to clear a previously specified PvD
- null PvD
  - permanently devoid of all configuration information
  - (system default PvD == null PvD) → (system policy == “all apps MUST be PvD-aware”)

Determination of a specific PvD

- Context 1: an app makes an API call that involves “networking functionality”
  - could be at a fairly high level or much more sophisticated
    - `getaddrinfo()`, `getnameinfo()`
    - `bind()`, `setsockopt()`, `sendmsg()`, `write()`
  - in these cases we can require that it be possible to identify the specific PvD of intent
  - “**active** PvD determination”
- Context 2: host operating system acting purely on behalf of an application
  - e.g. responding to an incoming packet with an ICMP error, fielding a TCP SYN, or receiving a UDP packet destined to an unconnected listening socket
  - in these cases we require that the host operating system do its best to identify the intended PvD based solely on whatever data is available
  - “**passive** PvD determination”

“Active” PvD determination

- Context 1: an app makes an API call that involves “networking functionality”
  1. was a PvD specified with the function call? => use this
  2. did the function call pass in:
     source address(es) or
     source address(es) and interface(s) or
     ...
     AND does this uniquely identify a PvD => use this
  3. (per thread default PvD specified?) => use this
  4. per process default PvD specified? => use this
  5. system default PvD specified? => use this
  6. else: use the null PvD

“Passive” PvD determination

- Context 2: host operating system acting purely on behalf of an application
  1. is this part of an ongoing communication for which the PvD is known? => use this
  2. does the:
     (destination) address or
     (destination) address + (incoming) interface or
     destination address + incoming interface + source address or
     ...
     uniquely identify a PvD? => use this
  3. else: TBD
     - “go home network, you’re drunk”
     - use the null PvD (i.e. drop)?
     - or the classic “undefined behaviour”: log an error and guess?

TBDs (and fruit-throwing)

- Does any of this make sense and/or seem reasonable?
- What about the claim that a conformant API suffices to build the APIs needed for basic and intermediate levels of PvD-awareness?
- What should we say about “passive PvD identification” failures?
- ...