Network Working Group
Internet-Draft
Intended status: Informational
Expires: Sep. 1, 2012

D. Liu
China Mobile
Ted. Lemon
Nominum
Yuri. Ismailov
Ericsson
Z. Cao
China Mobile
March 1, 2012

# MIF API consideration
# draft-ietf-mif-api-extension-00

Abstract

This document describes an abstract API that provides the minimal functionality required for a program to communicate effectively with peers and services on the network while running on a host that has more than one active network interface. This API is abstract: we describe the functionality that must be provided, not the bindings that should be used to provide that functionality. The functionality described here provides the building blocks from which higher-level APIs might be built, and is not intended to be used directly by typical applications.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 3, 2012.

Table of Contents

## 1. Introduction

Traditionally, hosts that communicate on the network have done so over a single network link, which is provided by a single service provider. This simple environment is relatively easy to program to, and relatively predictable.

However, this relatively simple case is no longer the norm. A typical modern host may have one or two wireless interfaces: a wireless interface connected to a broadband network, and possibly another connected to some kind of cellular network. The same host may also have a wired interface which is sometimes connected to another broadband link. It is also quite common for hosts to have VPN links that are configured, for example, for access to corporate networks, or for access to network privacy services.

As a result, it is now quite typical that a program attempting to communicate in such an environment will be presented with conflicting configuration information from more than one provider. In addition, the cost of bandwidth on different links and the power required ny those links may require consideration.

The API specified in this document is intended to describe the minimal complete set of API calls required to implement higher level APIs that solve these problems. It is not expected that applications will be implemented to this API, although it should be possible to do so. Rather, we expect this API to be used as a basis for building higher-level APIs that provide domain-specific solutions to these problems. The reason for specifying a lower-level API is to enable any arbitrary domain- specific API to be implemented, since no single higher-level API is likely to satisfy the needs of every application.

The API specified here is an abstract API. This means that we specify the functionality that is required to implement the API, but we do not provide specific bindings for any programming language: these are left up to the implementation. The API is described in terms of messages sent and messages received, rather than in terms of procedure calls, because it is necessary to be able to interleave these messages; a procedure call API necessarily precludes interleaving.

## 2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL","SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 3. MIF API Concept

The MIF API is intended to deal with situations where more than one interface may be active at a time. It must also deal with situations where a single interface is connected to a link that provides more than one type of network service. The most common example of this that we expect is a dual-stack network configuration.

### 3.1. Provisioning Domains

To properly handle these multiple-service interfaces, we specify the API not in terms of interfaces, but in terms of provisioning domains. So in the case of a dual-stack network attached to a single network interface, there would be two provisioning domains. If the host has a second interface that is connected to a link that only supports IPv6 service, then that host would be connected to a total of two network links, but three provisioning domains.

From the perspective of the MIF API, a provisioning domain consists of a link, plus all the configuration information received on that link for that provisioning domain. So for an IPv4 provisioning domain, that would be whatever information is received from the DHCP server. For an IPv6 provisioning domain, the information received through router advertisements would be combined with the information recieved via DHCPv6.

**point of discussion: it's actually possible to have two separate provisioning domains for IPv6 on the same wire. Is this a case that could happen in practice, and that we ought to support? I know that some asian countries have arrangements where the operator of the physical network is distinct from one or more operators who provide transit; I think this is all handled transparently to the host, but I don't really know the details.

**point of discussion: is IPv4 stateless/Bonjour a separate provisioning domain? What about IPv6 ULA?

### 3.2. Provisioning Domain Agnosticism

Although it is possible that a high-level API built on top of this API may be able to distinguish between provisioning domains, at the level of this API, no such distinction can be made. Each provisioning domain is treated separately, and it is the responsibility of the higher-level API or of the application to decide which provisioning domain or domains to actually use.

### 3.3. MIF API Elements

There are a number of different, essentially independent, pieces of software that need to be connected together in order to fully support a successful MIF communication strategy. These elements are shown in figure 3.1.

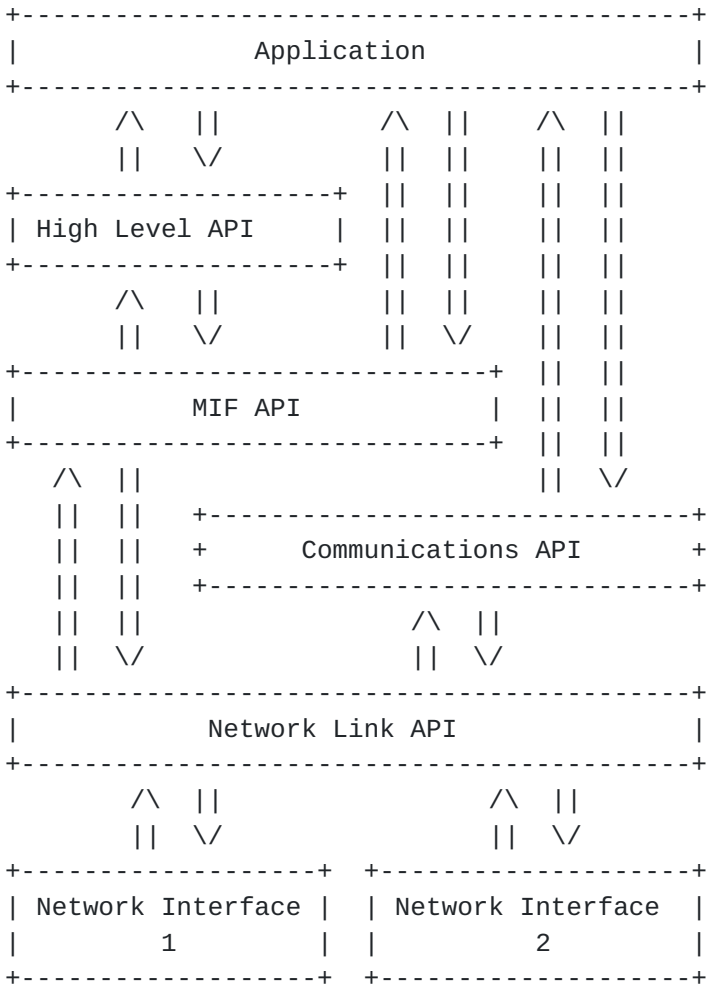


Figure 1

#### 3.3.1. Application Element

This is an actual application. Applications fall into a variety of broad categories, including network servers, web browsers, peer-to-peer programs, and so on. Although we are focusing here on the mechanisms required to allow these applications to originate connections to remote nodes, it is worth noting that applications must also be able to receive connections from remote nodes.

### 3.3.2. High Level API

Applications are generally expected to originate connections using some general-purpose high-level API suited to their particular function. It is likely that different applications may use different high-level APIs to communicate, depending on their particular needs. We do not describe the functioning of such high-level APIs; however, one such API under current consideration is the Happy Eyeballs for MIF [reference]. These APIs are expected to be able to be implemented using functionality like that described in the MIF API.

### 3.3.3. MIF API

This is the API being described in this document. Generally speaking, this API is used by higher-level APIs. However, it is permissible for applications to use the MIF API when it is deemed necessary. Currently, several modern web browsers take this approach to establishing network connections, rather than relying on vendor-provided connection mechanisms.

### 3.3.4. Communications API

Once an application has originated a connection with a remote node using either a high-level API or the MIF API, it must communicate. Similarly, when an application receives a connection from a remote node, it must communicate with that remote node. The communications API is used for this communication. Popular examples of such APIs include the POSIX socket API and a variety of other related APIs.

It is likely that in some instances, implementations of the MIF API will be done as extensions to the Communications API provided by a particular operating system; the functional separation we show here is intended to allow us to illustrate only those features required in a MIF environment, while relying on existing communications APIs to provide the rest.

### 3.3.5. Network Link API

This is the software that is responsible for actually managing whatever network links are present on a node, whether these are physical links or tunnels. What precisely this functional box contains may vary greatly from device to device. On a typical modern computer workstation, this functionality would almost certainly reside entirely in the system kernel; however, on an embedded device everything from the Application down to the Network Link API could easily be running together on the bare metal as a single program.

The Network Link API can completely concealed from the Application,

so we don't show a connection between them on the functional diagram, and indeed we do not talk about the functionality provided by this API. The reason for showing it on the functional diagram is simply to show that there likely is an API in common between MIF and the Communications API.

### 3.4. MIF API communication model

MIF API requests are made in the form of messages posted to the MIF API, and messages received from it. To accomplish this, several API calls are available. These calls mediate communication between the MIF API and the High Level API, or between the MIF API and the Application. In addition, the CHECK MESSAGE call allows the application to probe for or wait for messages from any of the APIs.

#### 3.4.1. POST MESSAGE call

This call causes a message to be posted to the MIF API. The call posts the message, and then returns.

#### 3.4.2. CHECK MESSAGE call

This call checks to see if there is a message waiting either from the High Level API, the MIF API, or the Communications API. Ideally it should be able to report the availability of any message or event that the application might anticipate receiving, so that the application can simply block waiting for such an event using this call. The application should be able to do a non-blocking probe, wait for some limited period of time, or wait indefinitely.

An example of a function of this type in existing practice is the POSIX poll() system call.

#### 3.4.3. GET MESSAGE call

This call checks to see if there is a message waiting. If there is no message, it returns a status code indicating that there is no message waiting. If there is a message, it returns the message.

### 3.5. MIF Messages

MIF messages always go in one direction or the other: from the subscriber to the MIF API, or to the subscriber from the MIF API. We use the term "subscriber" here to mean either the Application or the High Level API, since either is permitted to communicate with the MIF API.

Messages described here are grouped according to function.

### 3.5.1. Announce Interfaces

This message is sent to the MIF API to ask it to send a message announcing the existence of any interface. When the MIF API receives this message from a subscriber, it iterates across the list of all known interfaces; for each known interface, it sends an Interface Announcement message to the subscriber.

In addition, the MIF API sets a flag indicating that the subscriber is interested in learning about new interfaces. When the MIF API detects the presence of a new interface, it sends an Interface Announcement message for that interface to the subscriber. This would happen, for instance, when a new tunnel is configured, or when a USB device that is a network interface is discovered by the Network API.

Also, if a network interface goes away, either because the physical network device is disconnected, or because a tunnel is disabled, the MIF API will send a No Interface Announcement message to the subscriber.

### 3.5.2. Stop Announcing Interfaces

This message is sent to the MIF API when a subscriber is no longer interested in receiving announcements about new interfaces. Subsequently, the MIF API will no longer send Interface Announcement or No Interface Announcement messages to the subscriber.

### 3.5.3. Interface Announcement

This message announces the existence of an interface. The announcement includes an interface display name and interface identifier.

### 3.5.4. No Interface Announcement

This message announces that an interface that had been previously announced is no longer present. The announcement includes the interface identifier.

### 3.5.5. Announce Provisioning Domain

This message requests the MIF API to announce the availability of any provisioning domains configured on a particular interface. The interface identifier must be specified.

Upon receipt, the MIF API will iterate across the list of Provisioning Domains present for a particular interface, and will

send a Provisioning Domain Announcement for each such Provisioning Domain.

In addition, the MIF API will set a flag indicating that the subscriber wishes to know about new provisioning domains as they appear. Subsequently, when a new Provisioning Domain appears, the MIF API will send a Provisioning Domain Announcement message to the subscriber.

Finally, if a Provisioning Domain expires or is invalidated, the MIF API will send the subscriber a No Provisioning Domain Announcement message for that Provisioning Domain.

In the event that an interface on which provisioning domains has been announced goes away, a No Provisioning Domain Announcement message will be sent for each provisioning domain that had previously been announced on that interface before the No Interface Announcement message is sent.

Once a No Interface Announcement message has been sent, any subscriber that had subscribed to Provisioning Domain announcements for that interface will be automatically unsubscribed.

### 3.5.6. Stop Announcing Provisioning Domains

This message requests that the MIF API stop sending the subscriber Provisioning Domain Announcement and No Provisioning Domain Announcement messages. The subscriber must indicate the interface for which it no longer wishes to receive Provisioning Domain announcements.

### 3.5.7. Provisioning Domain Announcement

This message is sent by the MIF API to the subscriber to indicate that a new Provisioning Domain has successfully been configured on an interface. The announcement includes the interface identifier and the provisioning domain identifier.

### 3.5.8. No Provisioning Domain Announcement

This message is sent by the MIF API to the subscriber to indicate that an existing, previously announced provisioning domain has expired or otherwise become invalid, and can no longer be used.

### 3.5.9. Announce Configuration Element

This message is sent by the subscriber to request a specific configuration element from a specific provisioning domain. A

provisioning domain identifier must be specified.

The MIF API will respond by iterating across the complete list of configuration elements for a provisioning domain, sending a Configuration Element Announcement message to the subscriber for each one.

Additionally, if any Configuration Elements subsequently complete for a particular provisioning domain, the MIF API will send a Configuration Element Announcement message to the subscriber for each such element. If a Configuration Element becomes invalidated after it has been announced, the MIF API will send a No Configuration Element message.

If a provisioning domain expires or becomes invalid, the MIF API will iterate across the list of remaining configuration elements for that provisioning domain amd send a No Configuration Element Announcement message for each such configuration element.

### 3.5.10. Configuration Element Announcement

The Configuration Element Announcement message includes a Provisioning Domain ID and a Configuration Element Type, which can be one of the following:

- Config Element RA
- Config Element DHCPv6
- Config Element DHCPv4
- ...TBD...

### 3.5.11. No Configuration Element Announcement

The No Configuration Element Announcement message indicates that a previously valid configuration element for a provisioning domain is no longer valid. The message includes a provisioning domain identifier and a configuration element type.

### 3.5.12. Announce Address

This message is sent by the subscriber to request announcements of valid IP addresses for a specific provisioning domain. A provisioning domain identifier must be specified.

The MIF API will respond by iterating across the complete list of configuration elements for a provisioning domain, sending a Address Announcement message to the subscriber.

Additionally, if any new Address is subsequently configured on a particular provisioning domain, the MIF API will send an Address

Announcement message to the subscriber for each such element. If an address becomes invalidated after it has been announced, the MIF API will send a No Address Announcement message.

If a provisioning domain expires or becomes invalid, the MIF API will iterate across the list of remaining configuration elements for that provisioning domain amd send a No Address Announcement message for each such address.

### 3.5.13. Address Announcement

The Address Announcement message includes single IPv4 or IPV6 address and a Provisioning Domain identifier, as well as the valid and preferred lifetimes for that IP address (IPv6 only).

### 3.5.14. No Address Announcement

The No Address Announcement message indicates that a previously valid address for a provisioning domain is no longer valid. The message includes a provisioning domain identifier and an IPv4 or IPv6 address.

### 3.5.15. Get Configuration Data

The Get Configuration Data message is sent to the MIF API, and includes a Provisioning Domain ID, a Configuration Element Type, and a Configuration Information Identifier.

Configuration Information Identifiers:
   DNS Server List
   ...TBD...

The MIF API searches the configuration database for the specific type of Configuration Element on the specified Provisioning Domain to see if there is any configuration data of the specified type. If so, the MIF API sends a Configuration Data message to the subscriber; otherwise it sends a No Configuration Data message to the subscriber.

### 3.5.16. Translate Name

The Translate Name message is sent to the MIF API. It includes a provisioning domain and a name, which is a UTF8 string naming a network node. The message also includes a Translation Identifier, which the subscriber must ensure is unique across all outstanding name service requests.

The MIF API begins a name resolution process. As results come in from the name resolution process, the MIF API sends Name Translation

messages to the subscriber for each such result.

Name resolution can be handled by one or more translations systems such as local host table lookup, Domain Name System, NIS, LLMNR, and is implementation-dependent. **need to think about this

### 3.5.17. Stop Translating Name

This message is sent to the MIF API to indicate that the subscriber is no longer interested in additional results from a particular name translation process. The message includes the Translation Identifier.

### 3.5.18. Name Translation

The MIF API sends a Name Translation message to subscribers whenever results come in from a name translation process being performed on behalf of the subscriber. The Name Translation message includes the Translation ID generated by the subscriber, and an IP address returned by the translation process. If a single translation result contains more than one IP address, or IP addresses of different types, the MIF API sends a single Name Translation message for each such IP address.

### 3.5.19. Connect to Address

The Connect to Address message contains an IP address, a provisioning domain identifier, and a connection identifier which the subscriber must ensure is unique. The MIF API attempts to initiate a TCP connection to the specified IP address using one or more source addresses that are valid for the specified provisioning domain, according to the source address selection policy for that provisioning domain.

If the connection subsequently succeeds, the MIF API will send a Connected message to the subscriber. If it subsequently fails, the MIF API will send a Not Connected message to the subscriber.

### 3.5.20. Connect to Address From Address

The Connect to Address From Address message contains a source IP address, a destination IP address, a provisioning domain identifier, and a connection identifier which the subscriber must ensure is unique. The MIF API attempts to initiate a TCP connection to the specified IP address using the specified source address.

If the connection subsequently succeeds, the MIF API will send a Connected message to the subscriber. If it subsequently fails, the

MIF API will send a Connection Failed message to the subscriber.

### 3.5.21. Connected

The Connected message contains the connection identifier that was provided in a previous Connect to Address or Connect to Address From Address message sent by the subscriber. It also contains an token, suitable for use with the connection API, for communicating with the end node to which the connection was established.

### 3.5.22. Not Connected

The Not Connected message contains the connection identifier that was provided in a previous Connect to Address or Connect to Address From Address message sent by the subscriber. It also contains an indication as to what went wrong with the connection.

## 4. Example Usage

below is an example that shows how MIF API in use:

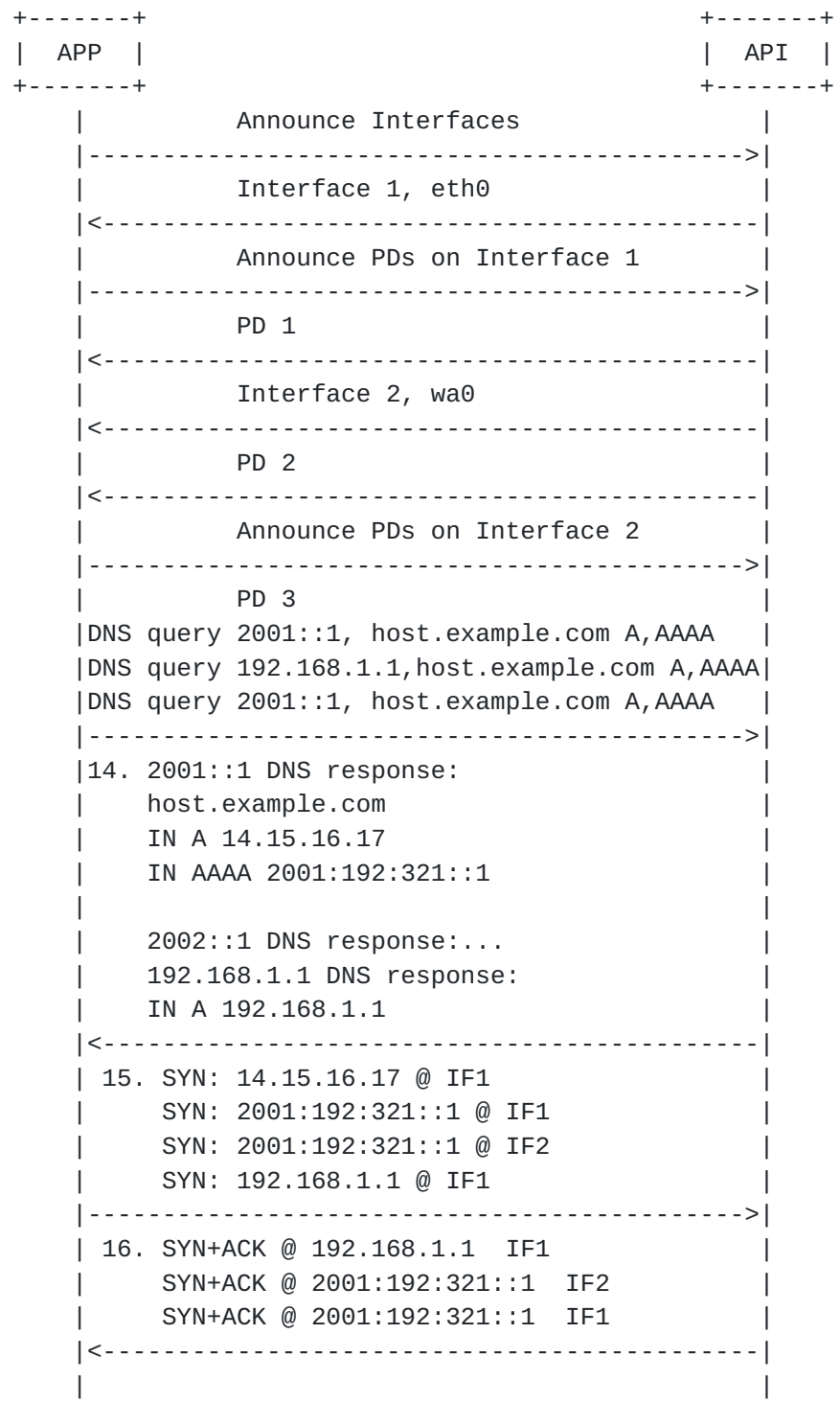


Figure 2

As described in the above communication model, the application first invoke the MIF API to query how many interfaces in the host. then, the application invokes MIF API to query how many networks attaches in each interface. application then invoke MIF API to query each DNS

configuration on each interface's attached network. application then send DNS query to each DNS server on each network. The DNS servers may return multiple IP address of the queried host name. The application then try to connect to each IP addresses of the host by sending tcp SYN packet to each destination IP addresses through multiple interfaces. Some of the destination IP address may return ACK packet some may not. The application then chose a best connection based on certain criteria. for example, the criteria may based on the qulity of the link.

## 5. Security Considerations

TBD

## 6. IANA Considerations

None

## 7. Acknowledgments

The authors want to thank Teemu Savolainen from Nokia, Dayi Zhao from Bitway, Dave Thaler from Microsoft and others for their useful suggestions and discussions.

## 8. References

### 8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 8.2. Informative References

[I-D.scharf-mptcp-api] Scharf, M. and A. Ford, "MPTCP Application Interface Considerations", draft-scharf-mptcp-api-02 (work in progress), July 2010.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

Authors' Addresses

Dapeng Liu
China Mobile
Unit2, 28 Xuanwumenxi Ave,Xuanwu District
Beijing 100053
China

Email: liudapeng@chinamobile.com

Ted Lemon
Nominum
Redwood City
CA 94063
USA

Email: Ted.Lemon@nominum.com

Yuri Ismailov
Ericsson
Stockholm
Sweden

Email: yuri@ismailov.eu

Zhen Cao
China Mobile
Unit2, 28 Xuanwumenxi Ave,Xuanwu District
Beijing 100053
China

Email: caozhen@chinamobile.com