

Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 3, 2011

B. Carpenter
Univ. of Auckland
S. Jiang
Huawei Technologies Co., Ltd
B. Zhou
ChinaMobile
August 30, 2010

Problem Statement for Referral
draft-carpenter-referral-ps-01

Abstract

The purpose of a referral is to enable a given entity in a multiparty Internet application to pass information to another party. It enables a communication initiator to be aware of relevant information of its destination entity before launching the communication. This memo discusses the problems involved in referral scenarios.

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 March 3, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	3
3. Goals of Referral	4
3.1. Reachability	4
3.2. Path Selection	4
3.2.1. An Example: Triangle Path Optimization	4
3.3. Interface Selection	5
4. Problem Statement	6
4.1. IP Addresses are not sufficient	6
4.2. FQDNs are not sufficient	7
4.3. Relevant Information is lack	8
4.4. Extra complexity from ID-Locator Split Mechanisms	9
5. A Generic Referral Mechanism is needed	9
6. Security Considerations	11
7. IANA Considerations	11
8. Acknowledgements	11
9. Change log	12
10. References	12
10.1. Normative References	12
10.2. Informative References	12
Authors' Addresses	13

1. Introduction

A frequently occurring situation is that one entity A connected to the Internet (or to some private network using the Internet protocol suite) needs to be aware of the information of another entity B in order to reach it. The information can be obtained from B itself or some third-party entity C. This is known as a referral.

Referral is the act whereby one entity informs another entity how to contact a specific entity. It enables a communication initiator to be aware of relevant information of its destination entity in order to launch a communication channel. This referral information can be obtained through an existing communication channel between these two entities or from third-party entities.

In the original design of the Internet, IP addresses were global, unique, and quasi-permanent. Also any differentiation beyond that provided by an IP address was done by protocol and port numbers. Referrals were therefore performed simply by passing an IP address and possibly protocol and port numbers. In fact simple referrals (the first case above, sometimes called first-party referrals) were never needed since A could simply use B's address. Third-party referrals were trivial: C would tell A about B's address. Thus, it became common practice to pass raw addresses between entities. A classical example is the FTP PORT command [RFC0959].

2. Terminology

This document makes use of the following terms:

- o "Entity": we use this rather than "application" to describe any software component embedded in an Internet host, not just a specific application, that sends, receives or makes use of referrals. Also, in case of dynamic load sharing or failover, an entity might even migrate between hosts.
- o "Referral": the act of one entity informing another entity how to contact a specific entity.
- o "Reference": the actual data (name, address, identifier, locator, pointer, etc.) that is the basis of a referral.
- o "Referring entity": the entity that sends a referral.
- o "Receiving entity": the entity that receives a referral.
- o "Referenced entity": the target entity of a reference.
- o "Scope": the region or regions of the Internet within which a given reference is applicable to reach the referenced entity.

3. Goals of Referral

The principal purpose of referral is to enable one entity in a multi-party application to pass information to another party involved in the same application. This document makes no assumptions about whether the entities are acting as clients, servers, peers, super-nodes, relays, proxies, etc., as far as the application is concerned. Neither does it take a position as to how the various entities become aware of the need to send a referral; this depends entirely on the structure of the application.

3.1. Reachability

The primary goals of referral is to enable a communication initiator to reach its destination entity. Referral is a best effort mechanism. It does not guarantee actual reachability, since the referring entity has no general way of knowing which paths exist between the receiving entity and the referenced entity. Even if a reference is theoretically in scope, and within its defined lifetime, it may have become unreachable since it was sent. A receiving entity should always be prepared for reachability failures and associated retry and failover mechanisms, which are out of scope for the referral mechanism itself.

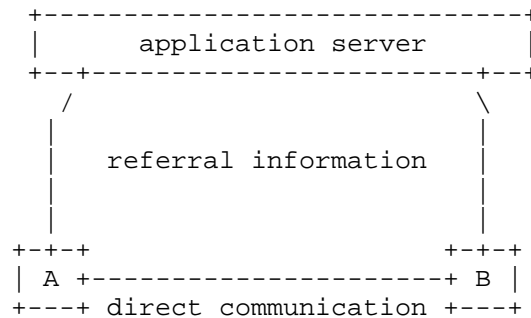
3.2. Path Selection

A reference might carry multiple references for the same target. These may lead to multiple possible paths from the receiving entity to the referenced entity. This scenario is particularly generic when the destination or/and source entity has multiple interfaces or is multi-homed.

The referring entity is not likely to know which path is best. The receiving entity will need to make a choice, possibly by local policy (e.g. [RFC3484]) or possibly by trial and error (e.g. [RFC4038], [RFC5534]). This choice is also out of scope for the referral mechanism itself.

3.2.1. An Example: Triangle Path Optimization

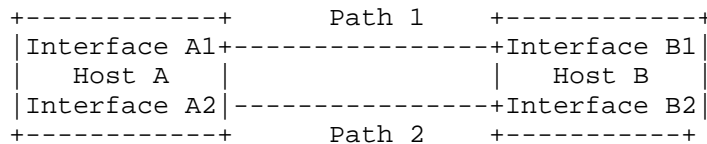
In application scenarios, the triangular path shown below is common. Both Host A and Host B connect to an application server and the application server forwards traffic as a relay agent. A slightly more complicated scenario is when the two hosts connect to different application servers individually and application servers talk to each other's relay agents. In SIP, this is often called the "SIP trapezoid".



By passing A's reference to B, B can try to communicate directly with A, using the communication line at the bottom. If the direct communication is established successfully, the triangle path gets optimized. Both the application server and network bandwidth can be benefit from this operation.

3.3. Interface Selection

We also encounter multi-interfaced hosts whose reachability is bound to a particular (logical/physical) interface. More information is required to indicate which interface may be used under different circumstances. Multi-interface is defined and studied by IETF MIF WG. Here referral can provide the host A's multi-interface information to host B, accordingly, host B can select one of the interface to establish the connection.



For example, as shown in the above figure, Host A has connected to Host B through Path 1. They can exchange references through Path 1. They may find out Path 2 using different interfaces is better than Path 1, maybe cheaper, faster or more stable. Then, they can switch to Path 2. Host A has interface A1 as broadband access, almost free; and interface A2 is 3G access, which costs 0.1 \$ per MB. Both of them are available for incoming connection. If these information is passed to host B, through referral, then host B should choose A1 interface to reach host A. This kind of information is useful to express the source's status or preference.

In order to choose between different interfaces, not only the connectivity information of these interfaces, but also some additional

information may be helpful, such as bandwidth, finance cost, latency, etc. These additional information may also be provided through referral. However, this additional information, even if known by the referring entity, may be invalid at the location of the receiving entity.

4. Problem Statement

Unfortunately, the simple approach to referrals, passing an IP address, often fails in today's Internet. As has been known for some time [RFC2101], hosts' IP addresses no longer all have global scope. They often have limited reachability, and may have limited lifetime. They are not sufficient to establish communication in many cases of dynamic referrals, for a variety of reasons. FQDN may be used instead in some scenarios. However, FQDN also has its own limitation and may fail in some scenarios.

4.1. IP Addresses are not sufficient

It is no longer reasonable to assume that a host with a fixed location has a fixed IP address, or even a stable IP address.

Furthermore, in the context of IPv4 address exhaustion, several solutions have emerged to share a single public IPv4 address between several customers simultaneously. Consequently, a public IPv4 address often no longer identifies a single customer/user/host while a private IPv4 address is meaningless out of the private network scope. Other information (e.g., port range) is required to identify unambiguously a given customer/user/host. Both IP addresses and port numbers may be different on either side of a NAT or some other middlebox [RFC3234], and firewalls may block them. It is no longer reasonable to assume that an IP address for a host, which allows a given peer to reach that host in one location, also works from a different location - even if that host is reachable from the second location.

Also, the Internet now has two co-existing address formats for IPv4 and IPv6. Direct communication can only be established when both peers use the same IP version. Having the address of the source and destination in the same IP version does not necessarily mean that the path will be using that IP version. Simple approaches may cause unnecessary double translation [I-D.boucadair-softwire-cgn-bypass]. Some addresses may even be the result of translation between IPv4 and IPv6, with severe limitations on their scope and lifetime. Sending an out-of-scope or expired address, or one of the wrong format, as a referral will fail.

IP addresses today may have an implied "context" (VPN, VoIP VC, IP TV, etc.): the reachability of such an address depends on that context.

An implication of these issues is that there is no clean definition of the scope of an address (especially an IPv4 address, due to the prevalence of NAT). It is impossible to determine algorithmically, by inspecting the bits of an address, what its scope of reachability is. Resolving this problem would greatly clarify the general problem of referrals.

4.2. FQDNs are not sufficient

In some cases, this problem may be readily solved by passing a Fully Qualified Domain Name (FQDN) instead of an IP address. Indeed, that is an architecturally preferred solution [RFC1958]. However, it is not sufficient in many cases of dynamic referrals. Experience shows that an application cannot use a domain name in order to reliably find usable address(es) of an arbitrary peer. Domain names work fairly well to find the addresses of public servers, as in web servers or SMTP servers, because operators of such servers take pains to make sure that their domain names work. But DNS records are not as reliably maintained for arbitrary hosts such as might need to be contacted in peer-to-peer applications, or for servers within corporate networks. Many small networks do not even maintain DNS entries for their hosts, and for some networks that do list local hosts in DNS, the listings may well be unusable from a remote location, because of two-faced DNS, or because the A record contains a private address. These cases may even be intentional as part of a security ring-fence, where w3.example.com only resolves within the corporate boundary, and/or resolves to IP addresses which are only reachable within the corporate administrative boundaries. In such contexts, incoming connections are usually filtered by the corporate firewall.

An additional issue with FQDNs is the very common situation where multiple hosts are hidden behind a NAT, but they share one FQDN which is in fact a dummy name, created automatically by the ISP so that reverse DNS lookup will succeed for the NAT's public IPv4 address. Such FQDNs are useless for identifying specific hosts.

Furthermore, an FQDN may not be sufficient to establish successful communications involving heterogeneous peers (i.e., IPv4 and IPv6) since A and AAAA records may not be consistently provisioned. There are known cases where a server has one name that produces an A record (e.g., www.example.com) and another name that produces an AAAA record (e.g., ipv6.example.com). An additional complication is that some answers from DNS may be synthetic IP addresses, e.g., AAAA records

sent by DNS64. The host may have no means to detect that such an address represents an IPv4 host. These addresses should not be interpreted as native IPv6 address.

In such cases, an IP address either cannot be derived from an FQDN, or if so derived, cannot be accessed from an arbitrary location in the Internet.

A related problem is that an application does not have a reliable way of knowing its own domain name - or to be more precise, a way of knowing a domain name that will allow the application to be reached from another location.

There are wider systemic problems with the DNS as a reliable way to find a usable address, which are somewhat out of scope here, but can be summarised:

- o In large networks, it is now quite common that the DNS administrator is out of touch with the applications user or administrator, and as a result, that the DNS is out of sync with reality.
- o DNS was never designed to accommodate mobile or roaming hosts, whose locator may change rapidly.
- o DNS has never been satisfactorily adapted to isolated, transiently-connected, or ad hoc networks.
- o It is no longer reasonable to assume that all addresses associated with a DNS name are bound to a single host. One result is that the DNS name might suffice for an initial connection, but a specific address is needed to rebind to the same peer, say, to recover from a broken connection.
- o It is no longer reasonable to assume that a DNS query will return all usable addresses for a host.
- o Hosts may be identified by a different URI per service: no unique URI scheme, meaning no single FQDN, will apply.

For all the above reasons, the problem of address referrals cannot be solved simply by recommending the use of FQDNs instead. The guideline in [RFC1958] is in fact too simple for today's network. Something more elaborate than an IP address or an FQDN appears to be needed in the general case of application referrals.

4.3. Relevant Information is lack

Neither an IP address nor an FQDN gives complete information about the referenced entity. For example, IP addresses normally have associated lifetimes (derived from DHCP, SLAAC or the relevant DNS TTL), so they should be treated as invalid after their lifetimes expire. A referral that does not convey the lifetime associated with an address is problematic. As mentioned above, the scope of a

reference also affects its usefulness. These are examples of additional information that is necessary to correctly interpret a referral; therefore part of the problem is conveying such information along with the reference.

4.4. Extra complexity from ID-Locator Split Mechanisms

Additional complexity for referrals would come from the deployment of any technology that separates locators from identifiers, rather than combining the two as an IP address. Since a very wide range of such solutions have been proposed (e.g. HIP, LISP, ILNP and Name-based Sockets) [I-D.ubillos-name-based-sockets], it is difficult to define the resulting problems precisely.

However, to consider the example of Name-based Sockets, if a referral was made based on the IP address being used at a given instant for a Name-based Socket, that address might be useless by the time the referral was completed, because the socket suddenly migrated to a different IP address.

The SHIM6 protocol [RFC5533] and the Multiple Interfaces (mif) Working Group may produce similar difficulties, since they also consider scenarios where the IP address in use for some purpose may change unexpectedly.

Any referral mechanism must be able to deal with situations where the locator corresponding to a given identifier is subject to change.

5. A Generic Referral Mechanism is needed

The first motivation is the observation that unless the parties involved have reached an understanding about the scope, lifetime, and format of the elements in a referral through some other means, that information must be passed with the referral. This is required so that the receiving entity can determine whether or not the referral is useful. The referral therefore needs to consist of a fully-fledged data structure, or to be made using a mutually agreed referral protocol.

When an attempt to establish a communication channel based on certain referral information fails, good design suggests that the receiving entity should attempt to correct the situation. For example, if communication fails to be established using an IP address, it would often be appropriate to attempt a DNS lookup, despite the difficulties mentioned above. The second motivating problem is that it may be helpful to the entity receiving a reference to also receive information about the source of the reference, such as an FQDN, if

that is known to the sender of the reference. The receiving entity can then attempt to recover a valid address (and possibly port number) for the referred entity.

The third motivating problem is to allow a reference to contain alternatives to an IP address or an FQDN, when any such alternatives exist.

Additional arguments for a generic referral mechanism include:

1. Allow for general mechanisms that can be used by any application to handle references and understand the meaning of referral information, such as IP address, possibly protocol and port numbers. However, there is an open question whether this standard referral design should be used for new applications only, or extended to existing applications.
2. Simplify ALG design during middlebox traversal. There are middleboxes, like firewalls and translators, especially in the mobile network, which require application layer gateways ALG. The cost of ALG functions is huge for the mobile operator in terms of implementation, performance. Standard references could simplify ALG implementation during middlebox traversal in the mobile network.
3. Simplify packet inspection. Operators sometimes need to inspect information or details during communication for administration reasons. If referral mechanism is standardized, it is easier for an operator to capture and investigate the required information.

We observe that we have identified two general requirements: the need to define address scope more precisely, and the need to communicate references in a generic way.

It should be noted that partial or application-specific solutions to these problems abound, because any multi-party distributed application must solve them. The best documented example is ICE [RFC5245], which is an active protocol specific to applications mediated by SDP [RFC4566]. ICE "works by including a multiplicity of IP addresses and ports in SDP offers and answers, which are then tested for connectivity by peer-to-peer connectivity checks." The question raised here is whether we can define requirements for a generic solution that can be used by future applications, and possibly be retro-fitted to existing applications.

One approach could be a "SuperICE" designed to be completely general and not tied to the SDP model. Another approach is the idea of a generic referral object. Such an object could be passed between the entities of a multi-party application, but without defining a specific protocol for that purpose. Some applications might choose to send it in-band as a raw binary object, others might use a simple

ASCII encoding, and still others might prefer to encode it in XML, for example. Finally, it might also be used as part of SuperICE.

6. Security Considerations

It should be noted that referral should not function as a way to nullify the effect of a firewall or any other security mechanism. If the receiving entity chooses a particular reference and attempts to send packets to the corresponding IP address, whether they are delivered or not will depend on the existing security mechanisms, whatever they may be.

Nevertheless, if a site security policy requires it, certain references may be excluded from referral information sent to certain destinations. This would require a security policy mechanism to be added to the process of generating referral information.

Forged or intercepted referral information would enable a wide variety of attacks. Although not fundamentally different from attacks based on forged or observed IP addresses or FQDNs, no doubt referral would allow such attacks to be more ingenious, simply because they provide more information than an address or FQDN alone. Referral information should be transmitted through authenticated and encrypted channels. It is not further elaborated here.

Referral may raise potential privacy issues, which are not explored in this document. For example, in the SIP context, mechanisms such as [RFC3323] and [RFC5767] are available to hide information that might identify end-points. Referral usage scenarios must ensure that they do not unintentionally defeat privacy solutions.

7. IANA Considerations

This document requests no action by IANA.

8. Acknowledgements

Valuable comments and contributions were made by Mohamed Boucadair, Dan Wing, Keith Moore and others.

This document was produced using the xml2rfc tool [RFC2629].

9. Change log

draft-carpenter-referral-ps-00: original version, 2010-06-21.

draft-carpenter-referral-ps-01: add content regarding to ID-Locator Split Mechanisms, 2010-08-30.

10. References

10.1. Normative References

- [RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.
- [RFC3323] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.
- [RFC3484] Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, February 2003.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC5533] Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", RFC 5533, June 2009.
- [RFC5534] Arkko, J. and I. van Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming", RFC 5534, June 2009.

10.2. Informative References

- [I-D.boucadair-softwire-cgn-bypass]
Boucadair, M., "Procedure to bypass DS-lite AFTR (work in progress)", December 2009.
- [I-D.ubillos-name-based-sockets]
Ubillos, J., "Name Based Sockets (work in progress)", July 2010.
- [RFC1958] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996.

- [RFC2101] Carpenter, B., Crowcroft, J., and Y. Rekhter, "IPv4 Address Behaviour Today", RFC 2101, February 1997.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC2775] Carpenter, B., "Internet Transparency", RFC 2775, February 2000.
- [RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.
- [RFC4038] Shin, M-K., Hong, Y-G., Hagino, J., Savola, P., and E. Castro, "Application Aspects of IPv6 Transition", RFC 4038, March 2005.
- [RFC5767] Munakata, M., Schubert, S., and T. Ohba, "User-Agent-Driven Privacy Mechanism for SIP", RFC 5767, April 2010.

Authors' Addresses

Brian Carpenter
Department of Computer Science
University of Auckland
PB 92019
Auckland, 1142
New Zealand

Email: brian.e.carpenter@gmail.com

Sheng Jiang
Huawei Technologies Co., Ltd
Huawei Building, No.3 Xinxu Rd.,
Shang-Di Information Industry Base, Hai-Dian District, Beijing
P.R. China

Email: shengjiang@huawei.com

Bo Zhou
China Mobile
Unit2, 28 Xuanwumenxi Ave,Xuanwu District
Beijing, 100053
P.R. China

Email: zhouboyj@gmail.com

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: March 21, 2011

J. Ubillos
Swedish Institute of Computer
Science
M. Xu
Z. Ming
Tsinghua University
C. Vogt
Ericsson
September 17, 2010

Name-Based Sockets Architecture
draft-ubillos-name-based-sockets-03

Abstract

This memo defines Name-based sockets. name-based sockets allow application developers to refer to remote hosts (and it self) by name only, passing on all IP (locator) management to the operating system. Applications are thus relieved of re-implementing features such as multi-homing, mobility, NAT traversal, IPv6/IPv4 interoperability and address management in general. The operating system can in turn re-use the same solutions for a whole set of guest applications. name-based sockets aims to provide a whole set of features without adding new indirection layers, new delays or other dependences while maintaining transparent backwards compatibility.

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 March 21, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Conventions	3
2. Terminology	3
3. Overview	3
3.1. Introduction	3
3.2. Motivation	3
3.3. Compatibility goals	4
3.3.1. Network compatibility	4
3.3.2. Application compatibility	4
3.4. name-based sockets overview	4
3.5. Protocol overview	5
4. Initial name exchange	5
4.1. Name format	7
4.2. Service names (and port numbers)	7
4.3. Transport selection	7
5. API	7
6. Security Considerations	9
7. IANA Considerations	9
8. Contributors	9
9. References	9
9.1. Normative References	9
9.2. Informative References	9
9.3. URL References	9

1. Conventions

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].

2. Terminology

Locator - An IP address (v4 or v6) on which a host can be reached.

Name - A character string (max 255 chars long) on which a host can be identified.

3. Overview

3.1. Introduction

name-based sockets provide a unified interface which caters to the application developers who wish to simply open up a communication to a remote host by its name and have the operating system perform the management of locators.

It does so by providing a new address family (AF_NAME) which allows the developer to use a name instead of an IP (or other locator).

3.2. Motivation

Network communication has for very long been based on the assumption that applications should deal with the IP (locator) management. This is based on the legacy notion that an IP does not change during a session and that a session is a communication between two given IPs (locators). This situation has changed, locators change during a session, and a device/host might have multiple locators, hosts may be behind NA(P)Ts or be on different locator domains (e.g. IPv4/IPv6). Today, this is mostly left to the individual applications to solve. This adds complexity to the applications making it a challenge in itself just to meet the networking demands of applications.

Name-based sockets aims to fix this by pushing the locator management to the operating system, without introducing any new limitations or delays.

By approaching the problem at the socket() level, existing abstraction frameworks can easily benefit from a change. By allowing the existing abstractions to simply pass along the name the whole way down to the socket() call, the abstraction or whole framework are able to benefit by name-based sockets.

Note that name-based sockets do not aim to replace all `socket()` based communications. There are of course cases which are limited due to obvious boot-strapping problems. E.g. a DHCP client, or an DNS-querying client would do better in not using a name oriented architecture.

3.3. Compatibility goals

3.3.1. Network compatibility

1. Minimal changes to "on the wire" protocol.
2. No changes to added 'features'

3.3.2. Application compatibility

The objective is to maintain the (BSD) `socket()` semantics commonly used today. It is important that the transition for developers should be trivial.

Maintaining these semantics allows existing abstraction frameworks to integrate name-based sockets to their frameworks. Socket abstractions are often used to simplify the use of a service e.g. HTTP. In those cases, they do not add functionality at the network or transport layer. These kinds of abstraction frameworks would quickly be able to make use of the extra functionality provided by name-based sockets.

3.4. name-based sockets overview

name-based sockets provide a new socket interface. It is implemented using a new address family (`AF_NAME`). This means that the sockets are only used by applications who explicitly invoke it. The result is that applications that do use name based sockets are very aware of the set of features they are provided with, hence they know not to re-implement them. Current implementations are implemented as kernel modules, however a user-space library implementation ought to work just as well.

By using DNS as the ID/Locator mapping structure, we do not introduce any new indirections. Please note that the responding host does not need to do any reverse-DNS resolution (explained below). We part from the assumption that most application network calls start with a FQDN.

The exchange of names is performed in-band, by piggy-backing the needed information on the first couple of packets exchanged. The consequence is:

- o No extra delays
- o No extra features until the name exchange has ended.
- o As a result of this, we also achieve backwards compatibility (a name exchange which never completes)

3.5. Protocol overview

name-based sockets work by performing a name exchange in the beginning of a communication. It does this in a non-blocking way. In practice what happens is that the first few packets are exchanged in a normal legacy fashion. However, to these packets, extra information about the corresponding hosts names are piggy-backed. If a name exchange is successful, the extra features provided by name-based sockets are enabled. If the exchange does not succeed, normal legacy communication continues unaffected.

The name of each host is either its FQDN, its IP in IPv6.arpa format or an arbitrary nonce. It may or may not be authenticated. In the ordinary case, the name is not authenticated, thus the receiver does not need to perform a reverse or forward lookup, hence not adding any further delays to the first packet(s). The motivation for this is to avoid any additional "first-packet" delays.

Once the name exchange has been performed successfully the complete feature set will be made available to the communication automatically.

The expected API for a socket using AF_NAME is the same as for e.g. TCP (SOCK_STREAM). This is also the case for SOCK_DGRAM and similar protocols, for all practical purposes, the functionality remains unchanged, however, as state is created in both ends, connection oriented semantics are more intuitive.

4. Initial name exchange

When the sender sends the first packet to the receiver it appends its own name as an IP-Option/IPv6-Extension header. It repeats this for a predefined amount of time or packets. On the receiving end, if the receiver supports name-based sockets it appends its own name in the same fashion for a predefined amount of time or packets. Should the receiver not be able to interpret the name, the option/extension header is ignored and the legacy communication precedes as normal.

This kind of name exchange has two consequences. First and foremost that there are no extra delays on the initial packets. Secondly that the complete feature set provided by name based sockets will not be

available until a few packets have been exchanged.

In figure 1 the exchange is depicted. The first packet from the sender has its own name appended to it. The next few packets sent will also have the name appended to it for a predefined number of packets (X in the figure) or until a reply including a name extension is received. On the receiving end, should an incoming packet have a name extension, the receiver begins to append its own name to the sent packets. It does so for a predefined number of packets (X in the figure).

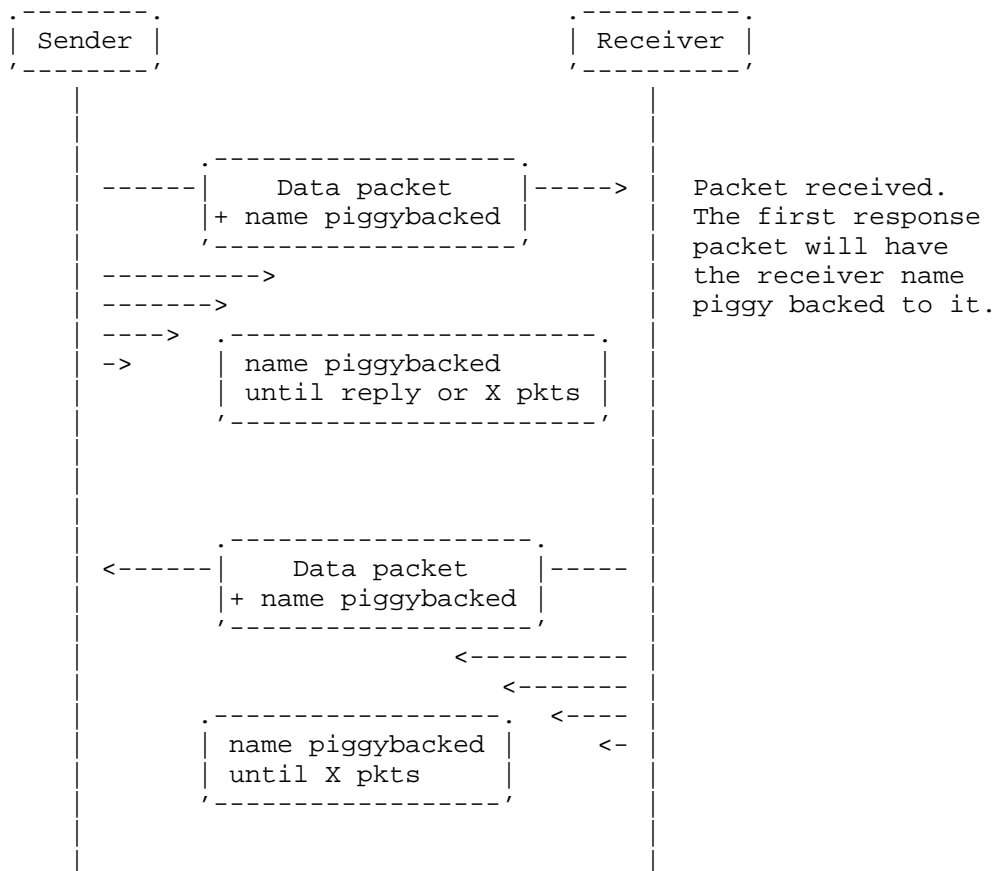


Figure 1

4.1. Name format

Names can be provided in any of three ways.

- o FQDN. The Fully Qualified Domain Name of the host. This will allow e.g. DNSsec to provide authenticity of the name.
- o ip6.arpa. Using one of the hosts interfaces addresses as a name.
- o Nonce. A one-use only session identifier.

4.2. Service names (and port numbers)

Services are identified by a string. For compatibility reasons, it is reasonable to use the "Keyword" field as an identifier of the service in question.

Excerpt from <http://www.iana.org/assignments/port-numbers>

Keyword	Decimal	Description	References
-----	-----	-----	-----
http	80/tcp	World Wide Web HTTP	
http	80/udp	World Wide Web HTTP	
www	80/tcp	World Wide Web HTTP	
www	80/udp	World Wide Web HTTP	
www-http	80/tcp	World Wide Web HTTP	
www-http	80/udp	World Wide Web HTTP	
http	80/sctp	HTTP	

Keyword and port number mappings.

4.3. Transport selection

Often today, the transport protocol is implied by the service in question. How this is handled remains to be defined. It is however likely that a default transport protocol can be provided depending on which service is requested.

5. API

The API follows the connection oriented model, e.g. TCP. The available calls are:

listen(): Prep for incoming session

```
fd = listen( local_name, peer_name, service, transport );

open():  Initiate outgoing session

fd = open( local_name, peer_name, service, transport );

accept():  Receive incoming session

accept( peer_name, fd );

read():  Receive data

data = read( fd );

write():  Send data

write( fd, data );

close():  Close session

close( fd );
```

Note: In the above examples

local_name The local identifier. Either an explicit name or a wildcard (*). As host may have multiple names, to choose which to listen to, a name must be chosen or a wildcard may be used. In the latter case, on listening, any destination name is accepted; on send, an arbitrary name (valid for the host) may automatically be chosen by the socket.

peer_name The remote identifier. Either an explicit name or a wildcard (*). In the accept() function, a wildcard might be inserted. In this case, all incoming packets will be accepted, independent of sender name.

service The service to be run. In general this will correspond to the keyword field in the IANA Port Numbers registry. E.g. http, ftp and ssh.

transport Transport refers to the chosen transport protocol. This could be e.g.

- * TCP (SOCK_STREAM)
- * UDP (SOCK_DGRAM)

- * SCTP (SOCK_SCTP)
- * DCCP (SOCK_DCCP)
- * NORM ...
- * And so on ...

6. Security Considerations

7. IANA Considerations

name-based sockets requires a new address family (AF_NAME) to be defined.

8. Contributors

9. References

9.1. Normative References

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

9.2. Informative References

- [RFC5533] Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", RFC 5533, June 2009.
- [RFC5534] Arkko, J. and I. van Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming", RFC 5534, June 2009.
- [I-D.cotton-tsvwg-iana-ports] Cotton, M., Eggert, L., Mankin, A., and M. Westerlund, "IANA Allocation Guidelines for TCP and UDP Port Numbers", draft-cotton-tsvwg-iana-ports-00 (work in progress), February 2008.

9.3. URL References

Authors' Addresses

Javier Ubillos
Swedish Institute of Computer Science
Kistagangen 16
Kista
Sweden

Phone: +46767647588
EMail: jav@sics.se

Mingwei Xu
Tsinghua University
FIT Building 4-104, Tsinghua University
Beijing
China

EMail: xmw@cernet.edu.cn

Zhongxing Ming
Tsinghua University
FIT Building 4-104, Tsinghua University
Beijing
China

EMail: mingzx@126.com

Christian Vogt
Ericsson
200 Holger Way
San Jose, CA 95134-1300
USA

EMail: christian.vogt@ericsson.com

SAM Research Group
Internet-Draft
Intended status: Informational
Expires: January 29, 2011

M. Waehlich
link-lab & FU Berlin
T C. Schmidt
HAW Hamburg
S. Venaas
cisco Systems
July 28, 2010

A Common API for Transparent Hybrid Multicast
draft-waehlich-sam-common-api-04

Abstract

Group communication services exist in a large variety of flavors, and technical implementations at different protocol layers. Multicast data distribution is most efficiently performed on the lowest available layer, but a heterogeneous deployment status of multicast technologies throughout the Internet requires an adaptive service binding at runtime. Today, it is difficult to write an application that runs everywhere and at the same time makes use of the most efficient multicast service available in the network. Facing robustness requirements, developers are frequently forced to using a stable, upper layer protocol controlled by the application itself. This document describes a common multicast API that is suitable for transparent communication in underlay and overlay, and grants access to the different multicast flavors. It proposes an abstract naming by multicast URIs and discusses mapping mechanisms between different namespaces and distribution technologies. Additionally, it describes the application of this API for building gateways that interconnect current multicast domains throughout the Internet.

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 January 29, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. Use Cases for the Common API	5
2. Terminology	6
3. Overview	7
3.1. Objectives and Reference Scenarios	7
3.2. Group Communication API & Protocol Stack	8
3.3. Naming and Addressing	10
3.4. Mapping	11
4. Common Multicast API	12
4.1. Abstract Data Types	12
4.1.1. Multicast URI	12
4.1.2. Interface	12
4.2. Group Management Calls	13
4.2.1. Create	13
4.2.2. Destroy	13
4.2.3. Join	14
4.2.4. Leave	14
4.2.5. Source Register	14
4.2.6. Source Deregister	15
4.3. Send and Receive Calls	15
4.3.1. Send	15
4.3.2. Receive	16
4.4. Socket Options	16
4.4.1. Get Interfaces	16
4.4.2. Add Interface	16
4.4.3. Delete Interface	17
4.4.4. Set TTL	17
4.5. Service Calls	17
4.5.1. Group Set	17
4.5.2. Neighbor Set	18

4.5.3. Children Set	18
4.5.4. Parent Set	19
4.5.5. Designated Host	19
4.5.6. Update Listener	20
5. Functional Details	20
5.1. Namespaces	20
5.2. Mapping	21
6. IANA Considerations	21
7. Security Considerations	21
8. Acknowledgements	21
9. Informative References	21
Appendix A. Practical Example of the API	23
Appendix B. Deployment Use Cases for Hybrid Multicast	24
B.1. DVMRP	24
B.2. PIM-SM	24
B.3. PIM-SSM	25
B.4. BIDIR-PIM	26
Appendix C. Change Log	26
Authors' Addresses	27

1. Introduction

Currently, group application programmers need to make the choice of the distribution technology that the application will require at runtime. There is no common communication interface that abstracts multicast transmission and subscriptions from the deployment state at runtime. The standard multicast socket options [RFC3493], [RFC3678] are bound to an IP version and do not distinguish between naming and addressing of multicast identifiers. Group communication, however, is commonly implemented in different flavors such as any source (ASM) vs. source specific multicast (SSM), on different layers (e.g., IP vs. application layer multicast), and may be based on different technologies on the same tier as with IPv4 vs. IPv6. It is the objective of this document to provide a universal access to group services.

Multicast application development should be decoupled of technological deployment throughout the infrastructure. It requires a common multicast API that offers calls to transmit and receive multicast data independent of the supporting layer and the underlying technological details. For inter-technology transmissions, a consistent view on multicast states is needed, as well. This document describes an abstract group communication API and core functions necessary for transparent operations. Specific implementation guidelines with respect to operating systems or programming languages are out-of-scope of this document.

In contrast to the standard multicast socket interface, the API introduced in this document abstracts naming from addressing. Using a multicast address in the current socket API predefines the corresponding routing layer. In this specification, the multicast name used for joining a group denotes an application layer data stream that is identified by a multicast URI, independent of its binding to a specific distribution technology. Such a group name can be mapped to variable routing identifiers.

The aim of this common API is twofold:

- o Enable any application programmer to implement group-oriented data communication independent of the underlying delivery mechanisms. In particular, allow for a late binding of group applications to multicast technologies that makes applications efficient, but robust with respect to deployment aspects.
- o Allow for a flexible namespace support in group addressing, and thereby separate naming and addressing/routing schemes from the application design. This abstraction does not only decouple programs from specific aspects of underlying protocols, but may

open application design to extend to specifically flavored group services.

Multicast technologies may be of various P2P kinds, IPv4 or IPv6 network layer multicast, or implemented by some other application service. Corresponding namespaces may be IP addresses or DNS naming, overlay hashes or other application layer group identifiers like <sip:*@peanuts.org>, but also names independently defined by the applications. Common namespaces are introduced later in this document, but follow an open concept suitable for further extensions.

This document also proposes and discusses mapping mechanisms between different namespaces and forwarding technologies. Additionally, the multicast API provides internal interfaces to access current multicast states at the host. Multiple multicast protocols may run in parallel on a single host. These protocols may interact to provide a gateway function that bridges data between different domains. The application of this API at gateways operating between current multicast instances throughout the Internet is described, as well.

1.1. Use Cases for the Common API

Four generic use cases can be identified that require an abstract common API for multicast services:

Application Programming Independent of Technologies Application programmers are provided with group primitives that remain independent of multicast technologies and its deployment in target domains. They are thus enabled to develop programs once that run in every deployment scenario. The employment of group names in the form of abstract meta data types allows applications to remain namespace-agnostic in the sense that the resolution of namespaces and name-to-address mappings may be delegated to a system service at runtime. Thereby, the complexity is minimized as developers need not care about how data is distributed in groups, while the system service can take advantage of extended information of the network environment as acquired at startup.

Global Identification of Groups Groups can be identified independent of technological instantiations and beyond deployment domains. Taking advantage of the abstract naming, an application is thus enabled to match data received from different interface technologies (e.g., IPv4, IPv6, or overlays) to belong to the same group. This not only increases flexibility, an application may for instance combine heterogeneous multipath streams, but simplifies the design and implementation of gateways and translators.

Simplified Service Deployment through Generic Gateways The API allows for an implementation of abstract gateway functions with mappings to specific technologies residing at a system level. Such generic gateways may provide a simple bridging service and facilitate an inter-domain deployment of multicast.

Mobility-agnostic Group Communication Group naming and management as foreseen in the API remain independent of locators. Naturally, applications stay unaware of any mobility-related address changes. Handover-initiated re-addressing is delegated to the mapping services at the system level and may be designed to smoothly interact with mobility management solutions provided at the network or transport layer.

2. Terminology

This document uses the terminology as defined for the multicast protocols [RFC2710],[RFC3376],[RFC3810],[RFC4601],[RFC4604]. In addition, the following terms will be used.

Group Address: A Group Address is a routing identifier. It represents a technological specifier and thus reflects the distribution technology in use. Multicast packet forwarding is based on this ID.

Group Name: A Group Name is an application identifier that is used by applications to manage communication in a multicast group (e.g., join/leave and send/receive). The Group Name does not predefine any distribution technologies, even if it syntactically corresponds to an address, but represents a logical identifier.

Multicast Namespace: A Multicast Namespace is a collection of designators (i.e., names or addresses) for groups that share a common syntax. Typical instances of namespaces are IPv4 or IPv6 multicast addresses, overlay group ids, group names defined on the application layer (e.g., SIP or Email), or some human readable strings.

Multicast Domain: A Multicast Domain hosts nodes and routers of a common, single multicast forwarding technology and is bound to a single namespace.

Interface An Interface is a forwarding instance of a distribution technology on a given node. For example, the IP interface 192.168.1.1 at an IPv4 host.

Inter-domain Multicast Gateway: An Inter-domain Multicast Gateway (IMG) is an entity that interconnects different multicast domains. Its objective is to forward data between these domains, e.g., between IP layer and overlay multicast.

3. Overview

3.1. Objectives and Reference Scenarios

The default use case addressed in this document targets at applications that participate in a group by using some common identifier taken from some common namespace. This group name is typically learned at runtime from user interaction like the selection of an IPTV channel, from dynamic session negotiations like in the Session Initiation Protocol (SIP), but may as well have been predefined for an application as a common group name. Technology-specific system functions then transparently map the group name to group addresses such that

- o programmers are enabled to process group names in their programs without the need to consider technological mappings to designated deployments in target domains;
- o applications are enabled to identify packets that belong to a logically named group, independent of the interface technology used for sending and receiving packets. The latter shall also hold for multicast gateways.

This document refers to a reference scenario that covers the following two hybrid deployment cases displayed in Figure 1:

1. Multicast domains running the same multicast technology but remaining isolated, possibly only connected by network layer unicast.
2. Multicast domains running different multicast technologies, but hosting nodes that are members of the same multicast group.

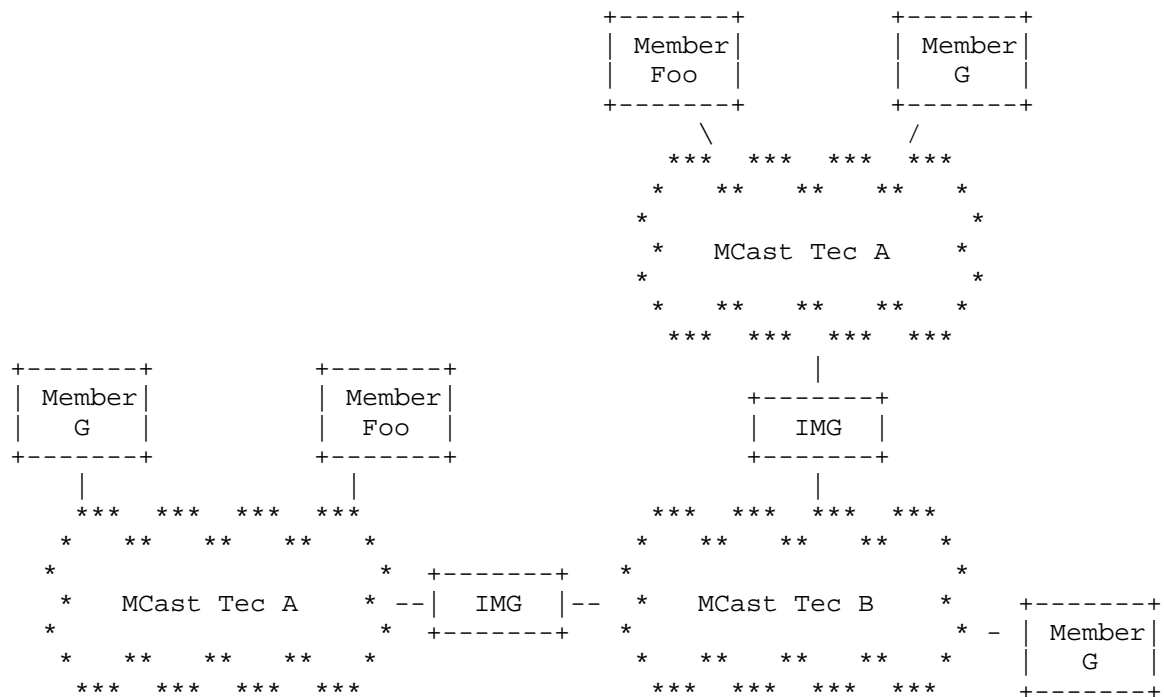


Figure 1: Reference scenarios for hybrid multicast, interconnecting group members from isolated homogeneous and heterogeneous domains.

It is assumed throughout the document that the domain composition, as well as the node attachment to a specific technology remain unchanged during a multicast session.

3.2. Group Communication API & Protocol Stack

The group communication API consists of four parts. Two parts combine the essential communication functions, while the remaining two offer optional extensions for an enhanced management:

Group Management Calls provide the minimal API to instantiate a multicast socket and manage group membership.

Send/Receive Calls provide the minimal API to send and receive multicast data in a technology-transparent fashion.

Socket Options provide extension calls for an explicit configuration of the multicast socket like setting hop limits or associated interfaces.

Service Calls provide extension calls that grant access to internal multicast states of an interface such as the multicast groups under subscription or the multicast forwarding information base.

Multicast applications that use the common API require assistance by a group communication stack. This protocol stack serves two needs:

- o It provides system-level support to transfer the abstract functions of the common API, including namespace support, into protocol operations at interfaces.
- o It bridges data distribution between different multicast technologies.

A general initiation of a multicast communication in this setting proceeds as follows:

1. An application opens an abstract multicast socket.
2. The application subscribes/leaves/(de)registers to a group using a logical group identifier.
3. An intrinsic function of the stack maps the logical group ID (Group Name) to a technical group ID (Group Address). This function may make use of deployment-specific knowledge such as available technologies and group address management in its domain.
4. Packet distribution proceeds to and from one or several multicast-enabled interfaces.

The multicast socket describes a group communication channel composed of one or multiple interfaces. A socket may be created without explicit interface association by the application, which leaves the choice of the underlying forwarding technology to the group communication stack. However, an application may also bind the socket to one or multiple dedicated interfaces, which predefines the forwarding technology and the namespace(s) of the Group Address(es).

Applications are not required to maintain mapping states for Group Addresses. The group communication stack accounts for the mapping of the Group Name to the Group Address(es) and vice versa. Multicast data passed to the application will be augmented by the corresponding Group Name. Multiple multicast subscriptions thus can be conducted

on a single multicast socket without the need for Group Name encoding at the application side.

Hosts may support several multicast protocols. The group communication stack discovers available multicast-enabled communication interfaces. It provides a minimal hybrid function that bridges data between different interfaces and multicast domains. Details of service discovery are out-of-scope of this document.

The extended multicast functions can be implemented by a middleware as conceptually visualized in Figure 2.

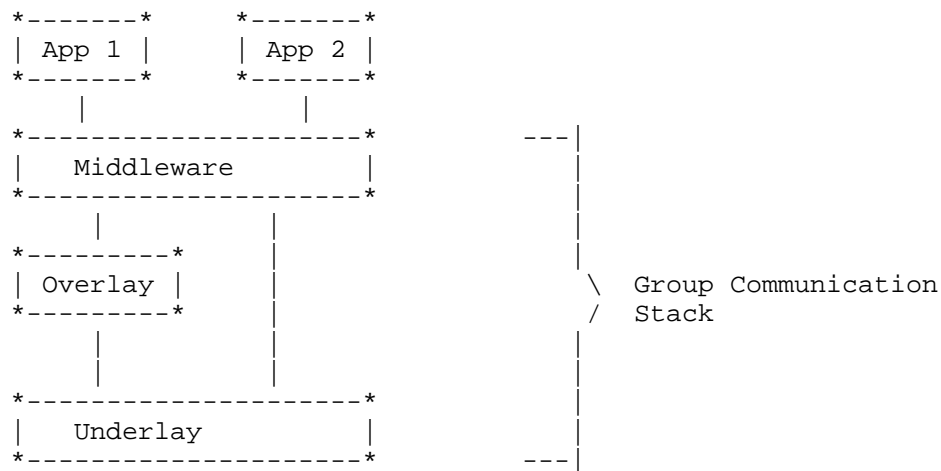


Figure 2: A middleware for offering uniform access to multicast in underlay and overlay

3.3. Naming and Addressing

Applications use Group Names to identify groups. Names can uniquely determine a group in a global communication context and hide technological deployment for data distribution from the application. In contrast, multicast forwarding operates on Group Addresses. Even though both identifiers may be identical in symbols, they carry different meanings. They may also belong to different namespaces. The namespace of a Group Address reflects a routing technology, while the namespace of a Group Name represents the context in which the application operates.

URIs [RFC3986] are a common way to represent namespace-specific identifiers in applications in the form of an abstract meta-data type. Throughout this document, any kind of Group Name follows a URI notation with the syntax defined in Section 4.1.1. Examples are,

ip://224.1.2.3:5000 for a canonical IPv4 ASM group,
sip://news@cnn.com for an application-specific naming with service
instantiator and default port selection.

An implementation of the group communication middleware can provide convenience functions that detect the namespace of a Group Name and use it to optimize service instantiation. In practice, such a library would provide support for high-level data types to the application, similar to the current socket API (e.g., InetAddress in Java). Using this data type could implicitly determine the namespace. Details of automatic namespace identification is out-of-scope of this document.

3.4. Mapping

All group members subscribe to the same Group Name taken from a common namespace and thereby identify the group in a technology-agnostic way.

Group Names require a mapping to addresses prior to service instantiation at an Interface. Similarly, a mapping is needed at gateways to translate between Group Addresses from different namespaces. Some namespaces facilitate a canonical transformation to default address spaces. For example, ip://224.1.2.3:5000 has an obvious correspondence to 224.1.2.3 in the IPv4 multicast address space. Note that in this example the multicast URI can be completely recovered from any data packet received from this group.

However, mapping in general can be more complex and need not be invertible. Mapping functions can be stateless in some contexts, but may require states in others. The application of such functions depends on the cardinality of the namespaces, the structure of address spaces, and possible address collisions. For example, it is not obvious how to map a large identifier space (e.g., IPv6) to a smaller, collision-prone set like IPv4.

Two (or more) Multicast Addresses from different namespaces may belong to

- a. the same logical group (i.e., same Multicast Name)
- b. different multicast channels (i.e., different technical IDs).

This decision can be based on invertible mappings. However, the application of such functions depends on the cardinality of the namespaces and thus does not hold in general. It is not obvious how to map a large identifier space (e.g., IPv6) to a smaller set (e.g., IPv4).

A mapping can be realized by embedding smaller in larger namespaces or selecting an arbitrary, unused ID in the target space. The relation between logical and technical ID is maintained by mapping functions which can be stateless or stateful. The middleware thus queries the mapping service first, and creates a new technical group ID only if there is no identifier available for the namespace in use. The Group Name is associated with one or more Group Addresses, which belong to different namespaces. Depending on the scope of the mapping service, it ensures a consistent use of the technical ID in a local or global domain.

4. Common Multicast API

4.1. Abstract Data Types

4.1.1. Multicast URI

Multicast Names and Multicast Addresses used in this API follow an URI scheme that defines a subset of the generic URI specified in [RFC3986] and is compliant with the guidelines in [RFC4395].

The multicast URI is defined as follows:

```
scheme "://" group "@" instantiation ":" port "/" sec-credentials
```

The parts of the URI are defined as follows:

`scheme` refers to the specification of the assigned identifier [RFC3986] which takes the role of the namespace.

`group` identifies the group uniquely within the namespace given in `scheme`.

`instantiation` identifies the entity that generates the instance of the group (e.g., a SIP domain or a source in SSM) using the namespace given in `scheme`.

`port` identifies a specific application at an instance of a group.

`sec-credentials` used to implement security credentials (e.g., to authorize a multicast group access).

4.1.2. Interface

The interface denotes the layer and instance on which the corresponding call will be effective. In agreement with [RFC3493] we identify an interface by an identifier, which is a positive integer

starting at 1.

Properties of an interface are stored in the following struct:

```
struct if_prop {
    unsigned int if_index; /* 1, 2, ... */
    char        *if_name;  /* "eth0", "eth1:1", "lo", ... */
    char        *if_addr;  /* "1.2.3.4", "abc123" ... */
    char        *if_tech;  /* "ip", "overlay", ... */
};
```

The following function retrieves all available interfaces from the system:

```
struct if_prop *if_prop(void);
```

It extends the functions for Interface Identification in [RFC3493] (cf., Section 4).

4.2. Group Management Calls

4.2.1. Create

The create call initiates a multicast socket and provides the application programmer with a corresponding handle. If no interfaces will be assigned based on the call, the default interface will be selected and associated with the socket. The call may return an error code in the case of failures, e.g., due to a non-operational middleware.

```
int createMSocket(uint32_t *if);
```

The if argument denotes a list of interfaces (if_indexes) that will be associated with the multicast socket. This parameter is optional.

On success a multicast socket identifier is returned, otherwise NULL.

4.2.2. Destroy

The destroy call removes the multicast socket.

```
int destroyMSocket(int s);
```

The s argument identifies the multicast socket for destruction.

On success the value 0 is returned, otherwise -1.

4.2.3. Join

The join call initiates a subscription for the given group. Depending on the interfaces that are associated with the socket, this may result in an IGMP/MLD report or overlay subscription.

```
int join(int s, const uri group_name);
```

The `s` argument identifies the multicast socket.

The `group_name` argument identifies the group.

On success the value 0 is returned, otherwise -1.

4.2.4. Leave

The leave call results in an unsubscription for the given Group Name.

```
int leave(int s, const uri group_name);
```

The `s` argument identifies the multicast socket.

The `group_name` identifies the group.

On success the value 0 is returned, otherwise -1.

4.2.5. Source Register

The `srcRegister` call registers a source for a Group on all active interfaces of the socket `s`. This call may assist group distribution in some technologies, the creation of sub-overlays, for example. Not all multicast technologies require this call.

```
int srcRegister(int s, const uri group_name,  
                uint_t num_ifs, uint_t *ifs);
```

The `s` argument identifies the multicast socket.

The `group_name` argument identifies the multicast group to which a source intends to send data.

The `num_ifs` argument holds the number of elements in the `ifs` array. This parameter is optional.

The `ifs` argument points to the list of interface indexes for which the source registration failed. If `num_ifs` was 0 on output, a NULL pointer is returned. This parameter is optional.

If source registration succeeded for all interfaces associated with the socket, the value 0 is returned, otherwise -1.

4.2.6. Source Deregister

The `srcDeregister` indicates that a source does no longer intend to send data to the multicast group. This call may remain without effect in some multicast technologies.

```
int srcDeregister(int s, const uri group_name,
                  uint_t num_ifs, uint_t *ifs);
```

The `s` argument identifies the multicast socket.

The `group_name` argument identifies the multicast group to which a source has stopped to send multicast data.

The `num_ifs` argument holds the number of elements in the `ifs` array.

The `ifs` argument points to the list of interfaces for which the source deregistration failed. If `num_ifs` was 0 on output, a NULL pointer is returned.

If source deregistration succeeded for all interfaces associated with the socket, the value 0 is returned, otherwise -1.

4.3. Send and Receive Calls

4.3.1. Send

The `send` call passes multicast data for a Multicast Name from the application to the multicast socket.

```
int send(int s, const uri group_name,
         size_t msg_len, const void *buf);
```

The `s` argument identifies the multicast socket.

The `group_name` argument identifies the group to which data will be sent.

The `msg_len` argument holds the length of the message to be sent.

The `buf` argument passes the multicast data to the multicast socket.

On success the value 0 is returned, otherwise -1.

4.3.2. Receive

The receive call passes multicast data and the corresponding Group Name to the application.

```
int receive(int s, const uri group_name,
            size_t msg_len, msg *msg_buf);
```

The `s` argument identifies the multicast socket.

The `group_name` argument identifies the multicast group for which data was received.

The `msg_len` argument holds the length of the received message.

The `msg_buf` argument points to the payload of the received multicast data.

On success the value 0 is returned, otherwise -1.

4.4. Socket Options

The following calls configure an existing multicast socket.

4.4.1. Get Interfaces

The `getInterface` call returns an array of all available multicast communication interfaces associated with the multicast socket.

```
int getInterfaces(int s, uint_t num_ifs, uint_t *ifs);
```

The `s` argument identifies the multicast socket.

The `num_ifs` argument holds the number of interfaces in the `ifs` list.

The `ifs` argument points to an array of interface index identifiers.

On success the value 0 or larger is returned, otherwise -1.

4.4.2. Add Interface

The `addInterface` call adds a distribution channel to the socket. This may be an overlay or underlay interface, e.g., IPv6 or DHT. Multiple interfaces of the same technology may be associated with the socket.

```
int addInterface(int s, uint32_t if);
```

The `s` and `if` arguments identify a multicast socket and interface, respectively.

On success the value 0 is returned, otherwise -1.

4.4.3. Delete Interface

The `delInterface` call removes the interface `if` from the multicast socket.

```
int delInterface(int s, uint32_t if);
```

The `s` and `if` arguments identify a multicast socket and interface, respectively.

On success the value 0 is returned, otherwise -1.

4.4.4. Set TTL

The `setTTL` call configures the maximum hop count for the socket a multicast message is allowed to traverse.

```
int setTTL(int s, int h, uint_t num_ifs, uint_t *ifs);
```

The `s` and `h` arguments identify a multicast socket and the maximum hop count, respectively.

The `num_ifs` argument holds the number of interfaces in the `ifs` list. This parameter is optional.

The `ifs` argument points to an array of interface index identifiers. This parameter is optional.

On success the value 0 is returned, otherwise -1.

4.5. Service Calls

4.5.1. Group Set

The `groupSet` call returns all multicast groups registered at a given interface. This information can be provided by group management states or routing protocols. The return values distinguish between sender and listener states.

```
int groupSet(uint32_t if, uint_t *num_groups,
             struct groupSet *groupSet);

struct groupSet {
    uri group_name; /* registered multicast group */
    int type;        /* 0 = listener state, 1 = sender state,
                     2 = sender & listener state */
};
```

The if argument identifies the interface for which states are maintained.

The num_groups argument holds the number of groups in the groupSet array.

The groupSet argument points to an array of group states.

On success the value 0 is returned, otherwise -1.

4.5.2. Neighbor Set

The neighborSet function returns the set of neighboring nodes for a given interface as seen by the multicast routing protocol.

```
int neighborSet(uint32_t if, uint_t *num_neighbors,
                const uri *neighbor_address);
```

The if argument identifies the interface for which neighbors are inquired.

The num_neighbors argument holds the number of addresses in the neighbor_address array.

The neighbor_address argument points to a list of neighboring nodes on a successful return.

On success the value 0 is returned, otherwise -1.

4.5.3. Children Set

The childrenSet function returns the set of child nodes that receive multicast data from a specified interface for a given group. For a common multicast router, this call retrieves the multicast forwarding information base per interface.

```
int childrenSet(uint32_t if, const uri group_name,  
               uint_t *num_children, const uri *child_address);
```

The `if` argument identifies the interface for which children are inquired.

The `group_name` argument defines the multicast group for which distribution is considered.

The `num_children` argument holds the number of addresses in the `child_address` array.

The `child_address` argument points to a list of neighboring nodes on a successful return.

On success the value 0 is returned, otherwise -1.

4.5.4. Parent Set

The `parentSet` function returns the set of neighbors from which the current node receives multicast data at a given interface for the specified group.

```
int parentSet(uint32_t if, const uri group_name, uint_t *num_parents,  
              const uri *parent_address);
```

The `if` argument identifies the interface for which parents are inquired.

The `group_name` argument defines the multicast group for which distribution is considered.

The `num_parents` argument holds the number of addresses in the `parent_address` array.

The `parent_address` argument points to a list of neighboring nodes on a successful return.

On success the value 0 is returned, otherwise -1.

4.5.5. Designated Host

The `designatedHost` function inquires whether the host has the role of a designated forwarder resp. querier, or not. Such an information is provided by almost all multicast protocols to prevent packet duplication, if multiple multicast instances serve on the same subnet.

```
int designatedHost(uint32_t if, const uri *group_name);
```

The if argument identifies the interface for which designated forwarding is inquired.

The group_name argument specifies the group for which the host may attain the role of designated forwarder.

The function returns 1 if the host is a designated forwarder or querier, otherwise 0. The return value -1 indicates an error.

4.5.6. Update Listener

The updateListener function is invoked to inform a group service about a change of listener states for a group. This is the result of receiver new subscriptions or leaves. The group service may call groupSet to get updated information.

```
const uri *updateListener();
```

On success the updateListener function points to the Group Name that experienced a state change, otherwise NULL is returned.

5. Functional Details

In this section, we describe specific functions of the API and the associated system middleware in detail.

5.1. Namespaces

Namespace identifiers in URIs are placed in the scheme element and characterize syntax and semantic of the group identifier. They enable the use of convenience functions and high-level data types while processing URIs. When used in names, they may facilitate a default mapping and a recovery of names from addresses. They characterize its type, when used in addresses.

Compliant to the URI concept, namespace-schemes can be added. Examples of schemes and functions currently foreseen include

IP This namespace is comprised of regular IP node naming, i.e., DNS names and addresses taken from any version of the Internet Protocol. A processor dealing with the IP namespace is required to determine the syntax (DNS name, IP address version) of the group expression.

OLM This namespace covers address strings immediately valid in an overlay network. A processor handling those strings need not be aware of the address generation mechanism, but may pass these values directly to a corresponding overlay.

SIP The SIP namespace is an example of an application-layer scheme that bears inherent group functions (conferencing). SIP conference URIs may be directly exchanged and interpreted at the application, and mapped to group addresses on the system level to generate a corresponding multicast group.

Opaque This namespace transparently carries strings without further syntactical information, meanings or associated resolution mechanism.

5.2. Mapping

Group Name to Group Address, SSM/ASM TODO

6. IANA Considerations

This document makes no request of IANA.

7. Security Considerations

This draft does neither introduce additional messages nor novel protocol operations. TODO

8. Acknowledgements

We would like to thank the HAMcast-team, Dominik Charousset, Gabriel Hege, Fabian Holler, Alexander Knauf, Sebastian Meiling, and Sebastian Woelke, at the HAW Hamburg for many fruitful discussions and for their continuous critical feedback while implementing API and a hybrid multicast middleware.

This work is partially supported by the German Federal Ministry of Education and Research within the HAMcast project, which is part of G-Lab.

9. Informative References

[I-D.ietf-mboned-auto-multicast]
Thaler, D., Talwar, M., Aggarwal, A., Vicisano, L., and T.

- Pusateri, "Automatic IP Multicast Without Explicit Tunnels (AMT)", draft-ietf-mboned-auto-multicast-10 (work in progress), March 2010.
- [RFC1075] Waitzman, D., Partridge, C., and S. Deering, "Distance Vector Multicast Routing Protocol", RFC 1075, November 1988.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2710] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, October 1999.
- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002.
- [RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.
- [RFC3678] Thaler, D., Fenner, B., and B. Quinn, "Socket Interface Extensions for Multicast Source Filters", RFC 3678, January 2004.
- [RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.
- [RFC4604] Holbrook, H., Cain, B., and B. Haberman, "Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast", RFC 4604, August 2006.
- [RFC5015] Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano,

"Bidirectional Protocol Independent Multicast (BIDIR-PIM)", RFC 5015, October 2007.

Appendix A. Practical Example of the API

```
-- Application above middleware:

//Initialize multicast socket;
//the middleware selects all available interfaces
MulticastSocket m = new MulticastSocket();

m.join(URI("ip://224.1.2.3:5000"));
m.join(URI("ip://[FF02:0:0:0:0:0:0:3]:6000"));
m.join(URI("sip://news@cnn.com"));

-- Middleware:

join(URI mcAddress) {
    //Select interfaces in use
    for all this.interfaces {
        switch (interface.type) {
            case "ipv6":
                //... map logical ID to routing address
                Inet6Address rtAddressIPv6 = new Inet6Address();
                mapNametoAddress(mcAddress,rtAddressIPv6);
                interface.join(rtAddressIPv6);
            case "ipv4":
                //... map logical ID to routing address
                Inet4Address rtAddressIPv4 = new Inet4Address();
                mapNametoAddress(mcAddress,rtAddressIPv4);
                interface.join(rtAddressIPv4);
            case "sip-session":
                //... map logical ID to routing address
                SIPAddress rtAddressSIP = new SIPAddress();
                mapNametoAddress(mcAddress,rtAddressSIP);
                interface.join(rtAddressSIP);
            case "dht":
                //... map logical ID to routing address
                DHTAddress rtAddressDHT = new DHTAddress();
                mapNametoAddress(mcAddress,rtAddressDHT);
                interface.join(rtAddressDHT);
                //...
        }
    }
}
```

Appendix B. Deployment Use Cases for Hybrid Multicast

This section describes the application of the defined API to implement an IMG.

B.1. DVMRP

The following procedure describes a transparent mapping of a DVMRP-based any source multicast service to another many-to-many multicast technology.

An arbitrary DVMRP [RFC1075] router will not be informed about new receivers, but will learn about new sources immediately. The concept of DVMRP does not provide any central multicast instance. Thus, the IMG can be placed anywhere inside the multicast region, but requires a DVMRP neighbor connectivity. The group communication stack used by the IMG is enhanced by a DVMRP implementation. New sources in the underlay will be advertised based on the DVMRP flooding mechanism and received by the IMG. Based on this the `updateSender()` call is triggered. The relay agent initiates a corresponding join in the native network and forwards the received source data towards the overlay routing protocol. Depending on the group states, the data will be distributed to overlay peers.

DVMRP establishes source specific multicast trees. Therefore, a graft message is only visible for DVMRP routers on the path from the new receiver subnet to the source, but in general not for an IMG. To overcome this problem, data of multicast senders will be flooded in the overlay as well as in the underlay. Hence, an IMG has to initiate an all-group join to the overlay using the namespace extension of the API. Each IMG is initially required to forward the received overlay data to the underlay, independent of native multicast receivers. Subsequent prunes may limit unwanted data distribution thereafter.

B.2. PIM-SM

The following procedure describes a transparent mapping of a PIM-SM-based any source multicast service to another many-to-many multicast technology.

The Protocol Independent Multicast Sparse Mode (PIM-SM) [RFC4601] establishes rendezvous points (RP). These entities receive listener and source subscriptions of a domain. To be continuously updated, an IMG has to be co-located with a RP. Whenever PIM register messages are received, the IMG must signal internally a new multicast source using `updateSender()`. Subsequently, the IMG joins the group and a shared tree between the RP and the sources will be established, which

may change to a source specific tree after a sufficient number of data has been delivered. Source traffic will be forwarded to the RP based on the IMG join, even if there are no further receivers in the native multicast domain. Designated routers of a PIM-domain send receiver subscriptions towards the PIM-SM RP. The reception of such messages invokes the `updateListener()` call at the IMG, which initiates a join towards the overlay routing protocol. Overlay multicast data arriving at the IMG will then transparently be forwarded in the underlay network and distributed through the RP instance.

B.3. PIM-SSM

The following procedure describes a transparent mapping of a PIM-SSM-based source specific multicast service to another one-to-many multicast technology.

PIM Source Specific Multicast (PIM-SSM) is defined as part of PIM-SM and admits source specific joins (S,G) according to the source specific host group model [RFC4604]. A multicast distribution tree can be established without the assistance of a rendezvous point.

Sources are not advertised within a PIM-SSM domain. Consequently, an IMG cannot anticipate the local join inside a sender domain and deliver a priori the multicast data to the overlay instance. If an IMG of a receiver domain initiates a group subscription via the overlay routing protocol, relaying multicast data fails, as data are not available at the overlay instance. The IMG instance of the receiver domain, thus, has to locate the IMG instance of the source domain to trigger the corresponding join. In the sense of PIM-SSM, the signaling should not be flooded in underlay and overlay.

One solution could be to intercept the subscription at both, source and receiver sites: To monitor multicast receiver subscriptions (`updateListener()`) in the underlay, the IMG is placed on path towards the source, e.g., at a domain border router. This router intercepts join messages and extracts the unicast source address S, initializing an IMG specific join to S via regular unicast. Multicast data arriving at the IMG of the sender domain can be distributed via the overlay. Discovering the IMG of a multicast sender domain may be implemented analogously to AMT [I-D.ietf-mboned-auto-multicast] by anycast. Consequently, the source address S of the group (S,G) should be built based on an anycast prefix. The corresponding IMG anycast address for a source domain is then derived from the prefix of S.

B.4. BIDIR-PIM

The following procedure describes a transparent mapping of a BIDIR-PIM-based any source multicast service to another many-to-many multicast technology.

Bidirectional PIM [RFC5015] is a variant of PIM-SM. In contrast to PIM-SM, the protocol pre-establishes bidirectional shared trees per group, connecting multicast sources and receivers. The rendezvous points are virtualized in BIDIR-PIM as an address to identify on-tree directions (up and down). However, routers with the best link towards the (virtualized) rendezvous point address are selected as designated forwarders for a link-local domain and represent the actual distribution tree. The IMG is to be placed at the RP-link, where the rendezvous point address is located. As source data in either cases will be transmitted to the rendezvous point address, the BIDIR-PIM instance of the IMG receives the data and can internally signal new senders towards the stack via `updateSender()`. The first receiver subscription for a new group within a BIDIR-PIM domain needs to be transmitted to the RP to establish the first branching point. Using the `updateListener()` invocation, an IMG will thereby be informed about group requests from its domain, which are then delegated to the overlay.

Appendix C. Change Log

The following changes have been made from
draft-waehlich-sam-common-api-03

1. Use cases added for illustration.
2. Service calls added for inquiring on the multicast distribution system.
3. Namespace examples added.
4. Clarifications and editorial improvements.

The following changes have been made from
draft-waehlich-sam-common-api-02

1. Rename `init()` in `createMSocket()`.
2. Added calls `srcRegister()/srcDeregister()`.
3. Rephrased API calls in C-style.

4. Cleanup code in "Practical Example of the API".
5. Partial reorganization of the document.
6. Many editorial improvements.

The following changes have been made from
draft-waehlich-sam-common-api-01

1. Document restructured to clarify the realm of document overview and specific contributions s.a. naming and addressing.
2. A clear separation of naming and addressing was drawn. Multicast URIs have been introduced.
3. Clarified and adapted the API calls.
4. Introduced Socket Option calls.
5. Deployment use cases moved to an appendix.
6. Simple programming example added.
7. Many editorial improvements.

Authors' Addresses

Matthias Waehlich
link-lab & FU Berlin
Hoenower Str. 35
Berlin 10318
Germany

Email: mw@link-lab.net
URI: <http://www.inf.fu-berlin.de/~waehl>

Thomas C. Schmidt
HAW Hamburg
Berliner Tor 7
Hamburg 20099
Germany

Email: schmidt@informatik.haw-hamburg.de
URI: <http://inet.cpt.haw-hamburg.de/members/schmidt>

Stig Venaas
cisco Systems
Tasman Drive
San Jose, CA 95134
USA

Email: stig@cisco.com

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: March 21, 2011

M. Xu
Z. Ming
Tsinghua University
J. Ubillos
Swedish Institute of Computer
Science
C. Vogt
Ericsson
September 17, 2010

Name Based Sockets - Shim6
draft-xu-name-shim6-00

Abstract

This document describes and defines shim6 as a mobility solution for name-based sockets. Using names rather than pseudo IP addresses, shim6 can handle a more diverse set of mobility scenarios. These changes allow a shim6 session to persist even through cases where one node has no working locators to its correspondent node. If the name is also a resolvable fully qualified domain name, the connection can be kept alive even if neither node have a working locator to the corresponding node. As can be the case if both nodes are mobile simultaneously.

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 March 21, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Conventions	3
2. Terminology	3
3. Overview	3
3.1. Introduction	3
4. Mobility support	4
4.1. Shim6	4
4.1.1. Brief overview of changes	4
4.1.2. Identity change	5
4.1.3. The hand-shake with name exchange	5
4.1.4. Triggers of shim6	6
4.1.5. Establishing Shim6 context	6
4.1.6. Problems for Shim6 to support mobility	6
4.1.7. Changes to REAP	8
4.1.8. Changes to STATE Machine	8
5. Security Considerations	9
6. IANA Considerations	9
7. Contributors	9
8. References	9
8.1. Normative References	9
8.2. Informative References	9
8.3. URL References	9

1. Conventions

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].

2. Terminology

Locator - An IP address (v4 or v6) on which a host can be reached.

Multi-home - A host which is reachable through multiple locators (on one interface or more)

Name - A character string (max 255 chars long) on which an endpoint can be identified. A name maps to zero or more locators.

3. Overview

3.1. Introduction

Mobility can be treated as a special case of multihoming. Shim6 provides a promising way to implement multihoming for upper layer protocols, thus it is reasonable to use Shim6 to provide NBS with mobility functionality.

The traditional Shim6 defined in RFC5533 [RFC5533] does not aim to solve mobility problem, so changes need to be made to the existing Shim6 protocol. One of the reasons for not supporting mobility is that Shim6 uses a specific IP address as the identifier of the upper layer protocol. To avoid confusion, communication must be stopped when this IP address becomes unavailable. With the presence of name based socket, Shim6 can use the name as the upper layer identifier rather than IP. In such a way, Shim6 will not encounter the problem that connection must be terminated when the locator used as ULID becomes invalid.

To allow connections to survive during movement, Shim6 context should be preserved for a certain period of time when there are no available locator to use. When both nodes move simultaneously, Shim6 path exploration may fail due to lost update messages. To keep the connection from being terminated in this case, DNS is involved to provide address information.

Technical details will be covered in this document.

4. Mobility support

4.1. Shim6

"... The Shim6 protocol, a layer 3 shim for providing locator agility below the transport protocols, so that multihoming can be provided for IPv6 with failover and load-sharing properties, without assuming that a multihomed site will have a provider-independent IPv6 address prefix announced in the global IPv6 routing table. The hosts in a site that has multiple provider- allocated IPv6 address prefixes will use the Shim6 protocol specified in this document to set up state with peer hosts so that the state can later be used to failover to a different locator pair, should the original one stop working. " RFC5533 [RFC5533]

4.1.1. Brief overview of changes

To the upper layers, shim6 provides a stable IP address-like identifier (ULID) to identify the remote host and make the IP addresses (locators) transparent to the application. This way of providing a pseudo-address (ULID) does however invite confusion. The ULID selected by Shim6 is actually the IP address which is available for the application when the connection is being established. This address (ULID) may become invalid during the connection (RFC5533 Section 1.5 [RFC5533]). ULID invalidation is beyond the control of the individual hosts, it is controlled by the network. This might cause confusion if the applications continues to use the ULIDs which are no longer valid. Shim6's solution to this problem is to terminate the communication immediately when ever any ULID becomes invalid. This is definitely inappropriate in a mobile scenario as connections are expected to be preserved during the mobile period. Moving between two distinct networks, changing your complete locator set is the common scenario (e.g. entirely switching from one WiFi-provider to another.)

Name Based Sockets suggest using the name of a host as the identifier. This solves the above problems, as a name is valid for as long as a host wishes it to be. Also, as Name Based Sockets provide a new explicit interface (names rather than 'pseudo IP addresses'), applications that use it will be aware of the available features, and may make correct assessments of the underlying IP stack and its enhancements.

This document describes a set of changes and improvements to shim6 that are to be incorporated with the Name Based Sockets.

Briefly, the changes are:

- o Name is used as ULID rather than an IP (Section 4.1.2).
- o Node inter-reachability resilience, for when both nodes are simultaneously mobile using DNS (Section 4.1.6.1).

4.1.2. Identity change

Shim6 selects a locator (IP) in the initial contact with the remote peer and uses this locator as an upper-layer identifier (ULID). To support NBS, we use name (or some structure related to name) as ULID instead of IP addresses.

Because the end point identifier is no longer a locator but rather a name, the initial name exchange is performed by NBS and Shim6 uses this name to construct the ULID.

Shim6 requires that any communication using a ULID MUST be terminated when the ULID becomes invalid. Using names as ULIDs instead of IP address is more in line with the transport semantic. Having names as ULIDs means that the session may still exist even if both communicating hosts locator lists are empty at a given point of time. This is particularly important when one or both peer(s) are moving.

Note that replacing a ULID with a name does not necessarily mean representing the ULID as a string or a string-like structure internally. In order to lower the complexity, one should make the least possible modification to both the Shim6 protocol (where ULID is a 128-bit IPv6 address) and its definition implementation, we propose to internally represent the name as a 128-bit MD5-hash and use this MD5-hash as the corresponding ULID.

4.1.3. The hand-shake with name exchange

As is described in RFC5533 [RFC5533], Shim6 does not need to react immediately when connections start up. The initial name exchange is performed by NBS and it requires no help from Shim6. The name exchanged by NBS will be further used as ULID by Shim6. At some time during the communication, some heuristic may determine that it is appropriate to use shim6 to support mobility/multi-homing, so the communicating hosts initiate a 4-way, context-establishment exchange. As a result, both hosts get a locator list of each other.

As an extension to Shim6, we do not change the operation sequence of the 4-way exchange, namely the order of I1, R1, I2, R2 will not be changed. What is changed is that the IP-based ULID is replaced by a name-based ULID and the hand shake no longer requires ULID negotiation because it has already been done by NBS.

4.1.4. Triggers of shim6

It is not necessarily worth paying the overhead of setting up a shim context when e.g. only a small number of packets are exchanged between two hosts. As a result, Shim6 functionality will not be started immediately as a new communication is initiated.

NBS uses some heuristic for determining when to perform a deferred context establishment. This heuristic might be that more than 50 packets have been sent or received, or that a timer expires while active packet exchange was in place RFC 5533 [RFC5533]. Exactly how the heuristic is designed is beyond the scope of this document.

4.1.5. Establishing Shim6 context

At a certain time during the connection, some heuristic on host A or B (or both) determine that it is appropriate to pay the Shim6 overhead to improve host-to-host communication. This makes the Shim6 initiate the 4-way, context-establishment exchange (defined in RFC 5533).

As a result, both A and B get a list of each others locators. In name-based Shim6, the ULID is represented as a MD5-hash of name rather than IP.

4.1.6. Problems for Shim6 to support mobility

When only one host moves to a new network, a REAP Update is triggered to prevent connection from being terminated. Under normal circumstances, connection will be smoothly preserved during the REAP Update process.

However, REAP itself is not sufficient to support full mobility functionality, as when both hosts move simultaneously, neither of them will receive the update message, which will lead to a connection loss. To deal with this problem, DNS should be involved to provide address information.

4.1.6.1. DNS querying

An effective solution for the mobility problem is to have a "stationary infrastructure" to provide address information for all mobile devices. We propose to use DNS as the stationary infrastructure as it associates addresses with names and has enough capability. How DNS incorporates with name-based Shim6 is described in the following part.

4.1.6.2. One peer moves

In the case that only one host moves, the moving host starts a REAP Update process to re-establish Shim6 context with the correspondent host. At the same time, DNS should be updated by the moving host. This procedure is the normal REAP [RFC5534] procedure with the added update to DNS.

The following sequence illustrates the details:

1. Two hosts, A and B are communicating using NBS and Shim6.
2. At certain moment, A moves to a new network and changes its IP address (locator).
3. A updates the authoritative DNS with its new IP address. In parallel, A starts the REAP Update process by sending B an Update Request and the REAP Update process is invoked.
4. New operational locator pair is found by REAP Update process.
5. Handover process is completed and connection is preserved during the process.

4.1.6.3. Both peers move

When both hosts moves simultaneously, neither host will receive the REAP Update Request, thus REAP will fail in finding the new operational locator pair. Under such circumstances, both hosts need to query DNS for the correspondent hosts addresses. When new address is retrieved, both hosts initiate REAP Update process as specified in RFC5534 [RFC5534].

The following sequence illustrates the details:

1. Two hosts, A and B are communicating using NBS and Shim6.
2. At certain moment, both A and B move simultaneously and both hosts change their respective IP addresses.
3. Both hosts update DNS with their new addresses and send REAP Update Request to their correspondent peer.
4. Due to concurrent move, Update Requests are lost for both directions.
5. Both hosts experience an Update Timeout and query DNS for correspondent hosts' locators using their names.

6. New addresses are returned by the respective DNS queries and REAP Update is now able to operate. A and B re-invoke a REAP Update process using the new addresses.
7. New operational locator pair is found by REAP Update process.
8. Handover process is completed and connection is preserved during the handover process.

4.1.7. Changes to REAP

We extend REAP by adding DNS Querying into its Path Exploration. For the sake of backwards compatibility, DNS can be implemented as a separate module and has no impact on the other part of REAP which is specified in RFC 5534. The DNS functionality can be turned off for stationary hosts and be turned on for mobile devices.

In the mobile scenario, DNS Query and REAP Path Exploration may work together to provide stronger reliability. DNS query might be lost due to link failure or timeout due to high network delay. Under such circumstances, REAP Path Exploration will be triggered because of a SEND TIMEOUT and tries to find an available path. This is meaningful when a host has multiple available interfaces (for instance Wi-Fi and 3G) and the address change for one interface does not lead to the change for others.

4.1.8. Changes to STATE Machine

In order to implement mobility, we propose to add a new state `NO_LOC` into the shim6 state machine. With this state, Shim6 does not directly transmit to a dead state when there is no available address, but transmits to `NO_LOC` state to wait for a new locator. There is a minor difference between the Exploring state and the InBoundOK state. For simplicity, in this draft we use the Exploring state to represent both Exploring state and InBoundOK state.

Upon realizing that there is no available address to use, Shim6 sets the state to `NO_LOC`, starts a time and wait for an available address. If the host successfully attaches to a new network, Shim6 performs the following operations:

1. Update the context.
2. Send Update Request to the remote peer.
3. Transitions into the Exploring state.

4. Stops the timer.

If the timer expires, Shim6 removes the context.

Upon entering the Exploring state, Shim6 waits for the Update ACK from the peer. If the update ACK is received, Shim6 can perform the normal operation as defined by RFC 5533 and RFC 5534 to continue the connection. Otherwise, Shim6 infers that the peer may also move and asks the DNS for the peers locator, insert the locator into the locator list of the peer (if not already present in the list) and starts the Reap Path Exploration process. As the DNS stores the latest address of the peer, we believe this locator is most likely to be available, thus in the Reap Path Exploration process, we first try the locator returned by DNS and then other locators (if any).

5. Security Considerations

6. IANA Considerations

7. Contributors

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

[RFC5533] Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6", RFC 5533, June 2009.

[RFC5534] Arkko, J. and I. van Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming", RFC 5534, June 2009.

8.3. URL References

Authors' Addresses

Mingwei Xu
Tsinghua University
FIT Building 4-104, Tsinghua University
Beijing
China

EMail: xmw@cernet.edu.cn

Zhongxing Ming
Tsinghua University
FIT Building 4-104, Tsinghua University
Beijing
China

EMail: mingzx@126.com

Javier Ubillos
Swedish Institute of Computer Science
Kistagangen 16
Kista
Sweden

Phone: +46767647588
EMail: jav@sics.se

Christian Vogt
Ericsson
200 Holger Way
San Jose, CA 95134-1300
USA

EMail: christian.vogt@ericsson.com

