Host Anycast Support in IPv6

Sameer Shah and Dheeraj Sanghi
Department of Computer Science & Engineering
Indian Institute of Technology
Kanpur - 208 016 (UP), INDIA.
dheeraj@iitk.ernet.in

Abstract

The IPv6 addressing architecture recognizes a need for anycast addressing. But there are problems in using anycast addresses. They cannot be used as source address in any case, and cannot be used as destination address in stateful protocols (such as TCP).

In this paper, we discuss a solution to anycast addressing problem. We specify the requirements for hosts to support anycasting. Actions are specified at the network, transport and API layers. These host requirements were implemented and tested on a Linux based IPv6 stack.

1 Introduction

While designing the next generation IP [5], a need for *anycast addressing* was felt. Anycast address is similar to a multicast address in the sense that it is assigned to a group of interfaces. But it is similar to unicast address in that a packet sent to this address is delivered to only one interface. A packet sent to an anycast address is routed to the "nearest" interface having that address, according to the routing protocols' measure of distance [4].

Some uses of anycasting in router addressing are - to identify set of routers belonging to an Internet service provider to enable provider selection; to identify set of routers attached to a subnet, e.g., the home network for mobile nodes. In all these scenarios an anycast address is obtained from the network prefix assigned to the particular level of hierarchy the routers belong to and do not require special routing in the backbone routers.

There are various scenarios of host addressing where anycasting can be beneficial. An example is load sharing - all servers providing a service can be members of an anycast group, and a client's request is automatically delivered to the nearest server. Use of anycasting leads to optimal routing paths and better network utilization.

Despite the benefits of host anycasting, current specifications [4] prohibit assignment of anycast address to an IPv6 host. There are two main problems:

- For every anycast address, there is a host-specific routing table entry in routers in the entire topological region covered by these hosts. This is acceptable if the region is small, like a campus-wide network. But if there is a worldwide anycast group, then this implies having a host specific entry for an address in Internet backbone routers. This leads to routing table explosion and associated overheads in critical backbone routers.

- An anycast address identifies multiple nodes. It cannot be used as source address in an IPv6 packet due to inability to identify the originating node. Also successive packets sent to an anycast address may not all be delivered to the same node. So, stateful protocols cannot rely on the use of an anycast destination address throughout a session. Additionally a packet can be received by multiple anycast nodes due to duplication and misrouting. So more than one node can reply simultaneously.

Amongst these, the former is unavoidable given the addressing architecture of IPv6. We believe that the use of anycast addresses will be restricted to organizational Intranets. There may also be a mechanism to create global anycast groups in a regulated fashion. In this paper we provide a solution to the latter problem.

A host anycasting service was proposed for IPv4 networks by Partridge, et. al. [8]. They discuss ideas on semantics of usage and suggestions for implementation. They allow use of an anycast source address in IP datagrams. A separate class of IP address is proposed for

anycast allocation. This way TCP can *recognize* an anycast address, and treat this differently. An anycast destination address can only be used in the first segment. The remote host uses a non-anycast source address in the reply. Demultiplexing treats anycast address as wildcard (as they are recognizable), so the reply matches the request. The requesting node then replaces the anycast destination with the returned source address. Future exchanges use only the non-anycast address. A major drawback of this scheme is use of a separate space for anycast allocation. It has not been implemented in IPv4 networks.

Bound and Roque [3] proposes a solution for a IPv6 based host anycasting service. A new Destination option is proposed - the Source Identification option. A node which responds to a request addressed to an anycast address places its unicast address as IP Source address, but also indicates its anycast address in this option. TCP and stateful applications on top of UDP, can use the value of this option for demultiplexing and learning the unicast address of the peer. Further communication solely uses unicast addresses.

In this paper, we extend the anycast model of Bound and Roque in several ways. We define the complete implementation at both ends of a communication that uses host anycast addresses. A set of requirements for a functional and flexible host anycasting service are given. Such a specification is necessary in order to avoid ambiguity in interpretation and to define a correct implementation. In case of TCP we allow use of the proposed option to identify the anycast address of a host doing an active connect. When a Source-Id option is used, it *should* be used in demultiplexing. The earlier model [3] suggests using this as an optional mechanism. The BSD API is extended to allow use of anycasting in a flexible manner. Using the proposed mechanism, many existing TCP and UDP applications continue to work unchanged. Applications that maintain state on top of UDP, across multiple TCP connections, or use end-point addresses at application layer, will need to be modified.

Our implementation allows merging of Source-Id option with the Binding Update option proposed for mobility support in IPv6 [6]. Binding Update conveys information about the care-of addresses associated with a mobile node's home address. An additional bit can be used to indicate the anycast case.

We have implemented these requirements on a Linux 2.1.21 based IPv6 stack and tested in a LAN environment. Normal cases of usage of applications like telnet, ftp, rlogin continue to work unchanged.

2 Background

A brief description on the working of TCP relevant to rest of the paper is given in this section. Implementation of datagram demultiplexing is also discussed.

TCP is an end-to-end, reliable, connection-oriented transport layer, that provides resequencing and flow control. It maintains state at the end-points of a connection and provides mechanisms for connection establishment, data exchange and connection release/abort.

A server opens a connection in passive mode, enters LISTEN state and waits on connection requests from clients. A client doing active open sends a segment with the SYN bit set, advertising its initial send sequence number (ISS), and enters SYN_SENT state. The response to a SYN is a segment with both SYN and ACK set, acknowledging the clients' ISS and advertising the server's ISS. The server enters SYN_RECEIVED state. On receiving a response the initiating client verifies the acknowledged sequence number, acknowledges the server's ISS in a reply segment with ACK set, and enters ESTABLISHED state. On receiving a valid ACK from the client, the server moves into ESTABLISHED state.

TCP provides a mechanism (using RST bit) to reset the peers' connection when a packet is received for which no corresponding state exists locally. This provides recovery in case of loss of synchronization. TCP also allows the case of a simultaneous active open at both ends.

2.1 Demultiplexing

A socket refers to a communication end-point in the Berkeley networking system. An association is a 5-tuple that uniquely identifies a connection comprising {*protocol, local-addr, local-port, foreign-addr, foreign-port*}. We refer to the *protocol* implicitly and refer to this as the four tuple in rest of the discussion. A protocol control block (PCB) is a conceptual data structure containing protocol specific information for a socket. There is usually a one-to-one correspondence between socket and PCB in networking implementation.

The local port and local address are set with using bind(). Usually implementations allow more than one socket to bind to the same local port using a socket option. The remote port and remote address members of the four tuple can be set using `connect()`. When a socket is unconnected, the remote address and remote port are wildcards. A `connect()` on a UDP socket does not involve data exchange, but merely saves the remote address and remote port values in the PCB. A UDP socket can issue `connect()` more than once. Most

Loc-IP	Loc-port	Rem-IP	Rem-port	Description
√	√	√	√	restricted to one client
√	√	*	*	restricted to dgrams arriving on Loc-IP
*	√	*	*	all dgrams sent to Loc-port

Figure 1. Specifying the four tuple

implementations, including ours, implicitly set the remote address to loopback when `connect()` is invoked with a wildcard remote address.

Figure 1 shows possibilities for values in the four tuple [10]. A '√' indicates a specific value while, '*' indicates wildcard. The order of the three rows is the order used in demultiplexing a received datagram to a corresponding socket. The first row with most specific binding is tried first, and the last row with least specific binding is tried last.

3 Proposed Solution

In this section we describe the proposed Source-Id option, and how TCP and UDP should use the option.

3.1 Source Identification Option

The Source Identification option is used by a host to inform a peer about its anycast address. The client node uses an anycast address as the destination address, but the server cannot use anycast address as its source address. If it were to use only its unicast address, the client would not be able to demultiplex it properly. Therefore, the server node, sends its anycast address in Source-Id option, in addition to sending its unicast address as the source address. The client side can demultiplex using the anycast address, and also notes down the unicast address for further communication. The proposed option is encoded in the Destination Extension Header of an IPv6 datagram. The option is shown in Figure 2.

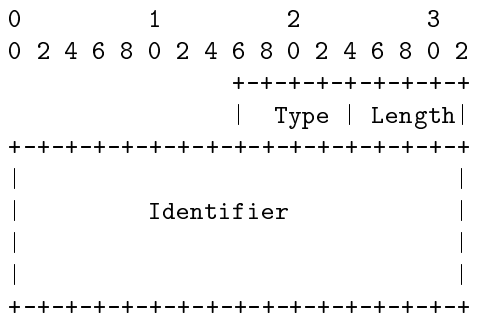

Figure 2. Source Identification Option

Option Type identifies type of the option. The most significant 3 bits are 110. So a node not recognizing the option type must discard the packet. The option data does not change en-route.

Bound and Roque [3] suggests a value of 000 for the most significant bits, which allows ignoring the option and further processing of datagram. But allowing further processing when the option is not recognized will not be useful in most cases of TCP and UDP usage. It may also lead to wrong demultiplexing in few cases.

Option Length is length of the option data field in octets. This carries a value of 16.

Identifier is anycast address of the node originating the datagram whose unicast address is indicated in the Source Address field of the IPv6 base header.

3.2 Requirements for TCP

The mechanism should ensure proper synchronization with use of anycast addresses. A common scenario consists of a client doing active open on the server's anycast address. The client waits for a matching response from the server in SYN_SENT state. The request is received by a server waiting on the anycast address or the wildcard address. In either case, the server response (containing SYN+ACK or an RST) *should* carry a Source-Id option to associate its unicast Source Address with the anycast address. The receiving TCP *should* use the identifier in the option for demultiplexing the received segment. If a matching socket is found, the client should switch to using the server's unicast address for all further exchanges. Otherwise an RST should be sent to the peer's unicast address.

Additionally, we allow the possibility of using an anycast indication during an active connect request, in this case the socket already selects a unicast source address for use during the connection, but it also sends a Source-Id option with the first SYN segment to identify itself with the anycast group. A matching socket at the remote TCP will be in either SYN_SENT or LISTEN state. If the state is SYN_SENT, this corresponds to a simultaneous open scenario. If it is LISTEN, the

socket moves to SYN_RECEIVED state upon receiving the request. The anycast address received in the option is saved in the PCB (separate from the sockets' four tuple).

The above discussion leads to following constraints on TCP interaction with host anycast addressing:

- An anycast destination address can only be used in an active connect request. Thus a segment with anycast destination is valid only if the SYN is set and the ACK or RST bits are not set.
- A Source-Id option can only be sent in an active connect request or in response to an active connect request received on the anycast address. Thus a segment with the option is valid only if the SYN or RST is set. The RST was sent in response to an active connect on the anycast address.

The demultiplexing as mentioned in this subsection is conceptually simple. But there are few subtle cases resulting from TCP's reliability in the face of losses, old incarnations and retransmissions.

3.3 Requirements for UDP

Applications should be able to specify the host's anycast address as a source address in outgoing datagrams. Since IPv6 does not allow anycast addresses as source address, the UDP entity should replace the anycast address by an unicast address before forwarding the packet to IP. It should also attach the Source-Id option to indicate the anycast address selected by application.

Upon receiving a datagram with Source-Id option, demultiplexing should use the identifier in the option as the remote's address. If no match is found, or the recipient application prohibits the option for security purposes, the datagram should be discarded and an ICMP port unreachable message be sent to the sender on its unicast address. Bound and Roque [3] ignore the option if it is not selected by application and allow further processing. In this case demultiplexing can match another socket specifying a remote address equal to the unicast address of destination (provided other members of four tuple are equal). It is incorrect to allow this in a host implementing anycast support, as the intended recipient is a socket connected either to the remote's anycast address or a wildcard address.

An application may *notice* a change in the address of responding node and take necessary action, e.g., issuing a new `connect()` on remote's unicast address. We also provide an API that applications can use to know the anycast address received in the Source-Id option on a per datagram basis. This proposal requires modifications to existing stateful applications. We propose a further extension to minimize the amount of change required. A flag `state_req` is used to indicate if application maintains state and desires UDP layer to guarantee communication with a unique node. When the flag is set, the UDP entity is responsible for noticing and changing the peer's address during the initial exchange.

3.4 Security Issues

A mechanism to allow dynamic change of a peer's address in transport end-points is subject to threats similar to the source routing attack described by Bellovin [2]. For better security the proposed mechanism should be used in conjunction with replay protection and Authentication Header [1] suggested in the mobile case [6].

We have not yet implemented the interaction with the security mechanisms, but some amount of weak security is inherent to the mechanisms we employ. TCP accepts a response to an active connect only if the acknowledged sequence number matches the ISS sent in the request. Spoofing an address requires an impersonator to guess the ISS used by sender. As sequence numbers are chosen with sufficient randomness, it is difficult to guess the ISS, unless the attacking node lies on the same subnet as the sender. But in this case other forms of attacks are also available. For UDP a datagram with Source-Id option received on a connected socket and `state_req` set, changes the remote address in the four-tuple only if it is the first datagram to be received on the socket.

Additionally there may be applications which do not concern with security issues, e.g., a time of day service; or some environments may have a level of trust. In such cases use of the AH can be avoided.

4 Implementation

We discuss the implementation of host anycast support at the network, transport and API layers. Exact algorithms for the case of TCP and UDP are available in [9]. We used the Linux 2.1.21 operating system for implementation and testing of the proposed mechanisms.

The primary objective of the implementation is to provide a mechanism that supports use of host anycast addressing in transport specific ways. The implementation should have minimal per-packet processing

overhead. There should be minimal changes to protocol specifications.

4.1 Network Layer

The network layer encodes a Source-Id option in outgoing datagrams when indicated by the upper layer. On an incoming datagram it decodes the option, and passes the value to the upper layer. It provides a mechanism to configure an anycast address on an interface. Duplicate address detection is not done when configuring an anycast address.

Anycast addresses are treated just like unicast addresses by Neighbour Discovery except for minor exceptions [7]. Neighbour Advertisements sent in response to a Neighbour Solicitation are delayed by a random amount of time between `0` and `MAX_ANYCAST_DELAY_TIME` to reduce probability of network congestion. Also, Neighbour Advertisements carry a value of `0` for the Override flag. Thus when multiple advertisements are received the first one is used, rather than the most recent.

The network layer prohibits selection of anycast address as a source address in outgoing datagrams. When a request is received on anycast address, a unicast source address is selected for use in the response. For this, an outgoing interface for destination is determined using next hop resolution, and an address with proper scope selected from the interface's list of addresses. If such an address is unavailable, any of the host's unicast address with appropriate scope can be selected. The outgoing interface may differ from the one identified by the anycast address. In this case further communication with unicast address may use entirely different route than the route taken for anycast address. This may not always be desirable, but is allowed.

4.2 TCP Layer

Below we discuss the actions at hosts sending and receiving a Source-Id option.

Sending Source Identification

A flag (`sndany`) is used to allow Source-Id option on outgoing segments. If the flag is not set, anycast TCP packets are responded with RST packets. The Source-Id option is sent in two instances: First, request with anycast destination is received on a socket bound to the anycast or wildcard address. Second, a socket bound to anycast address initiates active open. In either case the local anycast address is remembered in the PCB (apart from the four-tuple) for use in demultiplexing.

Receiving Source Identification

On receipt of a segment with Source-Id option, TCP should use the option value in demultiplexing. The exact demultiplexing is discussed later. For now we assume a recipient socket is found in demultiplexing. The action depends on state of the socket. In all of the following cases, a socket changes the remote address in the four-tuple to the remote unicast address, and remembers the remote anycast address also. In all other states the option is ignored.

- If the socket is in SYN_SENT state and segment is a response to an active connect on an anycast address, the state is changed to ESTABLISHED.
- If the socket is in LISTEN state and segment is an active connect request from the anycast node, the state is changed to SYN_RECEIVED.
- If the socket is in SYN_SENT state and segment is an active connect request, this corresponds to simultaneous active open. The state is changed to SYN_RECEIVED.

4.2.1 Demultiplexing

There are two observations to be made here. First, a socket listening on wildcard address prohibits another socket to bind on the same port for any of the local unicast addresses. Since an anycast address also behaves like a wildcard as it can select any of the host's unicast addresses in response to incoming request, no other socket is allowed to bind on the same port for any unicast address.

Second, for a connection doing initial handshake, all incoming segments are matched to four tuple of the socket in LISTEN state. A connection initiated with anycast destination receives future segments with unicast destination which should be matched to the anycast (or wildcard) listening socket. Therefore, we do not allow a socket bound to a wildcard or anycast address to enter LISTEN state, if another socket bound to any of the local addresses, for the same local port, is in LISTEN state.

Demultiplexing function has been modified to consider anycast address as wildcard and anycast destination is allowed only for incoming SYN packets. All future packets should be using the unicast addresses. Another interesting problem is when the same host tries to establish two connections, one using anycast address, and the other using a unicast address. Both the connections cannot be allowed since they will both lead to the same 4-tuple. Only the first connection request succeeds in our implementation.

If the incoming segment contains an anycast destination, and there is no socket that has the matching 4-tuple (or if the socket does not want to accept anycast messages), an RST message is to be sent back. But in this case, the peer only knows the anycast address, and not the corresponding unicast address. But the anycast address cannot be a source address. In order that a peer be able to identify the connection corresponding to the RST message, it is required that RST message, in such cases, contain Source-Id option.

Another interesting scenario is when the host has several anycast addresses. If an incoming anycast request matches a listening socket with a different anycast bound address, there cannot be another listening anycast socket. Hence a RST is sent with the Source-Id.

The last interesting case is when a SYN with Source-Id option is received and a matching socket is available, but there exists another connection using the same four tuple - either in synchronized state or on a listen queue. If the current segment is delivered, this leads to two connections with the same four tuple. Such a condition is usually checked at the remote host (it allows only one connection to proceed, and resets other), but this can occur for example if remote host crashed and lost state. In this case the incoming segment is silently discarded. One of the two sockets will ultimately be reset by the peer, when loss of synchronization is detected on a retransmit attempt. Then the other connection will proceed.

4.3 UDP Layer

Sending end

An application can specify an anycast address for outgoing datagrams in two different ways, by binding to an anycast address, or by specifying an anycast source address as ancillary data in a call to `sendmsg()` [11].

We provide a flag (`sndany`) to enable/disable sending of Source-Id option. When a socket is bound to an anycast address, value of the flag is tested on each outgoing datagram. A Source-Id option identifying the anycast address is sent only if it is set. If the flag is not set, the effect of a bind to anycast address is to allow reception of incoming datagrams destined to the anycast address, but outgoing datagrams use a unicast address without the option. This handling is similar to that for multicast addresses - a multicast address cannot be used as source address in outgoing packets [4].

When a socket is bound to wildcard address, it can receive datagrams on all local addresses. A server should use the destination address on incoming requests as the source address for outgoing responses. Receiving destination address on incoming datagrams and specifying source address on outgoing datagrams is accomplished using ancillary data to `recvmsg()` and `sendmsg()` respectively [11]. When an anycast address is specified as ancillary data to `sendmsg()` and `sndany` flag is set, a Source-Id option is automatically added. If `sndany` is not set, the `sendmsg()` returns an error.

Receiving end

On receiving a datagram with Source-Id option, demultiplexing uses remote's anycast address instead of the source address. If no match is found, an ICMP port unreachable is sent (with one exception as discussed further). A matched socket can either be unconnected (wildcard remote address in four-tuple) or connected to remote's anycast address. In both cases the datagram is delivered. In the latter case, further flexibility is provided. If `state_req` is true, application desires communication to unique node. So the remote address in four-tuple is set to remote's unicast address, and further communication is carried out with unicast addresses. Note that this transformation will occur as soon as the first datagram is received from peer.

There is a catch here. Remote UDP may continue sending the option in future datagrams for the socket. By the above demultiplexing rule, no PCB will match (usually), because the previously matched PCB changed remote address in four-tuple (automatically). A valid action is to deliver to same socket even after transformation. Hence the anycast address is saved in the PCB (apart from four-tuple), and demultiplexing is modified to allow this. Whenever a `connect()` is issued by user or `state_req` flag changed from false to true, this address is reset.

4.4 Raw Socket

Raw socket provides a raw interface to the IPv6 layer that can be used, for example, by ICMPv6. Any datagram received at the IPv6 layer and destined for the local host, is delivered to all raw sockets in addition to corresponding transport layer (if one exists), if the specified Next Header field matches. All fields in a received IPv6 header and all extension headers are also made available to the application [11]. Hence no change needs to be made for receiving a Source-Id option, but applications are responsible to interpret this appropriately and may need to be modified.

For sending a Source-Id option on outgoing datagrams, handling is similar to the case of UDP, i.e., by

setting `sndany` and either binding to an anycast address or specifying an anycast source address through ancillary data to `sendmsg()`.

4.5 Applications Programming Interface

An `ioctl()` is provided to assign an anycast address to an interface. Input to this routine is an interface index and an IPv6 address. An `ioctl()` is provided to determine if the input address is a local anycast address. Three socket options are provided to get/set values of various flags. These flags control per socket anycast handling: allow sending of a Source-Id option, allow receipt of a packet which is demultiplexed using a Source-Id option and manipulate the `state_req` flag that is specific to UDP.

`getpeername()` returns the remote address member of the four tuple. The remote address may change in some cases of anycast usage. Applications may use `getpeername()` to *notice* a change in the peer's address, e.g., after a `connect()` succeeds in TCP. Applications can also learn a peer's anycast address using ancillary data. UDP returns the anycast address received in the Source-Id option on a per datagram basis in a call to `recvmsg()`. TCP uses a call to `getsockopt()` to receive one or more ancillary data objects in the buffer, for all the optional information it wants to receive. One of the optional information is peer's anycast address that is remembered in the PCB.

5 Conclusions

In this paper we discussed a solution to the problem with host anycast addresses that they cannot be used as source address of datagrams and as destination address in ongoing stateful communication. We specified the requirements at both end-points of a communication. The paper focussed on mechanisms in two popular transport protocols TCP and UDP. API extensions are suggested to provide flexible usage of host anycasting by applications.

This mechanism allows many TCP/UDP applications to work unchanged. Applications that maintain state across multiple TCP connections or on top of UDP, or which use the end-point addresses at the application layer will need to be modified. The solution was implemented in our target operating system and tested in a LAN environment.

5.1 Future Work

More work needs to be done for secure communication using anycast address. The interaction with Authentication Header needs to be provided. Research is required to design a solution for key distribution for an anycast group. Currently, there is no mechanism to determine the required action when a datagram is received for which a corresponding security association does not exist. Research is also required to solve the problem with Internet wide routing of host anycast addresses.

References

[1] R. Atkinson. IP authentication header. RFC: 1826, Aug. 1995.

[2] S. Bellovin. Security problems in the TCP/IP protocol suite. *ACM Computer Communications Review*, 19(2), Mar. 1989.

[3] J. Bound and P. Roque. IPv6 anycasting service: Minimum requirements for end nodes. Internet Draft, June 1996. Work in Progress.

[4] R. Hinden and S. Deering. IP version 6 addressing architecture. Internet Draft, June 1997. Work in Progress.

[5] C. Huitema. *IPv6: The New Internet Protocol*. Prentice Hall, Englewood Cliffs, New Jersey, 1996.

[6] D. Johnson and C. Perkins. Mobility support in IPv6. Internet Draft, Nov. 1996. Work in Progress.

[7] T. Narten, E. Nordmark, and W. Simpson. Neighbor discovery for IP version 6 (IPv6). RFC: 1970, Aug. 1996.

[8] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. RFC: 1546, Nov. 1993.

[9] S. Shah. Support for host anycast, priorities and naming of link-local addresses in IPv6. Master's thesis, IIT, Kanpur, Computer Science & Engg. Dept., Aug. 1997.

[10] R. Stevens. *TCP/IP Illustrated: The Protocols*, volume 1. Addison-Wesley, Reading, MA, USA, 1994.

[11] R. Stevens and M. Thomas. Advanced sockets API for IPv6. Internet Draft, Mar. 1997. Work in Progress.