Anadolu Üniversitesi Bilim ve Teknoloji Dergisi A- Uygulamalı Bilimler ve Mühendislik
**Anadolu University Journal of Science and Technology A- Applied Sciences and Engineering**

**2016 - Volume: 17 Number: 1**
**Page: 139 - 149**
**DOI: 10.18038/btda.30517**
**Received: 28 January 2016 Accepted: 14 March 2016**

# MULTI-LINK SUPPORT FOR IPV6 NETWORKS

**Cüneyt AKINLAR**

Computer Engineering Department, Anadolu University, 2 Eylül Kampusü, 26470, Eskişehir, TURKEY

## ABSTRACT

The common practice for IPv6 network configuration is to assign a unique subnet prefix for each physical link in the network. This process not only demands multiple subnet prefixes, but also requires manual administration and configuration. An alternative configuration method is to bridge all links at the IPv6 layer and assign a single subnet prefix for the entire network, thus the name multi-link subnet. In this paper we address the auto-configuration of a multi-link subnet for IPv6 networks. We first describe how multi-link subnet support can be implemented in a single-router network, and extend the methods to multi-router networks. We believe that the proposed auto-configuration methods complement the standard host auto-configuration protocol, and enable true plug-and-play networking necessary for ubiquitous deployment of IPv6 networks.

**Keywords:** IPv6, Zeroconf Networks, Address Auto-configuration

# IPV6 AĞLARI İÇİN ÇOKLU-BAĞ DESTEĞİ

## ÖZET

IPv6 ağ konfigürasyonu için genel pratik ağ üzerindeki her fiziksel bağ için ayrı bir altağ öneki atamaktır. Bu uygulama hem birden fazla ağ öneki gereksimine sebep olur, hem de elle yönetim ve konfigürasyon gerektirir. Buna alternatif konfigürasyon yöntemi bütün bağları IPv6 seviyesinde bir köprü ile bağlamak ve bu sayede bütün ağ için tek bir ağ öneki ataması yapmaktır. Bu metoda çoklu-bağ öneki ismi verilir. Bu makalede IPv6 ağları için çoklu-bağ öneki oto-konfigürasyonu konusu ele alınmıştır. Öncelikle tek yönlendiricili ağlar için coklu-bağ desteğinin nasıl gerçekleneceği tarif edilmiş, daha sonra bu yöntemin çok yönlendiricili ağlara nasıl uygulanacağı anlatılmıştır. Önerilen oto-konfigürasyon yöntemleri standart alıcı oto-konfigürasyon protokollerini tamamlamakta ve IPv6 ağlarının yaygın kurulum ve kullanımları için gerekli tak-çalıştır metodolojine olanak sağlamaktadır.

**Keywords**: IPv6, Sıfır-konfigürasyon Ağları, Adres oto-konfigürasyonu

## 1. INTRODUCTION

Auto-configuration of IPv6 [1-4] addresses is an important problem especially for home and office networks [12, 13], self-configuring battlefield, ad-hoc and mobile networks having dynamic structures [14-22], and emerging wireless sensor networks [23]. Whether done manually or automatically, the common practice for IPv6 network configuration is to assign unique subnet prefixes, a.k.a., subnets, one for each physical link in the network. As it stands, a different subnet is assigned to each link in the network and the routers advertise the assigned subnets over their directly attached links [1, 2, 25]. The hosts then use the standard stateless address auto-configuration protocol [3, 4] to configure IPv6 addresses from within the advertised IPv6 subnet prefixes.

Alternative to multiple subnet prefixes, one for each link in the network, one can imagine assigning a single subnet prefix for the entire network that covers all links [26]. With this configuration method, not only a single subnet prefix is sufficient for the entire network, but also it allows easy address auto-configuration of the entire network, deeming manual administration unnecessary. Such plug-and-play IPv6 networking is a plus for management and deployment of corporate networks, and is an absolute necessity for ubiquitous deployment of emerging home and SOHO networks, a.k.a, the Internet of Things [12, 13]. Since a single physical link requires a single subnet prefix, one can argue that bridging at the link-layer

* Corresponding Author: cakinlar@anadolu.edu.tr

(L2) should be used to create a single link, but this has two drawbacks: (1) Not all link technologies can be bridged at the link-layer. When two link-layer technologies do not support IEEE 802 addressing or when they have different MTUs, classic L2 bridging fails [26], (2) L2 bridging creates a single link-layer broadcast domain, which may be undesirable for large networks. An alternative strategy would be to bridge several links at layer 3 (L3), i.e., IPv6-layer. Given this multi-link network formed at the IPv6-layer, we assign a single subnet prefix for the entire network, thus the name **multi-link subnet** [26]. With this configuration method, the routers would advertise a single subnet prefix over all links creating a multi-link subnet at the IPv6-layer.

Along with its advantages, multi-link subnet support brings its own challenges. Specifically, packet forwarding within the network must transparently be handled by the routers (just like L2 bridges would do). Hosts must not be aware that they are in a multi-link subnet; for them they are simply part of an IPv6 subnet, and exchange packets using the standard host packet forwarding algorithm.

In this paper we address how multi-link subnet support can be implemented in IPv6 networks. We first consider the simple single-router network, e.g., a home network, and describe 2 different methods by which multi-link subnet support can be realized. Then in section V, we show how the proposed methods can be extended to the more challenging multi-router networks. We believe that the proposed methods are novel, and complement the standard host auto-configuration protocol to allow true plug-and-play IPv6 networking.

## 2. RELATED WORK

Stateless host address configuration in a wired network is an important part of the IPv6 protocol [1-4]. When a host boots, it uses a well-defined procedure to configure its unicast addresses using router discovery and advertisement messages [3, 4]. The details of this procedure is outlined in section III. However, the other components of an IPv6 infrastructure, i.e., the routers, still require manual configuration. Although manual configuration may be necessary for large scale IPv6 networks consisting of tens of hundreds of routers, it is not feasible for small home and office networks [13], self-configuring battlefield networks [14], mobile ad-hoc networks [15-22], and emerging wireless sensor networks [23] for which the entire network infrastructure needs automatic configuration without the intervention of the network administrators.

The authors in [14] consider tactical battlefield networks, and propose methods for nodes to configure themselves with little or no human intervention. They extend the standard DHCP protocol to propose two new protocols: Dynamic Registration and Configuration Protocol (DRCP) to extend DHCP to new domains, and Dynamic Address Allocation Protocol (DAAP) to distribute addresses and other configuration parameters to DRCP servers. Misra et al. [15] propose extensions to DHCP, PPP and Mobile IP protocols to support pervasive network access. Weniger and Zitterbart [16] investigate the applicability of IPv6 stateless host configuration procedure to mobile ad-hoc networks. They propose extensions to the neighbour discovery protocol and a hierarchical leader nodes election mechanism to support host auto-configuration in an ad-hoc network. Park et al. [17] consider address allocation and distribution in ad hoc networks that are connected to the global Internet via some gateways. The proposed protocol shortens the address allocation time and improves duplicate address detection mechanism. Fan and Subramani [18] consider address allocation and duplicate address detection in the case of ad-hoc network partitioning and merging. Ghosh and Datta [19] propose an ID based address configuration method to securely allocate IP addresses to hosts in an ad-hoc network without the need for broadcasting. The proposed algorithm can also handle network partitioning and merging efficiently and securely. Li and Chao [20] address the problem of IPv6 auto-configuration in vehicle ad-hoc networks (VANET), which enables wireless communication between vechicles and the roadside infrastructure for driver assistance and safety. Grazjer et al. [21] propose an extended IPv6 Neighbour Discovery protocol for fast duplicate address discovery in ad-hoc networks. Sahadevaiah et al. [22]

propose an IPv6 address configuration scheme for an ad-hoc network with low latency and communication overhead. Montavont et al. [23] evaluate stateless IPv6 address auto-configuration in the context of the emerging low-power, lossy wireless networks such the Internet of Things, Smart Cities and Smart Homes. They propose optimizations for IPv6 neighbour discovery (ND) procedure. AlSadeh et al. [24] analyse the security and privacy aspects of IPv6 host configuration protocol, and propose modifications to the protocol to make it more reliable while protecting user privacy. Altug and Akinlar [25] propose an algorithm to uniquely auto-configure a subnet for each link of a multi-router IPv6 network. Thaler and Huitema [26] introduce the concept of IPv6 multi-link subnet, but consider it only in the context of a single gateway router. In this paper, we analyse this idea and extend it to complex multi-router IPv6 networks.

## 3. IPV6 UNICAST ADDRESSES AND HOST ADDRESS AUTO-CONFIGURATION

An IPv6 address is 128-bits long, and is represented in what is called a colon-hexadecimal notation [1, 2]. That is, each 16-bit block of an IPv6 address is converted into a 4-digit hexadecimal number and separated by colons. Figure 1 depicts the structures of 2 types of unicast IPv6 addresses [5-7].

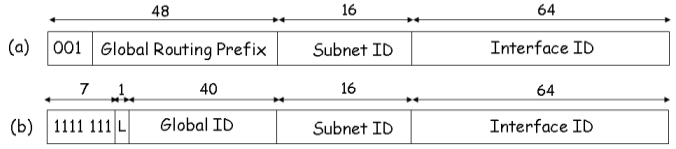



**Figure 1.** Unicast IPv6 address structures:
(a) Provider-based globally routable address, (b) Local unicast address

A global unicast address consists of a 48-bit global routing prefix, a 16-bit subnetid and a 64-bit interface-id. An organization is assigned the global routing prefix; provided by an ISP during initialization using DHCPv6 [8, 9], ICMPv6 [10] or some other mechanism. The 16-bit subnetid is used within a site to identify IPv6 subnets, and is currently assigned manually by the administrators. The interface-id indicates the interface on a specific subnet [5].

A local unicast address structure [6] is similar to a global routable address: The first 7 bits are fixed FC00::/7, followed by a bit named *L* which is set to 1, then followed by a 40-bit global ID. A 16-bit subnetid follows the 48-bit global ID, and the last 64-bits are the interface-id. Local unicast addresses are expected to be globally unique, but are intended to be used and routable within a site.

In the rest of this paper, we would denote a global or local unicast address as "p.s.i", where *p* is the first 48-bit prefix, *s* is the 16-bit subnetid, and *i* is the 64-bit interface-id.

### 3.1. IPv6 Host Auto-Configuration Process

Host auto-configuration process is described in RFC 2462 [3]. An auto-configured IPv6 address can be in one or more of the following states [3, 4]:

**Tentative**: The address is in the process of being verified as unique.
**Valid:** An address for which uniqueness has been verified and from which unicast traffic can be sent and received. The valid state covers both the preferred and deprecated states.
**Preferred:** A node can send and receive unicast traffic to and from the preferred address. The period of time that an address remains in preferred state is determined by the Preferred Lifetime field in a Router Advertisement message.
**Deprecated:** An address that is still valid, but its use is discouraged for new communication. Existing sessions can still use a deprecated address.
**Invalid:** An address for which a node can no longer send or receive unicast traffic.

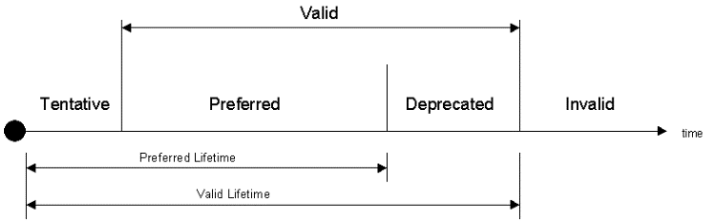



**Figure 2.** The states and lifetimes for an auto-configured IPv6 address

The relationship between different states of an auto-configured address is depicted in Figure 2. An IPv6 host auto-configures its addresses either through (1) **stateless** address configuration, which is solely based on the receipt of Router Advertisements with one or more Prefix Information options, (2) **DHCPv6**, which is used when the Managed Address Configuration flag within a Router Advertisement is set to 1, or (3) using both of the mechanisms. Here are the steps of the IPv6 host auto-configuration process:

**1.** A tentative link-local address is derived based on the link-local prefix FE80::/64 and the 64-bit interface-id, and its uniqueness is tested over the link using Neighbour Solicitation messages.

**2.** After the unique link-local IPv6 address setup is complete, the host sends several (default three) Router Solicitation messages.

**3.** If no Router Advertisement messages are received, then the host uses DHCPv6 to obtain an address.

**4.** If a Router Advertisement message is received then

**4.1.** For each Prefix Information Option with Autonomous flag set to 1, the host derives an address using the prefix.

**4.2.** If Managed Address Configuration flag is set to 1, the host uses DHCPv6 to obtain an address.

## 4. AUTO-CONFIGURING SINGLE-ROUTER IPV6 NETWORKS

A single-router network is one, where a router connects several segments into a star topology as illustrated in Figure 3. A typical example of a single-router IPv6 network is a simple home network, where the router, usually called a home or residential gateway, has several different internal segments such as Ethernet, WiFi (802.11), HomePNA, IEEE 1394, Bluetooth, etc., and also provides broadband Internet connectivity over one of xDSL, cable, or ISDN. In Figure 3 the router connects 3 internal links, L1, L2 and L3 together over its interfaces 1, 2 and 3, and provides Internet connectivity over its interface 4.

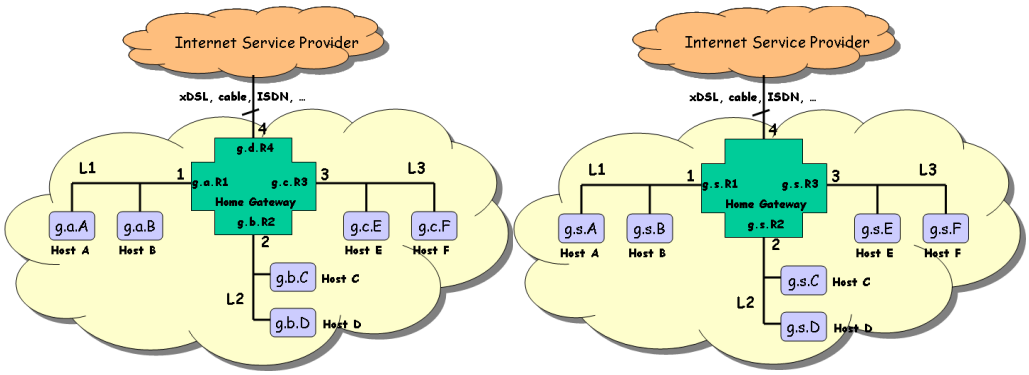



**Figure 3.** (a) An example single-router IPv6 network, e.g., a home network, where the router assigns unique IPv6 subnets g.a::/64, g.b::/64. g.c::/64 to links L1, L2 and L3, (b) The same single-router network, where the router assigns a single IPv6 subnet g.s::/64 to all links, thus creating a multi-link subnet

### 4.1. Unique Subnet over Each Link

One way to auto-configure a single-router IPv6 network is to simply assign unique IPv6 subnets over each of the internal links as illustrated in Figure 3(a). After R obtains the global routing prefix g::/48 from the ISP, it assigns unique 16-bit subnetids, *a*, *b* and *c*, forming prefixes g.a::/64, g.b::/64 and g.c::/64, and advertises these prefixes over L1, L2 and L3 respectively. Hosts then use IPv6 stateless auto-configuration algorithm of section III to get IPv6 addresses. In Figure 3(a), A has configured IPv6 address g.a.A, where *g* is the 48-bit global routing prefix, *a* is the 16-bit router-assigned subnetid for L1 and *A* is the host's 64-bit interface-id.

Packet forwarding in this network is quite easy: To reach a host on the same IPv6 subnet, e.g., A sending a packet to B, A simply sends a Neighbour Solicitation (NS) to B's solicited node multicast address. B will receive this NS, and reply with a Neighbour Advertisement (NA), and A would deliver the packet to B. To reach a host on another IPv6 subnet, e.g., A sending a packet to C, A sends the packet to its default router R, which then forwards the packet to C.

### 4.2. Multi-Link Subnet

Another way to auto-configure a single-router IPv6 network is to simply assign a single IPv6 subnet covering all links, thus the name multi-link subnet. The idea of an IPv6 multi-link subnet was introduced in [2], and is illustrated in Figure 3(b). After obtaining the global routing prefix g::/48, R assigns a single 16-bit subnetid *s* forming the prefix g.s::/64. All hosts in the network then configure IPv6 addresses within this multi-link subnet using the host auto-configuration algorithm of section III.

While this method has the advantage of requiring a single IPv6 subnet prefix for the entire network, it introduces its own problems: How would hosts in different links communicate? Note that hosts must be unaware of the multi-link subnet and work seamlessly.

There are three alternatives:

#### 4.2.1. Off-link model with router acting as a ND-proxy:

In this model the router sets the autonomous address-configuration flag (A) on, and the on-link flag (L) off for the prefix g.s::/64 advertised in RAs. Thus hosts send all packets to the default router R for delivery. It is then R's responsibility to forward packets to the destination hosts. To accomplish packet forwarding, R simply sends NSs on all links, and updates its routing table when it gets an NA. This way R would eventually learn the location of all hosts in the network, and can perform simple packet forwarding based on these host routes.

To make this method concrete, here is the possible set of actions when A sends a packet to B in Figure 3.

| | |
|---|---|
| **(1)** | A delivers the packet to R |
| **(2)** | R sends a NS over each of L1, L2 and L3 |
| **(3)** | B replies with a NA to R |
| **(4)** | R updates its routing table with (g.s.B, Intf 1) |
| **(5)** | R sends the packet to B |
| **(6)** | R sends an ICMPv6 Redirect to A |
| **(7)** | A sends following packets directly to B |

During this process, R also learned how to reach B. So if any other host in the network sends a packet to R for delivery to B, R can immediately forward it to B over L1.

Had A sent the packet to a host on L2 or L3, e.g., A sends a packet to C, then C would have replied in step 3. Thus R would have learned the location of C and would have simply forwarded the packet to C

over L2. Then subsequent packets would be forwarded to C without requiring ND messages.

### 4.2.2. On-link model with router acting as a ND-proxy:

In this model the router sets the autonomous address-configuration flag (A) off, and the on-link flag (L) on for the prefix g.s::/64 advertised in RAs. This means that when a host wants to send a packet to another host on the same IPv6 subnet, it simply starts by sending a NS to the corresponding solicited node multicast address. If the destination host is on the same link, it will reply with a NA, and the packet would get delivered. If the destination host is not on the same link however, R is supposed to receive this NS and respond with a NA. This requires that R knows the IPv6 addresses and locations of all hosts in the network and thus would listen to solicited node multicast addresses for NSs. For example, assuming R knows the IPv6 addresses and locations of all hosts in the network, when A sends an NS for C, R would reply with its own MAC address over L1. A would then deliver the packet to R, which will forward the packet to C. Although this method may work from a theoretical point of view, we are against it for the following reasons:

(1) It is not clear how R would learn the IPv6 addresses and locations of all hosts in the network,
(2) Assuming R has this information, having R listen to solicited node multicast addresses for all hosts on the network over all interfaces is impractical.

### 4.2.3. Off-link model with router running a DHCPv6 server:

Instead of having the router learn the location of a host on-demand as is done with method in section IV.2.1., we propose that R runs a DHCPv6 server, and advertises in RAs that all hosts must configure from this DHCPv6 server by setting the managed address configuration flag on. As hosts configure from the DHCPv6 server, R updates its routing table. Thus R learns that A and B are on L1, C and D are on L2 and E and F are on L3. Since the prefix *g.s::/64* is marked off-link, all packets are sent to R, which would forward packets to their destinations by simply consulting its routing table. Furthermore, if a host moves from one link to another, it would reconfigure from the DHCPv6 server. At that point, R can update its routing table to reflect the move.

## 5. AUTO-CONFIGURING MULTI-ROUTER IPV6 NETWORKS

A multi-router IPv6 network is one where two or more routers connect several segments together. One or more of the routers may also provide Internet connectivity. Any non-trivial network would have multiple routers. Due to multiplicity of link technologies at home, even a home or SOHO network may consist of several routers that connect these links together.

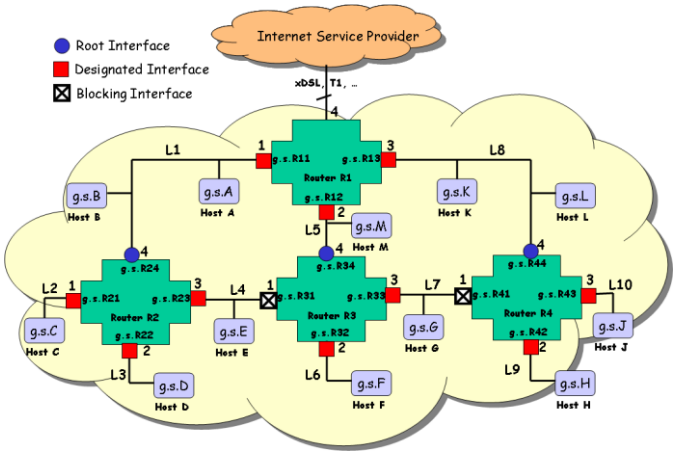



**Figure 4.** A multi-router network with multi-link subnet g.s::/64 configured over the entire network

Figure 4 depicts an example multi-router IPv6 network having 4 routers connecting 10 links. A multi-link subnet g.s::/64 is configured to cover all links in the network. While it is possible to configure a unique IPv6 subnet over each of the links in the network, this requires multiple subnets, and currently requires manual configuration. Instead we propose extending the multi-link subnet support to multi-router networks.

Extending multi-link support to multi-router networks requires special coordination among routers in the network. Below we extend ND-Proxy (refer to section IV.2.1.) and DHCPv6 (refer to section IV.2.3.) methods to multi-router networks.

### 5.1. Off-Link Model with Router Acting as a ND-Proxy

Recall that in this model routers send RAs for prefix g.s::/64 by setting the autonomous address-configuration flag (A) on, and the on-link flag (L) off. Thus hosts send all packets to their default router for delivery. It is then the router's responsibility to forward packets towards their destinations.

Packet delivery is easy when there is a single router in the network, but when there are several routers, special coordination among the routers is required. Specifically, in the single-router scenario the router simply sends NS over all attached links, and either gets a NA from the destination host, or no NS arrives indicating the nonexistence of the destination. In a multi-router scenario however, non-arrival of NS may simply mean that the destination is not located on one the directly-attached links. However, the destination may be located on some other link in the network. As an example, consider A in Figure 4 sending a packet to H. A would deliver the packet to R1, its default router. R1 then sends NS to H's solicited node multicast address over L1, L5 and L8. But no NA would arrive, but R1 cannot conclude the nonexistence of H, as H is located at L9. Instead R1 must send the same NS to all neighbour routers R2, R3 and R4 to make sure that the NS is distributed to all links in the network. Notice that the first NS sent by R1 cannot be received by R2, R3 and R4 since they are not listening to H's solicited node multicast address. Therefore, R1 must specifically send another NS to R2, R3 and R4 asking for H's MAC address.

Special care must be taken when distributing a NS over the network. If each router simply sends the NS to each of its neighbours and they employ the same algorithm, a NS may loop forever if there are loops in the network. To prevent such loops, we propose that the routers run Perlman's spanning tree algorithm [11] to eliminate loops in the network as illustrated in Figure 4.

**Computation of the Spanning Tree:** Here is a brief description of the spanning tree algorithm: Routers first choose a root node, which usually is the node with the smallest identifier, R1 in our example. Then non-root routers mark their interface that provides the shortest path to the root as the root interface. In our example interface 4 of R2, R3 and R4 is their root interface since the root R1 is just one hop away through their interface 4. Then for each link in the network, one of the routers attached to the link becomes the designated router. The designated router for a link is responsible for forwarding packet to/from the link, and is the router that provides the shortest path to the root. In case of ties, the router having the smallest identifier wins. In our example, both R2 and R3 connect L4 in two hops to the root, but R2 is the designated router for L4 since its identifier is smaller. The designated interfaces for all other links are chosen similarly. Routers then mark their interfaces attached to links for which they are the designated router as “designated” interfaces. All other interfaces are marked blocking, meaning that the router would not receive or forward any traffic through that interface.

**Router Auto-Configuration and Packet Forwarding:** Upon startup, all routers compute a spanning tree as described above. We can easily make sure that the edge router, i.e., the router providing Internet connectivity, becomes the root by making its identifier the smallest, (e.g., make the most significant bit of the identifier to denote the provision of Internet connectivity: 0 means edge router, 1 means internal

router. Thus an edge routers will always have the smallest identifier and elected as the root). Upon completion of the spanning tree computation, the routing table of all routers are empty. Routers then assign IPv6 addresses using g.s::/64 prefix to each of their non-blocking interfaces. Routers also advertise g.s::/64 over links for which they are the designated router with off-link flag set. Thus a designated router becomes the default router for the hosts on the link, and hosts always send their packets to this router.

Upon reception of a packet from a host, the router first consults its routing table. If there is an entry for the destination, the packet is forwarded accordingly. If not, the router starts by sending a NS over each of its non-blocking interfaces. If a NA is received, the router simply updates its routing table and forwards the packet. If no NA is received, the NS is sent to the neighbouring routers over the spanning tree edges. Neighbouring routers would apply the same algorithm until the NS reaches all links of the network. When the destination host replies with a NA, routers would forward the NS towards the originating router while updating their routing tables. Next time a NS arrives for the same host, a router can simply respond after consulting its routing table. An example would solidify this discussion. Below we describe the set of actions that the routers take when A sends a packet to H:

| | |
|---|---|
| **(1)** | A delivers the packet to R1 |
| **(2)** | R1 sends a NS over L1, L5 and L8 |
| **(3)** | No NA arrives, so R1 sends a NS to R2, R3 and R4 for H |
| **(4)** | R2 forwards this NS over L2, L3 and L4; R3 over L6 and L7; and R4 over L9 and L10 |
| **(5)** | H replies with a NA to R4. R4 updates its routing table with (g.s.H, Intf2), and sends NA to R1 |
| **(6)** | R1 updates its routing table with (g.s.H, Intf3, NextHop=R4), and forwards the packet to R4 |
| **(7)** | R4 delivers the packet to H |
| **(8)** | Next time R1 directly sends packets destined to H to R4 without sending ND messages |

### 5.2. Off-Link Model with a DHCPv6 Server

Rather than having the routers learn the locations of hosts on-demand by proxying NSs, we can have them learn host locations during host address auto-configuration from a DHCPv6 server. The idea is the following: First have the routers compute a spanning tree as described in section V.1. Then we let the root router run a DHCPv6 server, and all other routers run DHCPv6 proxies. Routers then force hosts to configure IPv6 addresses using DHCPv6 by setting managed address configuration flag in RAs.

Consider a host using DHCPv6 to configure an IPv6 address. It will send out a *dhcpdiscover* message. If the host is on a link that is directly attached to the root router, e.g., host A, then the DHCPv6 server running at R1 will receive this message and address configuration will complete following the usual DHCPv6 protocol message exchanges. At the completion of the address configuration, R1 will know that A is located on L1, and update its routing table accordingly. If the host is not on a link that is directly attached to the root router, e.g., host H, then the *dhcpdiscover* message will be received by a DHCPv6 proxy running at a non-root router, e.g., router R4, which would forward this message up the spanning tree toward the root router. When the message arrives at the DHCPv6 server, an address will be allocated for the host. R1 will update its routing table, and send the *dhcpack* message to the DHCP proxy running at R4. R4 would also update its routing table before delivering the *dhcpack* message to host H. Thus during host address configuration, all routers along the path from the root to the host will have their routing tables updated and know how to reach the host. When all hosts complete address configuration, the root router R1 will have complete host routes. All other routers will have complete host routes for hosts located in the tree for which they are the root.

**Packet Forwarding:** Upon reception of a packet from a host, a router first consults its routing table. If there is an entry for the destination, then the packet is forwarded accordingly. If not, the router simply delivers the packet to the next router along the path to the root. That is, to the designated router attached to the same link that the router's root interface is attached to. This way the packet either makes it to the

root, which knows how to reach all hosts, or an intermediate router knows how to deliver the packet. So the packet eventually makes it to the destination. To make this more concrete, an example would be beneficial: Below we describe the set of actions that the routers take when host H sends a packet to host C:

| | |
|---|---|
| **(1)** | H delivers the packet to R4 |
| **(2)** | R4 does not know how to reach C. So it simply sends the packet upstream to R1 |
| **(3)** | R1 being the root, knows that C is reachable through R2, so it sends the packet to R2 |
| **(4)** | R2 knows that C is on link L2 and delivers the packet to C |

While this method allows correct packet delivery, all packets must travel over the spanning tree edges. This means that a packet may have to take a longer path to reach the destination than the shortest existing path. For example, when H sends a packet to G, the packet follows the path: H-R4-R1-R3-G. Although the shortest path from H to G is H-R4-G, R4 must send the packet to R1 as its interface 1 is a blocking interface due to spanning tree computation. The question that arises is if it is possible to enable packet forwarding without computing a spanning tree. We answer this question in the next section.

### 5.3. Off-Link Model with All Routers running a DHCPv6 Server

Another way to implement a multi-link subnet is to have each router run a DHCPv6 server, and have hosts configure from these DHCPv6 servers. At the completion of host address auto-configuration, routers would only learn the locations of hosts on directly attached links, and can only forward packets among hosts on directly attached links. To enable network-wide packet delivery, the routers then run an intra-domain routing algorithm such as RIP [13] or OSPF [14], and distribute *host routes*. This way all routers would learn how to reach any host in the network through the shortest possible path. Also note that routers do not have to compute a spanning tree.

To make this discussion more concrete, consider the multi-router network in figure 4. Assume A, M, K and L get IPv6 addresses from the DHCPv6 server running at R1; B, C, D and E get IPv6 addresses from the DHCPv6 server running at R2; F and G get IPv6 addresses from the DHCPv6 server running at R2; and H and J IPv6 addresses from the DHCPv6 server running at R4. R1 would have 4 host routes in its routing table and advertise these routes to other routers in the network. All other routers would do the same and advertise their host routes. When the routing protocol messages are received by all routers, all routers will learn how to reach all hosts in the network.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper we described multi-link subnet support for single and multi-router IPv6 networks. We proposed 3 different methods by which multi-link subnet support can be realized in a multi-router network. Proposed methods complement the existing host auto-configuration algorithm [12], and allow true plug-and-play IPv6 network auto-configuration without requiring any manual administration and configuration. We believe that the standardization of such auto-configuration methods is necessary for ubiquitous deployment of future IPv6 networks.

## REFERENCES

[1] Hinden R, Deering S. Internet Protocol Version 6 (IPv6) Addressing Architecture. RFC 3513, 2003.

[2] Davies J. Microsoft Corporation: Introduction to IP Version 6. https://technet.microsoft.com/en-us/library/bb726944.aspx [accessed Jan. 2016]

[3] Thomson S, Narten T. IPv6 Stateless Address Autoconfiguration. RFC 2462, 1998.

[4] Narten T. Neighbor Discovery and Stateless Autoconfiguration in IPv6. IEEE Internet Computing 1999; 3(4): 54-62.

[5] Hinden R, Deering S, Nordmark E. IPv6 Global Unicast Address Format. RFC 3587, 2003.

[6] Hinden R, Haberman B. Unique Local IPv6 Unicast Addresses. RFC 4193, 2005.

[7] Huitema C, Carpenter B. Deprecating Site Local Addresses. RFC 3879, 2004.

[8] Troan O, Droms R. IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6. RFC 3633, 2003.

[9] Miyakawa S, Droms R. Requirements for IPv6 Prefix Delegation. RFC 3769, 2004.

[10] Thulasi A, Raman S. IPv6 Prefix Delegation Using ICMPv6. https://www.ietf.org/archive/id/draft-arunt-prefix-delegation-using-icmpv6-00.txt [accessed Jan. 2016]

[11] Perlman R. An Algorithm for Distributed Computation of a Spanning Tree. ACM SIGCOMM Computer Communications Review 1985; 15(4): 44-53.

[12] Esaki H. A consideration on R&D direction for future internet architecture. International Journal of Communication Systems 2010; 23(6-7): 694-707.

[13] Sinclair B. Finally, IPv6's killer app: The Internet of Things. http://www.zdnet.com/article/finally-ipv6s-killer-app-the-internet-of-things/ [accessed Jan. 2016]

[14] McAuley AJ, Manousakis K. Self-configuring networks, In: IEEE 21st Century Military Communications Conference (MILCOM); 2000; pp. 315-319.

[15] Misra A, Das S, McAuley A, Das SK. Autoconfiguration, registration and mobility management for pervasive computing. IEEE Personal Communication 2001; 8(4): 24-31.

[16] Weniger K, Zitterbart M. IPv6 Autoconfiguration in Large Scale Mobile Ad-Hoc Networks. In: European Wireless conference; 2002; pp. 142-148.

[17] Park Il-Kyun, Kim YH, Park SS. IPv6 address allocation in hybrid mobile ad-hoc networks. IEEE Software Technologies for Future Embedded and Ubiquitous Systems; 2004; pp. 58-62.

[18] Fan Z, Subramani S. An address autoconfiguration protocol for IPv6 hosts in a mobile ad hoc network. Computer Communications 2005; 28(4): 339-350.

[19] Ghosh U, Datta R. A Secure Dynamic IP Configuration Scheme for Mobile Ad Hoc Networks. Ad Hoc Networks 2011; 9: 1327-1342.

[20] Li CS, Chao HC. IPv6 auto-configuration VANET cross layer design based on IEEE 1609. IET Networks 2012; 1-4: 199-206.

[21] Grazjer M, Zernicki T, Glabowski M. ND++-an extended IPv6 Neighbour Discovery protocol for enhanced stateless address autoconfiguration in MANETs. International Journal of Communication Systems 2014; 27(10): 2269-2288.

[22] Sahadevaiah K, Ramakrishnaiah N, Prasad Reddy PVGD. IPv6 Address Auto-Configuration Protocol for Mobile Ad Hoc Networks. Procedia Computer Science 2015; 57: 907-914.

[23] Montavont J, Cobarzan C, Noel T. Theoretical analysis of IPv6 stateless address autoconfiguration in Low-power and Lossy Wireless Networks. In: IEEE Conference on Computing and Communication Technologies – Research, Innovation, and Vision for Future (RVIF); 2015; pp. 198-203.

[24] AlSadeh A, Rafiee H, Meinel C. IPv6 Stateless Address Autoconfiguration: Balancing between Security, Privacy and Usability. Lecture Notes in Computer Science 2013; 7743: 149-161.

[25] Altug RO, Akinlar C. Unique Subnet Auto-Configuration in IPv6 Networks. Lecture Notes in Computer Science 2006; 4268: 108-119.

[26] Thaler D, Huitema C. Multi-link Subnet Support in IPv6. https://tools.ietf.org/id/draft-ietf-ipv6-multilink-subnets-00.txt [accessed Jan. 2016]