T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

IPv6 AĞLAR İÇİN GÜVENLİ DHCP SİSTEM TASARIMI

Gürsoy DURMUŞ

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

GEBZE
2006

T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

IPv6 AĞLAR İÇİN GÜVENLİ DHCP SİSTEM TASARIMI

Gürsoy DURMUŞ

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

TEZ DANIŞMANI
Yrd.Doç.Dr. İbrahim SOĞUKPINAR

GEBZE
2006

ÖZET

TEZ BAŞLIĞI : IPv6 AĞLAR İçin Güvenli DHCP Sistem Tasarımı
YAZAR ADI : Gürsoy DURMUŞ

Internet'in hızlı gelişimi ile IPv4 protokolü, yerini yakın gelecekte IPv6 protokolüne bırakacaktır. IPv6 protokolünün yapısı gereği, adres dağıtımı ve yönetiminin otomatik olarak yapılması gereklidir. Bu ise, dinamik konak yapılandırma protokolü (DHCP) ile mümkündür.

DHCP protokolü, birçok güvenlik açığına sahip olmasına rağmen sağladığı kolaylıklar nedeni ile günümüzde yaygın olarak kullanılmaktadır. Fiziksel güvenliğin sağlandığı yerel alan ağlarında DHCP istemci ve sunumcuları için iç tehditler dışında çok fazla risk olmamasına karşın, özellikle kablosuz ağlarda fiziksel güvenliğin olmaması nedeni ile dış unsurlar çok büyük riskler doğurmaktadır.

Bu çalışmada, IPv6 protokolü için güvenli DHCPv6 sunumcu ve istemci tasarımı yapılarak gerçeklenmiştir. Gerçeklenen sunumcu ve istemci için güvenlik sınamaları yapılarak sonuçlar verilmiştir.

SUMMARY

TITLE OF THESIS: Secure DHCP System Design for IPv6 Networks
AUTHER : Gürsoy DURMUŞ

With the rapid growth of the Internet, IPv4 protocol will leave its duty to IPv6 in the near future. Due to the nature of IPv6, address distribution and administration have to be done automatically. This is possible with the help of Dynamic Host Configuration Protocol (DHCP).

Although the DHCP protocol has a lot of security deficiencies, it is being widely used for its easiness in applications nowadays. Despite the fact that there are not so many risks other than internal threats for the DHCP client and server in local area networks where physical security is provided, external elements constitute great risks in wireless networks (i.e. Physical security is not present).

In this work, the secure DHCPv6 client and server applications were designed and implemented. Security tests were done for the implemented client and server, and the results were given.

TEŞEKKÜR

Çalışmama değerli yorumları ile yön veren, yardımlarını esirgemeyen saygı değer hocam Yrd. Doç. Dr. İbrahim SOĞUKPINAR'a,

Çalışmama teknik açıdan destek olan, değerli meslektaşım Y.Müh. Tomasz MRUGALSKI'ye (Gdansk Teknik Üniversitesi, Polonya) ve Yrd. Doç.Dr. Turgay KORKMAZ'a (Texas Üniversitesi, ABD),

Çalışmamı manen destekleyen, Prof. Dr. Ramazan KORKMAZ'a (Doğu Akdeniz Üniversitesi, KKTC), Yrd. Doç.Dr. Mitat DURMUŞ'a (Kafkas Üniversitesi) ve Yrd. Doç.Dr. Turan KARACA'ya (Yüzüncü Yıl Üniversitesi),

Emeğinin hakkını ödeyemeyeceğim sevgili anneme, babama ve eşime teşekkürü bir borç bilirim.

Saygılarımla...

İÇİNDEKİLER DİZİNİ

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1 GİRİŞ

Bilindiği üzere, Internet üzerinde kullanılan iletişim protokolü TCP/IP'dir. Ağa bağlanacak her bir konağın, diğer konaklarla iletişim kurabilmesi ve ağ hizmetlerinden yararlanabilmesi konağın ağ ayarlarına uygun olarak yapılandırılmasıyla mümkündür.

IPv4 ağlarda konak yapılandırılması, elle veya otomatik olarak BOOTP [Croft and Gilmore, 1985] veya DHCP [Droms, 1993; Droms, 1997] protokolleri üzerinden yapılabilmektedir. BOOTP protokolü sayesinde konaklar için yalnızca IP adresi yapılandırması yapılabilirken, BOOTP protokolünün geliştirilmiş hali olan DHCP protokolü ile IP adresi ve ağ konfigürasyonlarının yapılandırılması dinamik olarak yapılabilmektedir [Comer, 2002]. DHCP protokolü sayesinde, IP adresinin atanması ve ağ servis bilgilerinin gönderilmesi hem otomatikleştirilmiş hem de merkezi olarak yönetimi sağlanmıştır.

IPv4 üzerinde yapılan çalışmalar sonucunda, gerek güvenlik eksikleri gerekse adres uzayının yetersiz kalması nedeniyle tasarım çalışmaları yapılan IPv6'nın gelecekte yeni Internet iletişim protokolü olacağı öngörülmektedir [Deering and Hinden, 1995]. IPv6'da konak yapılandırma işleminin otomatik şekilde DHCP sunumcular tarafından yapılması kaçınılmazdır [Annala, 2004]. Bu nedenle DHCPv6 istemci ve sunumcu tasarımı için standart özellikler değişik dokümanlarda tanımlanmıştır [Droms at al, 2003; Droms and Arbaugh, 2001]. Bu dokümanlara bağlı olarak, farklı tasarım ve gerçekleme çalışmaları yapılmıştır.

DHCPv6 protokolü, IETF bünyesinde geliştirilen ve en çok değişikliğe uğrayan protokollerden biridir. İlk taslaklar, genelde DHCPv4 protokolü temel alınarak hazırlanmış olmasına karşın gelinen son noktada alınan kararlar iki protokol arasında büyük farklılıklar doğurmuştur. Bu durum, taslakları referans alarak geliştirilen DHCPv6 uygulamalarının geçerliliğini yitirmesine, dolayısıyla bu alanda geliştirilen uygulama sayısının azalmasına neden olmuştur [Perkins and Bound, 1998].

Son yıllarda, DHCPv6 için geliştirilen değişik sunumcu ve istemciler mevcuttur [KAME Project; Mrugalski, 2005]. Bu tasarımlarda her ne kadar IPv6'da dinamik adres dağıtımı için çözümler öngörülmüş ise de, istemci-sunumcu doğrulama mekanizmalarının olmaması nedeniyle, DHCP saldırılarına karşı güvenlik zafiyetlerinin olduğu bilinmektedir.

Bu çalışmada önerilen ve geliştirilen güvenli DHCPv6 sistemi, istemci ve sunumcu uygulamalarını içermektedir. Sunumcu ve istemci arasındaki mesajlaşmanın güvenli bir şekilde sağlanması amaçlandığı için, genellikle güvenlik mekanizması üzerinde durulmuştur. Kısmen, RFC-3315 [Droms at al, 2003], RFC-3118 [Droms and Arbaugh, 2001] ve RFC-3646 [Droms, 2003] uyumludur. İstemci ve sunumcu uygulamaları, IPv6 uyumlu Windows XP ve Windows 2003 Server işletim sistemlerinde çalışabilmektedir. Sistemde, istemci güvenilir sunumculardan IP adresi ve ağ konfigürasyon bilgilerini almakta, sunumcu ise güvenilir istemcilere hizmet vermektedir. Bu sayede DHCP saldırılarına karşı korunaklıdır. Sistemde, RFC-3118'de [Droms and Arbaugh, 2001] tanımlanan ve RFC-3315'te [Droms at al, 2003] güncellenen *"gecikmeli doğrulama yöntemi"* güvenlik mekanizması olarak gerçeklenmiştir. Tasarlanan güvenli DHCPv6 sunumcu ve istemcisi programlanarak gerçeklenmiş, fonksiyonel testleri yapılmış ve güvenlik özellikleri sınanarak bulunan sonuçlar tezde verilmiştir.

2. bölümde, DHCPv6 protokolünün çalışma mantığı, mevcut güvenlik açıkları ve bu konuda yapılan çalışmalar hakkında bilgi verilmiştir. 3. bölümde, tasarlanan ve gerçeklenen güvenli DHCP sistemi sunulmuş, istemci ve sunumcu uygulamaların özellikleri açıklanmıştır. 4. bölümde, gerçeklenen sistem için yapılan uyumluluk, performans ve güvenlik testlerine ilişkin sonuçlar sunulmuştur. Son bölüm, sonuç ve öneriler bölümü olarak düzenlenmiştir.

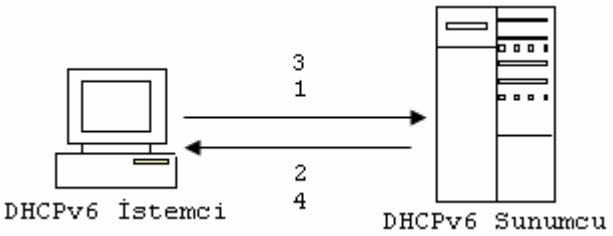
2 DHCPV6 VE IPV6'DA GEREKLİLİĞİ

IP protokolündeki sürüm yenileme çalışmaları sonucu, IPv4 için geliştirilen DHCP protokolünde [Droms, 1993; Droms, 1997] de değişiklikler yapılmış ve IPv6 ağlarda kullanılmak üzere DHCPv6 protokolü [Droms at al, 2003] geliştirilmiştir.

IPv6'da [Deering and Hinden, 1995] otomatik IP adres atama işlemi iki farklı şekilde sağlanmaktadır. Birinci yöntemde, her istemci ağdaki yönlendiricilerden gelen ağ adresi bilgisini alarak, MAC adresi ile harmanlayıp tekil IP adresi elde eder (Stateless address auto-configuration) [Thomson and Narten,1998]. İkinci yöntem ise DHCP kullanımıdır (Statefull address auto-configuration). Birinci yöntemde, istemciler her ne kadar IP adreslerini otomatik olarak elde etseler de, diğer ağ konfigürasyonlarını (DNS vb.) elde edemedikleri için DHCP kullanımına yine ihtiyaç duyarlar.

2.1 DHCPv6 Nasıl Çalışır?

DHCP sistemi, istemci, sunumcu ve ajan olmak üzere üç öğeden oluşur. İstemci, IP adresi ve diğer konfigürasyon bilgilerini kendine verebilecek sunumcuyu bulmak için, sunumcuların üyesi olduğu çoklu yayın adresine (FF01::1:2) mesaj (Şekil 2.1, 1.SOLICIT) gönderir. Ağda ilgili adres ve portları dinleyen sunumcular, istemciye kendilerini ve önceliklerini bildirmek amacıyla bir mesaj (Şekil 2.1, 2.ADVERTISE) gönderirler. Tanıtım mesajlarını alan istemci, en yüksek önceliğe sahip sunumcuyu seçerek, IP adresi ve diğer konfigürasyon bilgilerini almak istediklerini belirten bir mesaj (Şekil 2.1, 3.REQUEST) gönderir. Gelen bu mesajı alan sunumcu, istemciye içinde IP adresi ve diğer ağ konfigürasyon bilgilerini içeren mesaj (Şekil 2.1, 4.REPLY) göndererek istemcinin ayarlarını tamamlamasını sağlar.

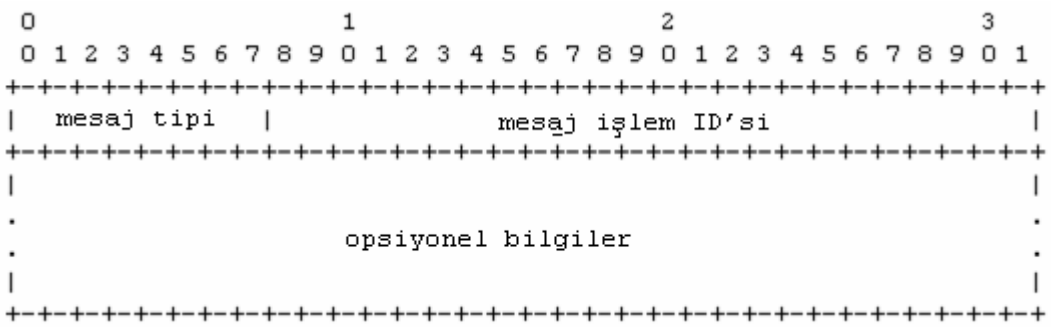


Şekil 2.1 DHCPv6 işleyişi (istemci, sunumcu)

İstemci ve sunumcunun farklı alt ağlarda olmaları durumunda, istemci ve sunumcu mesajları ajanlar aracılığı ile iletilir.

2.2 DHCPv6'da Mesaj Yapıları

DHCPv6 protokolünde istemci ve sunumcu mesajlarının hepsi ortak bir veri yapısına göre oluşturulur. Bu durum, istemci-sunumcu mesajlarının kodlanmasını kolaylaştırdığı gibi, nesneye yönelik tasarımı da mümkün kılmaktadır. Şekil 2.2'de istemci-sunumcu mesajlarının ortak yapısı verilmiştir.



Şekil 2.2 DHCPv6 mesajlarının genel yapısı [Droms at al, 2003]

2.3 DHCP Protokolünde Güvenlik Açıkları

DHCP protokolü, UDP ve IP protokolünü kullandığı için her iki alt protokolün güvenlik açıklarının yanı sıra, tasarımında yer alan bazı güvenlik açıklarına da sahiptir. DHCP protokolünde, mesaj içeriğinin gizlilik derecesi olmadığı kabul edildiğinden mesajlaşmada herhangi bir şifreleme, doğrulama veya yetkilendirme yöntemi kullanılmamaktadır. Bu durum, istemcinin sunumcuyu doğrulayamaması, sunumcunun istemciyi doğrulayamaması ve DHCP mesaj içeriğinin doğrulanamaması gibi protokol tabanlı güvenlik açıklarını yaratmakta; istemci, sunumcu ve ajan için aşağıda bahsedilen riskleri doğurmaktadır [Hibbs at al, 2003].

Günümüzde DHCP sistemleri ile diğer güvenlik sistemleri birlikte kullanılarak ağ güvenliği daha üst seviyelere çıkarılmaya çalışılmaktadır. Örneğin, anti virüs uygulamalarının DHCP sistemleri ile birlikte kullanılması durumunda, virüslerin

mobil cihazlardan ağa geçmeleri engellenmeye çalışılmaktadır [Tanaka at al, 2005]. Bu ve benzer yaklaşımların başarılı olabilmesi DHCP sistemlerinin güvenliği ile orantılıdır.

2.3.1 DHCP İstemcisi için Mevcut Tehditler

Bir DHCP istemcisi için, art niyetli DHCP sunumcular, DHCP mesajlarına müdahale edilmesi ve hatalı / yanlışlıkla hizmete sunulan DHCP sunumcular potansiyel tehdit unsurlarındandır.

Bir DHCP istemcisi için en büyük tehdit, art niyetli konfigüre edilmiş DHCP sunumculardır. Bu yolla bir ağdaki istemcileri kendi amaçları doğrultusunda konfigüre edecek bir saldırgan, aşağıdaki senaryoları gerçekleştirebilir:

- İstemciye, sistem yöneticisinin tanımladığı IP adresinden farklı IP adresi vererek ağ hizmetlerinden yararlanmasını engelleyebilir.
- İstemciye, geçerli bir IP adresi ile birlikte farklı bir ağ geçidi adresi vererek, istemciden gönderilen bütün paketleri dinleyebilir. Saldırgan, tanımladığı ağ geçidi konfigürasyonunu ağa uygun yaparsa, istemci ağdan hizmet almaya devam eder ve bir saldırıya uğradığının da farkına varmaz. İstemciden gönderilen paketleri dinleyen saldırgan, istemci ve kullanıcısına ait gizli bilgileri elde edebilir.
- İstemciye, geçerli bir IP adresi ile birlikte farklı bir DNS adresi vererek istemcinin DNS sorguları için saldırganın tanımladığı değerler döndürülür. Bu durumda, örneğin web üzerinden e-postalarını okumak isteyen bir kullanıcıyı, kendi DNS kayıtlarında tanımladığı, arayüzü gerçeği ile birebir aynı olan başka bir sunumcuya yönlendirmesi durumunda kullanıcının e-posta şifresini elde edebilir.
- İstemciden gelen IP adresi isteklerine hızla cevap veren bir DHCP sunumcusu, bütün DHCP isteklerini üzerine toplar. Temel mesajlaşma adımlarını yerine getirerek işleme devam eder. En son aşamada, IP adresinin sunumcudan tahsis edilmesi isteğine cevap vermeyen sunumcu istemciyi oyalamış olur. İstemci, zaman aşımına uğrayan isteği için IP isteğini yeniler.

Eğer istemci kodlamasında, zaman aşımına neden olan sunumcularla iletişim engellenmiyorsa, bu yöntemle saldırgan, istemcinin IP adresi almasına mani olarak, ağ erişimini engelleyebilir.

Mevcut DHCP sistemlerinde, DHCP istemcileri ve DHCP sunumcuları için yetki kontrolü yapılmamaktadır. Bu durumda, sistem yöneticileri belli aralıklarla, ağlarında hizmet veren DHCP sunumcuları taramalı ve yetkilerini kontrol etmelidir.

DHCP mesaj içeriklerine müdahale yöntemini kullanan bir saldırgan aşağıdaki senaryoları gerçekleştirebilir:

- İstemcinin hatalı veya saldırganın çıkarlarına göre konfigüre edilmesini sağlayabilir.
- İstemcisinin konfigürasyonuna engel olarak, istemcinin ağ hizmetinden yararlanmasını engelleyebilir.

Mevcut DHCP sistemlerinde, DHCP mesajlarının doğrulaması yapılmamaktadır. Bu tehditlere karşı sistem yöneticisi, mümkünse DNS ayarlarını elle yapmalıdır.

Art niyet güdülmeden, yanlışlıkla hizmet veren DHCP sunumcular da diğer bir tehdit unsurudur. Genellikle sistem yöneticilerinin, hatalı/gereksiz sistem kurmalarından kaynaklanır. İstem dışı hizmet veren bir DHCP sunumcu, istemcilerin hatalı IP adresi almalarına ve ağ hizmetinden yararlanamamalarına neden olur. Geliştirilen bazı DHCP sunumcularda, hizmet öncesi yetki ve konfigürasyon kontrolü işlemlerini sistem yöneticisinin yürütmesini zorunlu hale getirilerek bu gibi durumların önüne geçilmeye çalışılmıştır.

2.3.2 DHCP Sunumcusu İçin Mevcut Tehditler

Bir DHCP sunumcu için en büyük tehdit, art niyetli istemcilerdir. Art niyetli bir DHCP istemcisi, DHCP sunumcusundan sürekli IP adresi talebinde bulunarak sunumcu için tanımlı IP adres havuzunu boşaltabilir, diğer istemcilerin IP adresi almalarına engel olabilir.

- Sunumcu ile sürekli mesajlaşarak sunumcu performansını düşürebilir.
- Sunumcuya geçersiz DHCPRELEASE mesajları göndererek kullanımda olan IP adreslerinin IP adres havuzuna atılmasına neden olabilir. Sunumcunun bu adresi daha sonra farklı istemcilere iletmesi durumunda mesaj tekrarlama dolayısıyla istemci ve sunumcu tarafında performans düşüşüne neden olabilir.

Ayrıca, DHCP sunumcularının hizmet vereceği istemcileri doğrulayamamaları, yetkisiz istemcilerin hizmet almalarına ve ağ kaynaklarından yararlanmalarına olanak sağlamaktadır. Bu tür istemci sayının çoğalması, sunumcunun geçerli istemcilere verebileceği IP adreslerinin tükenmesine neden olabilir.

2.3.3 DHCP Ajanı İçin Mevcut Tehditler

Bir DHCP sisteminde, DHCP ajanı, istemci ile sunumcunun farklı alt ağlarda olması durumunda, istemci ve sunumcu arasındaki mesajlaşmayı sağladığı gibi, sunumcunun istemciye hangi IP adres bloğundan adres tahsisi yapacağını, hangi konfigürasyon bilgilerini göndermesi gerektiğini bildirir [Patrick, 2001].

DHCP ajanı ve DHCP sunumcusu arasında gönderilen DHCP mesajlarına müdahale edilmesi (kesilmesi, bozulması, değiştirilmesi) istemci ve sunumcu tarafında, aşağıda listelenen istenmeyen durumlara neden olabilir:

- İstemci, IP adresi ve konfigürasyon bilgilerini alamayabilir.
- İstemci yanlış konfigüre edilebilir.
- Sunumcunun IP adres kaynaklarını hatalı kullanmasına neden olunabilir.

DHCP ajanı ve sunumcuları sabit IP adreslerine sahip oldukları için, ajan-sunumcu mesajlaşmasının güvenliğini sağlamada IPSec kullanımı önerilmektedir [Annala, 2004]. Bu sayede DHCP mesajlarına dışardan müdahale engellenebilir ve ajanların belli sunumcularla iletişim kurmaları sağlanabilir.

2.4 DHCP Protokolü İçin Geliştirilen Güvenlik Yöntemleri

DHCP protokolünün günümüzde yaygın kullanılması ve kablosuz ağların yaygınlaşması nedeni ile DHCP protokolünde güvenlik konusu önem kazanmıştır. DHCP protokolüne ait mevcut güvenlik açıklarını kapatmak amacıyla birçok güvenlik yöntemi geliştirilmiştir.

2.4.1 Anahtar Doğrulama

Anahtar doğrulama (token authentication) yöntemi, RFC-3118'de tanımlı olan ilk güvenlik yöntemidir [Droms and Arbaugh, 2001]. Bu yöntemde, istemci ve sunumcu tarafında tanımlı olan ortak anahtar, düz metin olarak DHCP mesaj doğrulama opsiyonel bilgisine eklenerek gönderilir. Mesajı alan karşı birim, gelen anahtar ile kendi anahtarını karşılaştırır. Anahtarların uyuşması durumunda, gelen mesaj işleme alınır. Bu sayede, istemci sunumcuyu, sunumcu da istemciyi doğrular. Bu yöntem, mesaj içeriğinin doğrulanmasını sağlamaz.

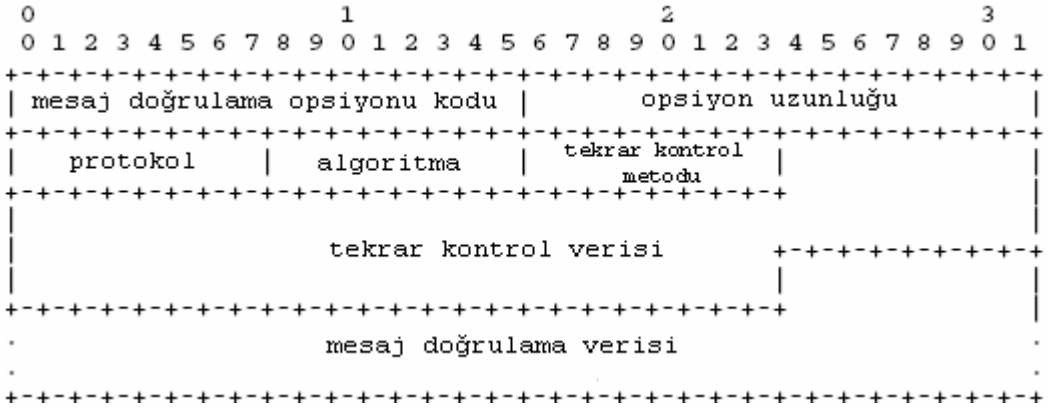
Anahtarların, düz metin olarak gönderilmesi sistemde güvenlik açıkları yaratır. Yanlışlıkla hizmete sunulan DHCP sunumcularından, istemcilerin etkilenmemesini sağlayan basit bir güvenlik mekanizmasıdır.

2.4.2 Gecikmeli Doğrulama

Gecikmeli doğrulama (delayed authentication) yöntemi, RFC-3118'de tanımlı olan ikinci güvenlik yöntemidir [Droms and Arbaugh, 2001]. RFC-3315'te, veri yapısı DHCPv6 protokolü için güncellenmiştir. Diğer güvenlik yöntemleri için alt yapı teşkil eder.

Bu yöntemde, istemci ve sunumcu daha önceden belirlenen anahtarlara sahiptirler. İstemci ve sunumcu birbiri ile mesajlaşırken bu anahtarları birbirlerine iletmezler. Anahtarlar, mesaj özetini şifrelemede kullanılır.

Bu yöntemde kullanılan DHCP mesaj doğrulama opsiyonel bilgisine ait veri yapısı Şekil 2.3'te verilmiştir.

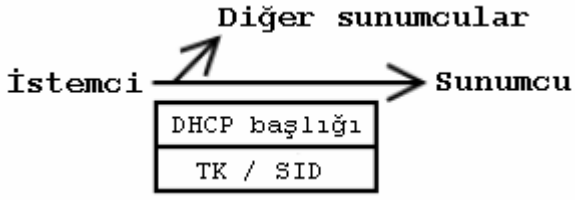


Şekil 2.3 Mesaj doğrulama opsiyonel bilgisi veri yapısı [Droms at al, 2003]

Bu yöntemde, uç birim (istemci veya sunumcu) bir mesaj göndermeden önce, mesajın özet bilgisini istemcinin anahtarı ile şifreledikten sonra mesaj doğrulama opsiyonel bilgisi içinde mesaja ek opsiyon olarak ekleyip gönderir. Sadece, istemci SOLICIT mesajı gönderirken tekrar kontrol bilgisini mesaj doğrulama bilgisine ekleyip göndererek sunumculara mesaj doğrulaması yapacağını bildirir. Yine, alınan bir mesajın doğruluğu, mesaj özet bilgisinin istemci anahtarı ile şifrelendikten sonra mesaj doğrulama opsiyonel bilgisi ile mukayese edilerek kontrol edilir. Değerlerin uyuşması durumunda mesaj geçerli kabul edilir.

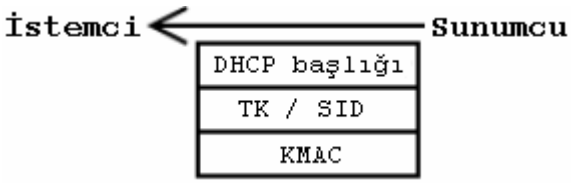
Sistemin çalışma mantığını anlatmadan önce, kullanılan kısaltmaları tanımlayalım.

K : İstemci ve sunucu tarafından bilinen, istemci anahtarı.
SID : İstemci için tanımlı olan K'yı bulmada kullanılan istemci ID'si.(Secret ID)
KMAC : DHCP mesajına ait özet bilginin K ile şifrelenmiş hali
TK : Mesaj tekrarını önlemek için kullanılan tekrar kontrol verisi.



Şekil 2.4 Gecikmeli doğrulama yöntemi, 1.adım [Glazer at al, 2003]

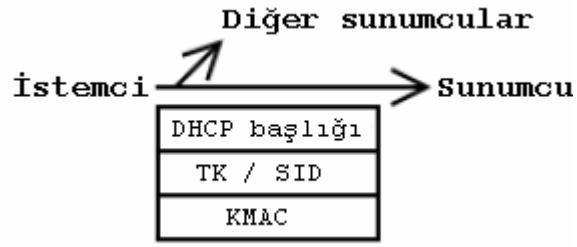
İstemci, SOLICIT mesajına doğrulama opsiyonel bilgisini ekleyerek gönderir.



Şekil 2.5 Gecikmeli doğrulama yöntemi, 2. adım [Glazer at al, 2003]

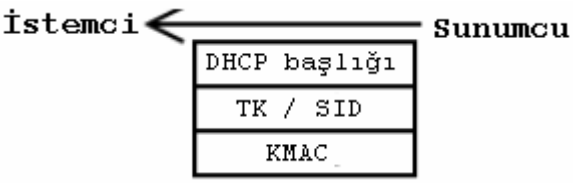
Sunumcu, gelen SOLICIT mesajından istemcinin doğrulama isteğini anlar ve istemci için veritabanında tanımlı olan anahtar ile ADVERTISE mesajının özet bilgisini şifreleyip mesaja ekler.

ADVERTISE mesajını alan istemci, mesajın özet bilgisini, anahtarı ile şifreledikten sonra, ADVERTISE mesajı içersindeki doğrulama opsiyonel bilgisi ile kontrol eder. Bilgilerin uyuşması durumunda istemci, sunumcunun güvenilir olduğunu anlar. Bu aşamada istemci sunumcuyu doğrulamıştır, fakat sunumcu istemciyi henüz doğrulamamıştır.



Şekil 2.6 Gecikmeli doğrulama yöntemi, 3. adım [Glazer at al, 2003]

İstemci, REQUEST mesajına, mesaj özet bilgisinin anahtar ile şifrelenmiş halini ekleyerek gönderir. Gelen REQUEST mesajını alan sunumcu, istemci için elde ettiği anahtar ile mesaj özet bilgisini şifreleyip mesaj doğrulama opsiyonel bilgisi ile karşılaştırır. İçeriklerin uyuşması durumunda sunumcu, REQUEST isteğinin geçerli bir istemciden geldiğini anlar. Bu aşamada sunumcu da istemciyi doğrulamıştır.



Şekil 2.7 Gecikmeli doğrulama yöntemi, 4. adım [Glazer at al, 2003]

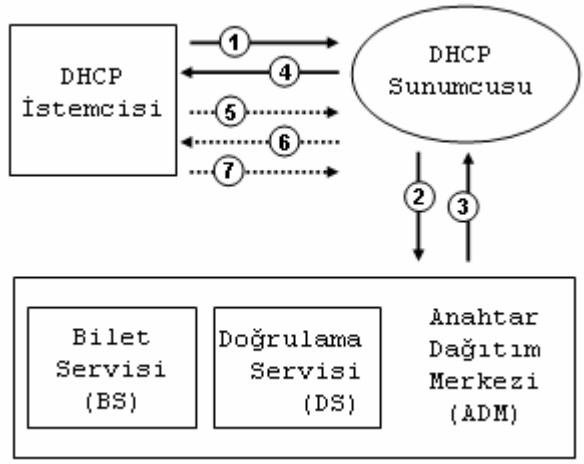
Sunumcu REPLY mesajını gönderir. İstemci yine aynı şekilde REPLY mesajının doğruluğunu kontrol ettikten sonra IP ayarlarını tamamlar.

Yöntem, istemci ve sunumcunun birbirlerini doğrulamasını sağladığı gibi, geçerli mesajların sonradan tekrar gönderilmesi ile yapılacak saldırıları da engellemektedir.

Yöntemin kullanımındaki tek sıkıntı, istemci ve sunumcuya önceden ortak anahtarların dağıtılmasıdır.

2.4.3 Kerberos-V ile Doğrulama

Kerberos-V alt yapısını kullanarak, DHCP istemcilerinin ve mesaj içeriklerinin doğrulanması için geliştirilmiş bir yöntemdir [Hornstein at al, 2000].



Şekil 2.8 Kerberos-V ile DHCP mesajlarını doğrulama [Hornstein at al, 2000]

Bu yöntemde, istemci SOLICIT mesajına, sunumcunun istemci adına Kerberos sunumcusundan oturum bileti alabilmesi için tamamlanmamış bir AS_REQ (authentication service request) paketini ekler (Şekil 2.8, 1. adım). Sunumcu, gelen mesajdan aldığı AS_REQ paketine istemci için verebileceği IP adresini ekleyip istemcinin ADM'sine (Anahtar Dağıtım Merkezi-Key Distribution Center) göndererek istemci adına TGT (Ticket-Granting Ticket) almaya çalışır (Şekil 2.8, 2. adım). ADM, kendine yapılan AS_REQ isteğine karşılık AS_REP (authentication service response) paketini DHCP sunumcusuna gönderir (Şekil 2.8, 3. adım). DHCP

sunumcusu gelen AS_REP paketinden TGT bilgisini çekip ADVERTISE mesajına ek bilgi olarak ekledikten sonra istemciye gönderir (Şekil 2.8, 4. adım). İstemci ve sunumcu elde ettikleri TGT paketlerinden oturum anahtarını (session key) kullanarak DHCP mesajları için mesaj doğrulama verilerini oluştururlar ve bu anahtarı kullanarak diğer mesajların doğruluğunu kontrol ederler (Şekil 2.8, 5., 6., 7. adımlar).

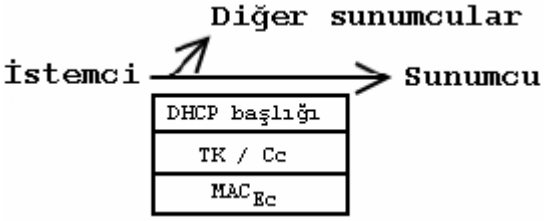
Kerberos sunumcuda istemcinin önceden tanımlanması ve mevcut DHCP mesajlaşmasına ilave olarak DHCP sunumcu ile Kerberos sunumcu arasında fazladan mesajlaşma yapılaması yöntemin en önemli dezavantajlarındandır.

2.4.4 Sertifika Bazlı Doğrulama

Sertifika bazlı doğrulama (certification-based authentication) yönteminde gecikmeli doğrulama yöntemini referans alınmış, anahtar yönetiminin kolay olması, istemci ve sunumcunun birbirlerini doğrulaması, farklı ağlarda en az anahtar ile düzenli çalışabilme ve en az gecikmeli doğrulama yönteminin sağladığı güvenlik kadar güvenlik hedeflenmiştir [Glazer at al, 2003].

Yöntem için, daha önceden geliştirilen güvenlik mekanizmaları incelenmiş ve sonuç olarak X509 sertifika kullanımına karar kılınmıştır. Yöntemde kullanılan kısaltmalar verildikten sonra genel işleyişi DHCPv6 protokolüne göre uyarlanarak aşağıda açıklanmıştır.

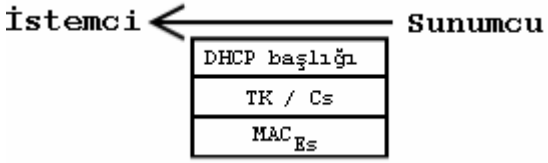
Lc : İstemcinin güvendiği yetkili makam listesi
Kc : İstemcinin açık anahtarı
Ec : İstemcinin gizli anahtarı
Cc : İçinde Kc'yi de içeren sertifika
Ls : Sunumcunun güvendiği yetkili makam listesi
Ks : Sunumcunun açık anahtarı
Es : Sunumcunun gizli anahtarı
Cs : İçinde Ks'yi de içeren sertifika
Tca : İstemci ve sunumcunun güvendiği ortak yetkili makamlar ($Tca = Lc \cap Ls$)



Şekil 2.9 Sertifika bazlı doğrulama-1 [Glazer at al, 2003]

İstemci, SOLICIT mesajına, sertifikasını ve mesajın özet bilgisinin şifrelenmiş halini ekleyerek yayımlar. SOLICIT mesajını alan sunumcular, istemcinin sertifikasını güvenilir makamlara sorarak geçerli olup olmadığını tespit ederler.

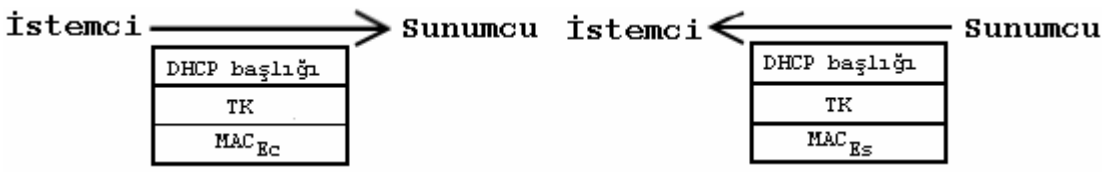
Sunumcu, sertifikanın geçerliliği onaylandıktan sonra, sertifika içerisindeki istemcinin açık anahtarını kullanarak mesajın şifreli özet bilgisini çözer ve hesapladığı mesaj özet değeri ile karşılaştırarak mesajın doğruluğunu kontrol eder.



Şekil 2.10 Sertifika bazlı doğrulama-2 [Glazer at al, 2003]

Sunumcu, ADVERTISE mesajına, kendi sertifikasını ve mesajın özet bilgisinin şifrelenmiş halini ekleyerek istemciye gönderir.

İstemci gelen ADVERTISE mesajındaki sunumcu sertifikasının geçerliliğini sertifika makamlarından onaylatır. Daha sonra, sertifika içerisindeki sunumcunun açık anahtarını kullanarak mesajın şifreli özet bilgisini çözer ve hesapladığı mesaj özet değeri ile karşılaştırarak mesajın doğruluğunu kontrol eder. Bu aşamada istemci ve sunumcu birbirlerinin genel anahtarlarını kaydederek, daha sonraki mesajlarda sertifika göndermekten kurtulurlar.



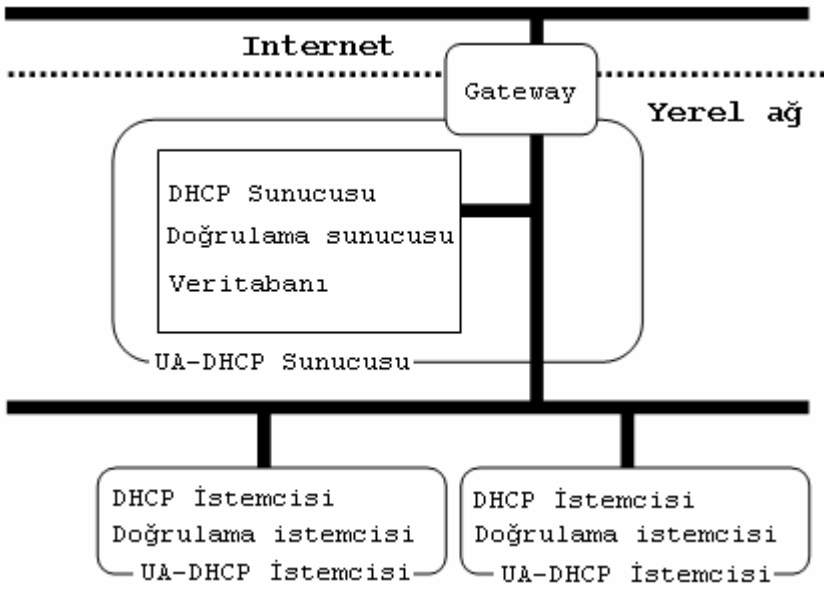
Şekil 2.11 Sertifika bazlı doğrulama-3 [Glazer at al, 2003]

Bu aşamadan sonraki mesajlaşmalarda, istemci ve sunumcu gönderecekleri mesajlara, mesaj özetinin gizli anahtar ile şifrelenmiş halini ekleyerek karşı tarafa gönderirler. Bu bilginin karşı tarafta analiz edilmesi ile mesaj içeriği ve gönderen doğrulanmış olur.

Bu yöntemin dezavantajları, istemcinin IP konfigürasyonu öncesinde geçerli bir sertifikaya sahip olması ön koşulu, farklı ağlarda farklı sertifikalara ihtiyaç duyulması ve süresi dolan sertifikaların yenilenmesi olarak sıralanabilir.

2.4.5 Kullanıcı Doğrulama Yöntemi

Mevcut DHCP sistemlerinde, DHCP hizmetinden yararlanmak isteyen istemciler MAC adresleri ile sisteme tanımlanarak erişim sınırlandırılabilmektedir. Oysa MAC adresini, izin verilen MAC adreslerinden biriymiş gibi gösteren bir istemci, DCHP sunumcusundan hizmet alabilir. Benzer durumların önüne geçmek için geliştirilen bu yöntem [Komori and Saito, 2002] aslında DHCP güvenliğinden ziyade ağ güvenliğini sağlamaktadır.



Şekil 2.12 Kullanıcı doğrulama yöntemi [Komori and Saito, 2002]

Ağdaki herhangi bir konak üzerinden ağa oturum açan kullanıcı, DHCP sunumcudan gerekli ağ parametrelerini alır. Ancak, Gateway yönlendirme tablosunda kendi IP adresi olmadığı için dış ağa (Internet'e) bağlanamaz. İstemci

üzerinde oturum açıldıktan sonra, sistemin bir bileşeni olan kullanıcı doğrulama hizmeti devreye girerek, kullanıcı adı ve şifre sorulur. Girilen bilgiler, doğrulama sunumcusunda kontrol edildikten sonra kullanıcının IP adresi Gateway yönlendirme tablosuna eklenerek kullanıcıcının dış ağ ile iletişimi aktif hale getirilir. Bu arada IP kullanım süresinin artırımı için DHCP sunumcuya bir istekte bulunularak istemcinin IP ayarları yeniden konfigüre edilir.

2.4.6 Güvenlik Yöntemlerinin Mukayesesi

DHCP protokolünün güvenliği için geliştirilen güvenlik yöntemlerinin mukayeseleri yapılarak Çizelge 2.1'de verilmiştir.

Çizelge 2.1 DHCP güvenlik yöntemlerinin mukayesesi

Yöntem	**Mesaj Doğrulama**	**Kaynak doğrulama**	**Güvenlik seviyesi**	**Ek sistem**
Anahtar doğrulama	-	İstemci, Sunumcu	Düşük	-
Gecikmeli doğrulama	✓	İstemci, Sunumcu	Yüksek	-
Kerberos-V ile doğrulama	✓	İstemci	Orta	Kerberos sunumcusu
Sertifika bazlı doğrulama	✓	İstemci, Sunumcu	Yüksek	Sertifika yönetim sistemi
Kullanıcı doğrulama	-	Yok	Düşük	Doğrulama sunumcusu, doğrulama istemcisi

Yöntemlerden, "gecikmeli doğrulama" yöntemi, sağladığı güvelik seviyesi, gerçekleme kolaylığı ve herhangi bir ek sisteme ihtiyaç duymaması nedeni ile diğer yöntemlerden daha kullanışlıdır.

2.5 İlgili çalışmalar

DHCPv6 ile ilgili olarak yapılmış çalışmalardan bazıları aşağıda özetlenmiştir.

WIDE-DHCPv6, KAME Project [KAME Project] grubu tarafından geliştirilmiş olan DHCPv6 sistemidir İstemci, sunumcu ve ajan uygulamalarını içerir. Kısmen, RFC-3315, RFC-3319, RFC-3633 ve RFC-3646 uyumludur. Linux ve BSD işletim sistemleri üzerinde çalışabilmektedir. Sunumcu uygulaması istemcilere IPv6 adres tahsisi yapmamaktadır, sadece ağ konfigürasyon bilgilerini iletir. İstemci ve sunumcu için doğrulama ve yetkilendirme mekanizması yoktur. DHCP saldırılarına karşı savunmasızdır.

WIDE-DHCPv6 projesi C/C++ programlama dillerinde geliştirilmiştir. Açık kaynak kodlu olmasına rağmen uygulama geliştiriciler için yardımcı dokümanlar içermemektedir.

Dibbler, Gdansk Teknik Üniversitesi, Bilgisayar Mühendisliği bölümünde yüksek lisans tez çalışması olarak Tomasz Mrugalski ve Marek Senderski tarafından 2004 yılında geliştirilmiştir [Mrugalski, 2005]. Sistem üzerinde geliştirme çalışmaları halen devam etmektedir. İstemci, sunumcu ve ajan uygulamalarını içerir. Kısmen, RFC-3315, RFC-3736, RFC-3898 ve RFC-3646 uyumludur. Linux ve Windows XP/2003 işletim sistemlerinde çalışabilmektedir. İstemci ve sunumcu için doğrulama ve yetkilendirme mekanizması yoktur. DHCP saldırılarına karşı savunmasızdır.

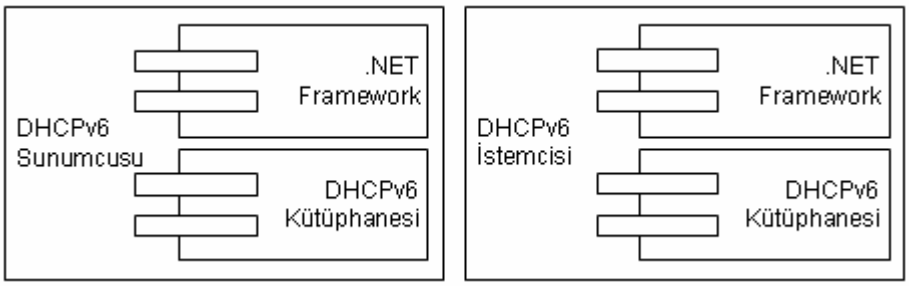
Dibbler projesi açık kaynak kodlu DHCPv6 uygulamaları arasında en yaygın kullanılanıdır. Çok katmanlı mimaride, nesneye yönelik programlama metodolojisine uygun olarak geliştirilmiştir. DHCPv6 temel fonksiyon ve yapıları C++ programlama dilinde, sistem seviyesindeki işlemler ise C programlama dilinde yazılmıştır. Dibler'in Windows XP/2003 işletim sistemleri için geliştirilmiş sürümünde sistem seviyesindeki bazı fonksiyonlar Microsoft tarafından işletim sistemi ile birlikte dağıtılan küçük uygulama parçacıkları ile sağlanmıştır.

Dibbler projesi, kullanıcı ve uygulama geliştiriciler için kılavuzlar ve kaynak kodlardan oluşturulmuş yardım dokümanları içermektedir.

3 GÜVENLİ DHCPV6 SİSTEMİNİN TASARIMI

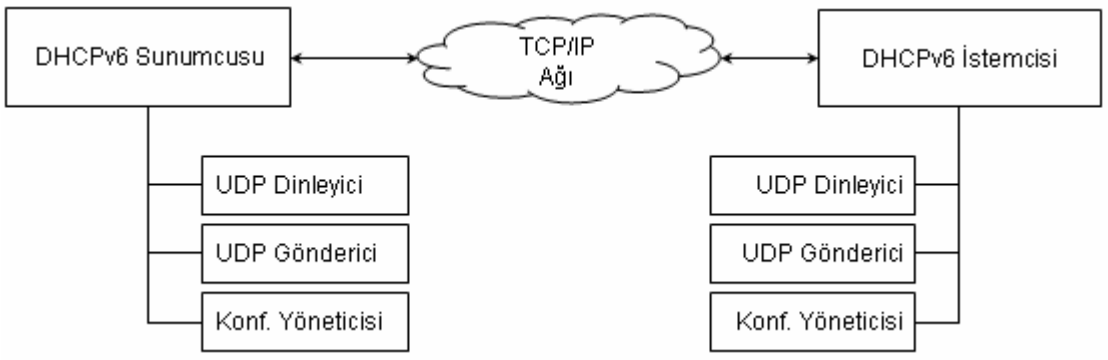
DHCP servisi ağdaki kritik servislerden kabul edildiği için güvenliğinin sağlanması gereklidir. Bu nedenle farklı yöntemler geliştirilmiş olmasına karşın mevcut sistemler güvenliği uygulama seviyesinde sağlamaktadırlar. Bu durum protokol bazlı yapılacak saldırılara karşı sistemlerin savunmasız olduklarını göstermektedir. DHCP sistem güvenliği, protokol bazında güvenliği sağlayan DHCP sunumcu ve istemci uygulamaları ile sağlanabilir. Tasarlanan sistemde, protokol bazlı güvenliği sağlamak için Bölüm 2.4'te incelenen güvenlik yöntemlerinden "gecikmeli doğrulama" yönteminin sistemde kullanılması kararlaştırılmış, yöntemin mevcut dezavantajları en aza indirgenerek tasarlanan sistemin kullanılabilir olması hedeflenmiştir.

İstemci ve sunumcu uygulamalarının hızlı geliştirilebilmesi için ortak ihtiyaçlar (mesaj yapıları, sabit değerler, günlük mekanizması vs.) bir araya toparlanarak ayrı bir bileşen olarak tasarlanmış ve DHCPv6 kütüphanesi olarak adlandırılmıştır. Genel olarak sistem bileşenleri Şekil 3.1'de verilmiştir.



Şekil 3.1 Güvenli DHCPv6 sistemi bileşenleri

Tasarlanan güvenli istemci ve sunumcu uygulamaları için fonksiyonel bağlantı diyagramı Şekil 3.2'de verilmiştir.



Şekil 3.2 İstemci-Sunumcu fonksiyonel bağlantı diyagramı

3.1 Güvenlik Yönteminin Seçimi

Tasarlanan DHCPv6 sisteminde güvenlik yöntemi olarak, RFC-3118'de [Droms and Arbaugh, 2001] tanımlanan, RFC-3315'te [Droms at al, 2003] DHCPv6 için güncellenen "gecikmeli doğrulama" yöntemi seçilmiştir. Yöntemin tercih edilmesindeki temel nedenler;

- İstemci ve sunumcunun birbirlerinin kimliklerini doğrulayabilmesi,
- DHCP mesaj içeriklerinin doğrulanabilmesi,
- Mesaj tekrarlama yöntemiyle yapılabilecek saldırıların önlenebilmesi,
- Başka alt sistemlere ihtiyaç duymaması,
- Diğer yöntemler için alt yapı oluşturması,
- Yerel alan ağlarda kullanımının kolay olmasıdır.

Yöntemin en belirgin dezavantajlarından olan, istemci ve sunumcu arasında mesajlaşma yapılmadan önce anahtar paylaşım sorununun, önerilen "varsayılan anahtar kullanımı" yöntemi ile giderilmesi tasarlanmıştır.

3.2 Güvenlik Mekanizmasının Tasarımı

Sistemde, güvenlik mekanizması sistem yöneticisi tarafından isteğe bağlı olarak istemci ve sunumcu tarafında ayrı ayrı devreye sokulabilir özellikte tasarlanmıştır.

Tasarlanan güvenlik mekanizmasının sağladığı özellikleri şunlardır:

- DHCPv6 istemcisi ve sunumcusu birbirlerinin kimliklerini doğrulayabilmektedirler.
- Sistem, DHCPv6 mesaj içeriklerine yapılacak müdahalelere karşı korunaklıdır.
- Sistem, mesaj tekrarlama yöntemi ile yapılacak saldırılara karşı korunaklıdır.
- Sistemde varsayılan anahtar kullanımı ile anahtar dağıtımına ihtiyaç yoktur.

DHCPv6 sunumcusunda güvenlik modülü devreye alındığında; anahtarı veritabanında tanımlı olan istemciler için sunumcu hizmet verir. Bu sayede, art niyetli istemcilerden gelen mesajlar gözardı edilerek, DHCPv6 saldırılarından korunur.

DHCP istemcisinde güvenlik modülü devreye alındığında, istemci, anahtarını bilen sunumculardan hizmet almaya çalışır. Gelen ADVERTISE mesajlarından sunumculardan hangilerinin güvenilir sunumcular olduğunu tespit eder ve o sunumcularla işleme devam eder. İstemci şifresini bilemeyen sunumcular istemci için sahte sunumcu kabul edilir ve bu sunuculardan hizmet alınmaz.

Genel olarak, DHCP protokolünün güvenliği için geliştirilen yöntemlerin uygulanabilirliliğinde en büyük sıkıntı, istemcilerin konfigürasyon öncesi bazı ayarlara veya özelliklere sahip olma gereksinimleridir. Seçilen güvenlik yönteminin istemci ve sunumcunun konfigürasyon öncesi anahtar paylaşım ihtiyacı, tasarlanan sistemde ***"varsayılan anahtar"*** kullanımı ile kısmen giderilmiştir.

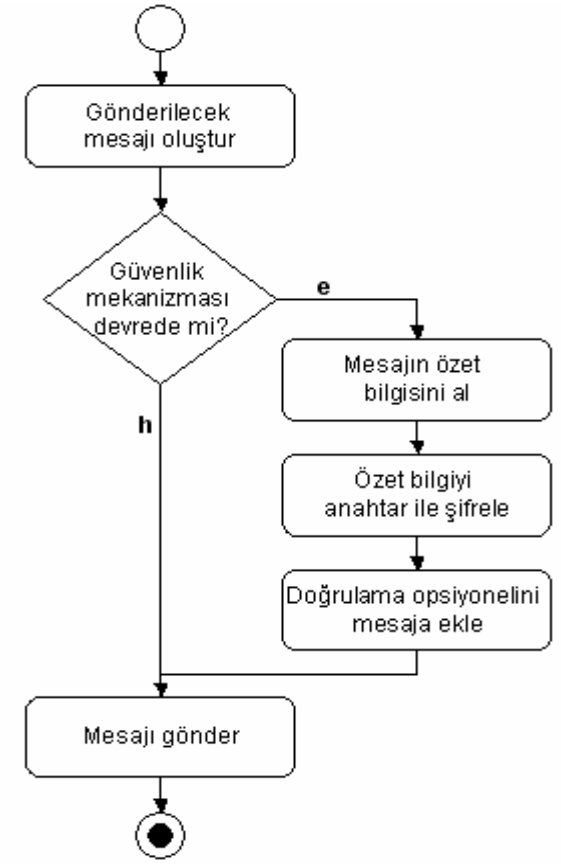
Tasarlanan DHCPv6 istemcisine herhangi bir anahtar atanmadığı müddetçe MAC adresini varsayılan anahtarı olarak kullanarak güvenli iletişim kurmak ister. Tasarlanan DHCPv6 sunumcusu, anahtarını bulamadığı DHCP istemcisi için varsayılan anahtar olarak istemci MAC adresini seçer ve istemci ile iletişimine devam eder. Bu özelliği sayesinde, tasarlanan sistemde anahtar paylaşımının önüne geçilmiştir. Benzer yaklaşımlar farklı tasarımlarda da benimsenmiş olmasına karşın, DHCPv6 sistemlerinde uygulanmamıştır [Ju and Han, 2005]. Varsayılan anahtar

kullanım yöntemi her ne kadar sistemde güvenlik açığı yaratsa da, bilinen güvenlik açıklarını kapmakta, anahtar paylaşımını ortadan kaldırmaktadır. Varsayılan anahtar kullanımı mekanizması da yine güvenlik mekanizması gibi opsiyonel olup, sistem yöneticisi tarafından devre dışı bırakılabilir.

Tasarlanan sistemde, mesaj gönderecek uç birim, DHCP mesajı göndermeden önce güvenlik mekanizmasının durumunu kontrol eder ve güvenlik mekanizmasının devrede olması durumunda, MD5 algoritmasına [Rivest, 1992] göre mesaj özetini hesapladıktan sonra istemci anahtarı ile şifreleyerek "mesaj doğrulama" opsiyonel bilgisini oluşturur. Karşı tarafın, gelen mesajın değiştirilmediğini ve yetkili bir uç noktadan gönderildiğini anlayabilmesi için gönderici oluşturduğu mesaj doğrulama opsiyonel bilgisini DHCP mesajına ekleyerek gönderir. Benzer şekilde, mesajı alan uç birim mesajın doğruluğunu ve yetkili bir uç birim tarafından gönderildiğini kontrol etmek için, mesajın özet bilgisini istemci anahtarı ile şifreleyerek, doğrulama opsiyonel bilgisi içinde gönderilen mesaj doğrulama verisi ile karşılaştırır. Değerlerin birbiri ile uyuşması durumunda mesaj geçerli kabul edilir ve işleme alınır. İstemci ve sunumcu tarafında mesaj gönderme ve gelen mesajların kontrol edilmesine ilişkin akış diyagramları Şekil 3.3, Şekil 3.4, Şekil 3.5, ve Şekil 3.6'da verilmiştir.

3.2.1 İstemcinin Mesaj Göndermesi

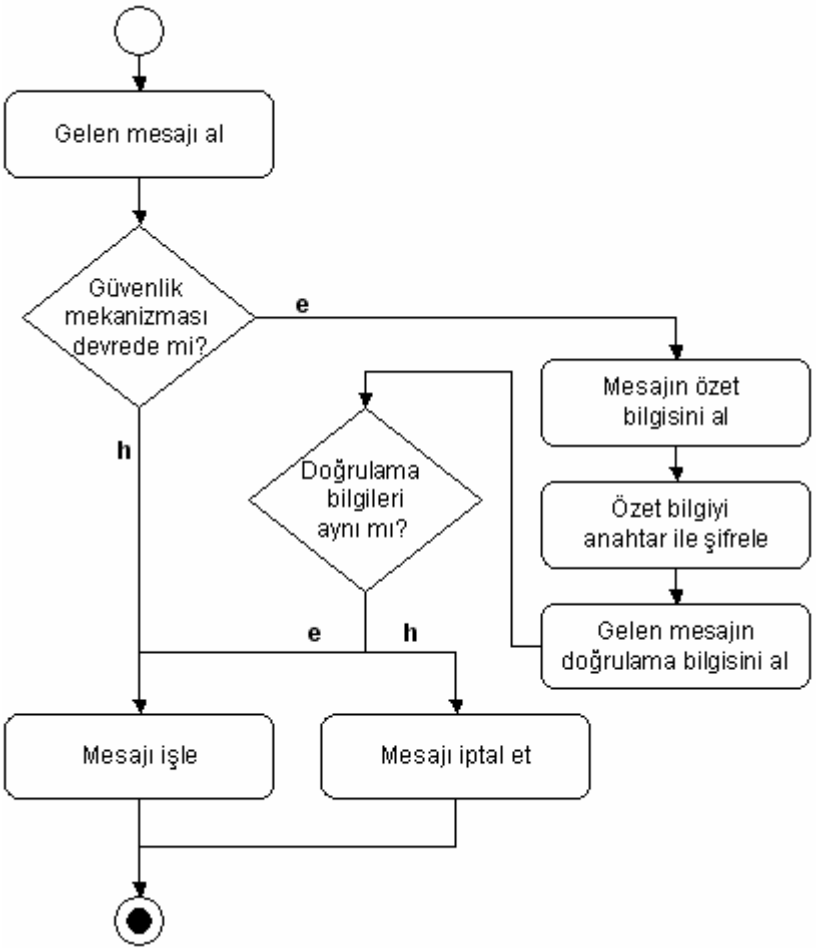
Gerçeklenen DHCPv6 istemcisi, güvenlik mekanizması devreye alındığında mesaj göndermeden önce mesajın sunumcu tarafından doğrulanabilmesi için mesaj doğrulama opsiyonel bilgisine mesaj doğrulama değerini eklenerek gönderir.



Şekil 3.3 İstemcinin mesaj göndermesi

3.2.2 İstemcinin Gelen Mesajları İşlemesi

Gerçeklenen DHCPv6 istemcisi, güvenlik mekanizmasının devrede olması durumunda mesaj doğrulama bilgisini kontrol ettikten sonra mesajı işleme alır. Gelen mesajın özet bilgisini oluşturarak, anahtarı ile şifreler. Mesajın doğrulama bilgisindeki şifreli değer ile karşılaştırarak değerlerin aynı olup olmadığını kontrol eder. Değerlerin uyuşması durumunda mesajın güvenilir bir sunumcudan geldiğini anlar ve mesajı işleme alınır.



Şekil 3.4 İstemcinin gelen mesajı işlemesi

3.2.3 Sunumcunun Mesaj Göndermesi

Gerçeklenen DHCPv6 sunumcusu, güvenlik mekanizması devreye alındığında mesaj göndermeden önce mesajın istemi tarafından doğrulanabilmesi için mesaj doğrulama opsiyonel bilgisine mesaj doğrulama değerini ekleyerek gönderir.

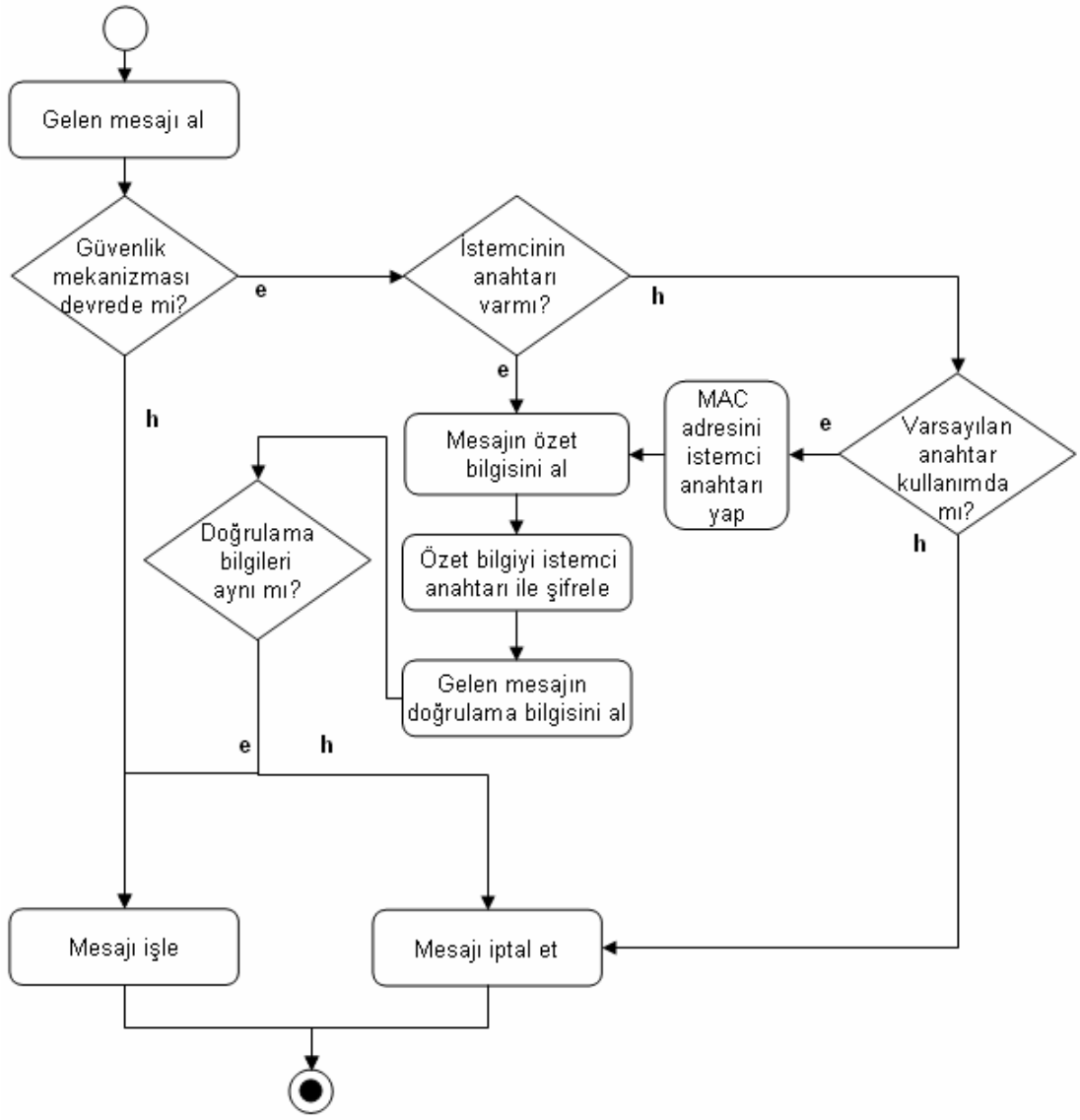
Şekil 3.5 Sunumcunun mesaj göndermesi

3.2.4 Sunumcunun Gelen Mesajları İşlemesi

Gerçeklenen DHCPv6 sunumcusu, güvenlik mekanizmasının devrede olması durumunda mesaj doğrulama bilgisini kontrol ettikten sonra mesajı işleme alır. Gelen mesajın özet bilgisini oluşturarak, veritabanından çektiği istemci anahtarı ile şifreler. Mesajın doğrulama bilgisindeki şifreli değer ile karşılaştırarak değerlerin aynı olup olmadığını kontrol eder. Değerlerin uyuşması durumunda mesajın güvenilir bir istemciden geldiğini anlar ve mesajı işleme alınır.

İstemci anahtarının veritabanında olmaması durumunda, varsayılan anahtar kullanımı kontrol edilir. Eğer "varsayılan anahtar kullanma" özelliği sistem yöneticisi tarafından devreye alınmış ise; istemci için varsayılan anahtar kullanılarak mesaj doğrulu kontrol edilir. Eğer istemci de "varsayılan anahtarı" kullanıyor ise,

mesaj doğrulaması yapılmış olur ve sunumcu istemci için anahtar tanımını veritabanına ekler. “Varsayılan anahtar kullanım” özelliği devre dışı ise, istemcinin bilinmeyen bir istemci olduğu kabul edilir ve istemciden gelen mesajlar işlenmez.



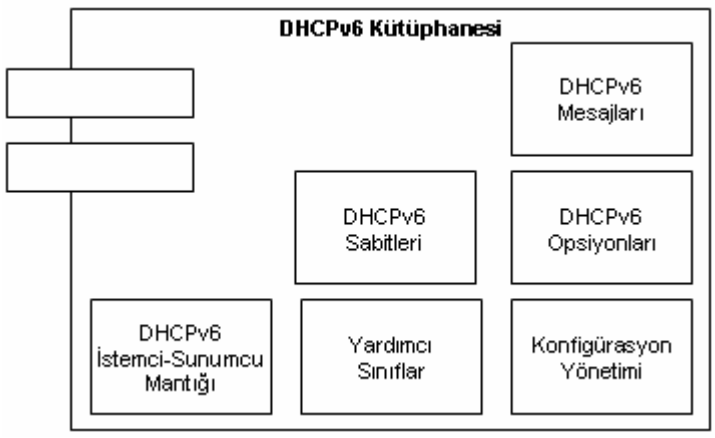
Şekil 3.6 Sunumcunun gelen mesajı işlemesi

3.3 DHCPv6 Kütüphanesi

DHCP istemci ve sunumcu yazılımlarının tarafından kullanılan ortak ihtiyaçların ayrı bir bileşen olarak tasarlandığı yardımcı kütüphanedir. DHCPv6 kütüphanesi, hem istemci hem de sunumcu için ortak alt yapı oluşturacağı gibi

istemci ve sunmucu yazılımlarının farklı uygulama tiplerinde (masaüstü, sistem servisi vb.) geliştirilmesinde kolaylık sağlayacak şekilde tasarlanmıştır.

DHCPv6 kütüphanesi, istemci ve sunumcu uygulamaları tarafından kullanılacak olan DHCPv6 mesajlarını, opsiyonlarını, sabit değerlerini, istemci-sunumcu sınıflarını, konfigürasyon yönetimi bileşenlerini ve uygulamalarda kullanılan diğer yardımcı sınıfları içerir (Şekil 3.7).



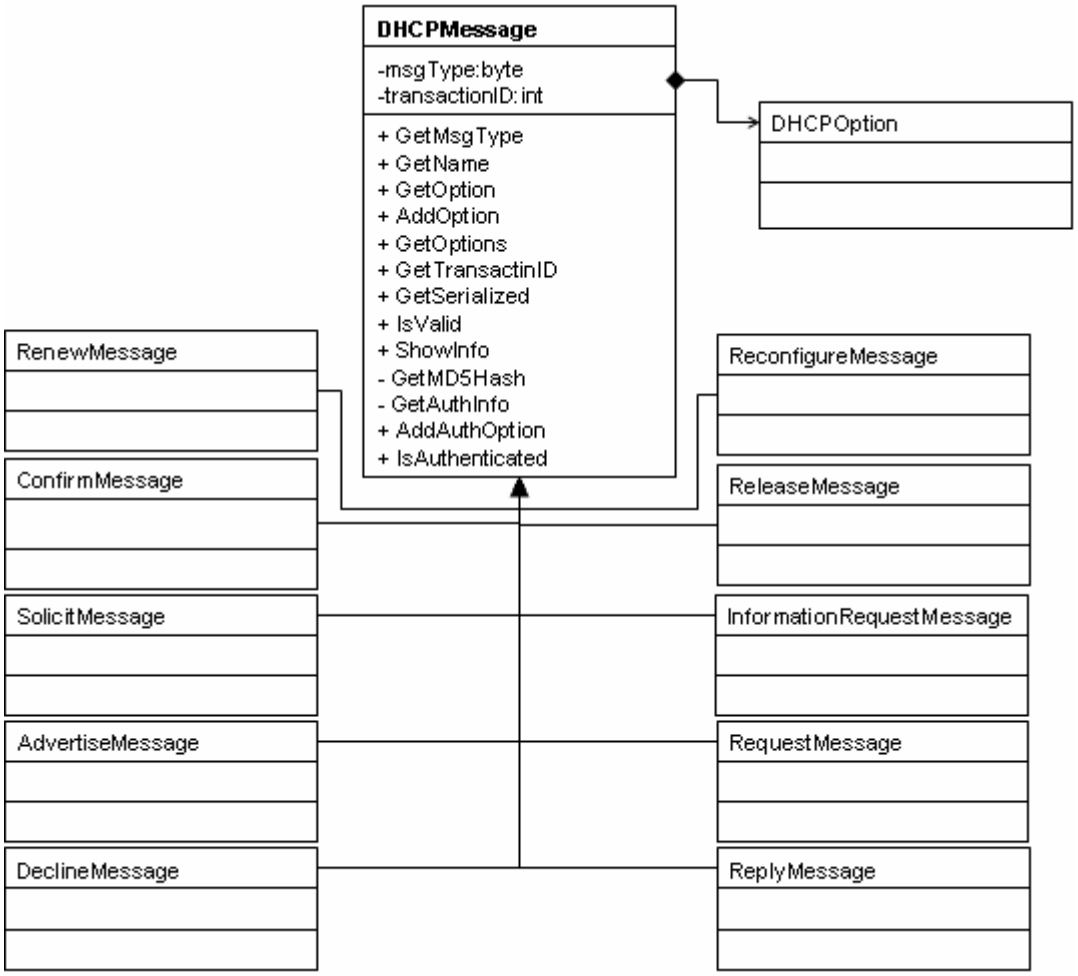
Şekil 3.7 DHCPv6 kütüphanesi

3.3.1 DHCPv6 Mesajları

Gerçeklenen güvenli DHCPv6 sistemi kütüphanesi, RFC3315'te tanımlı aşağıdaki mesajları içermektedir:

- SOLICIT
- ADVERTISE
- REQUEST
- INFORMATION-REQUEST
- REPLY
- RELEASE
- DECLINE
- RENEW
- CONFIRM
- RECONFIGURE

Tanımlı mesajlar, nesneye yönelik programlama metodolojine uygun olarak tasarlanan DHCPMessage sınıfından türetilmişlerdir. Her mesaj kendine özgü opsiyonel bilgi ekleme ve geçerlilik kontrolü fonksiyonlarına sahiptir. Şekil 3.9'da gerçeklenen mesajlar ve aralarındaki hiyerarşik yapı gösterilmiştir.



Şekil 3.8 DHCPv6 mesajları sınıf diyagramı

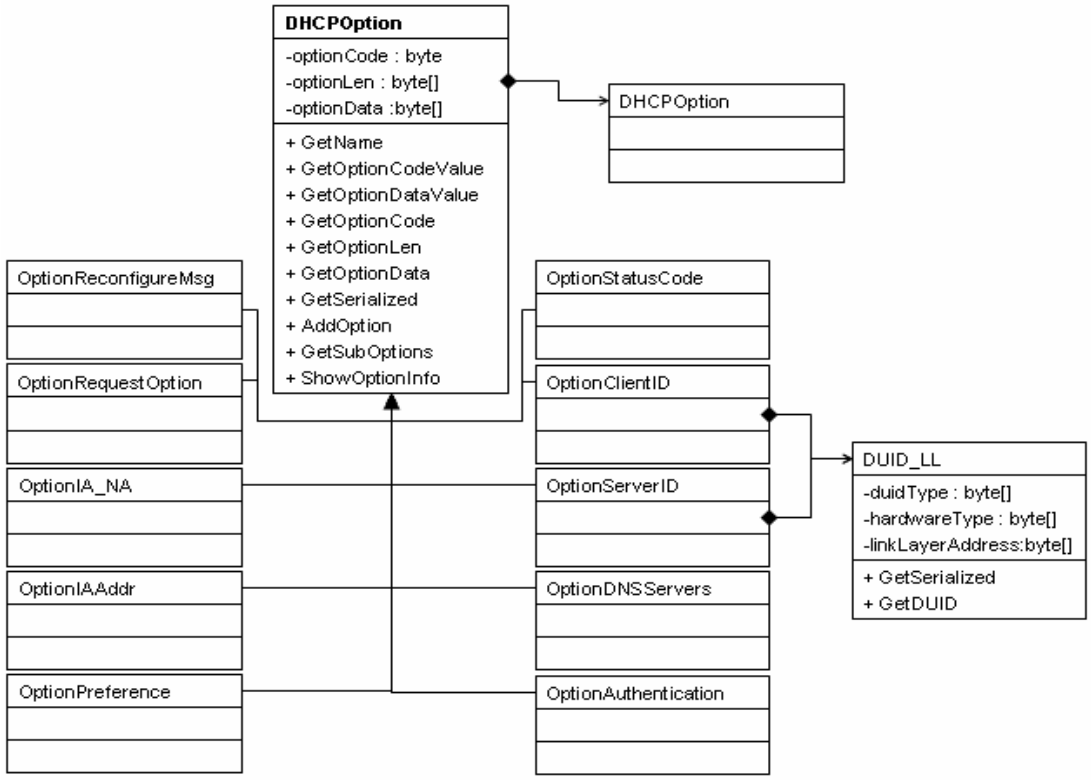
Yeni bir mesajın eklenmesi durumunda, ilgili mesaj DHCPMessage sınıfından türetilerek ihtiyaç duyulan temel fonksiyonlar otomatik olarak sağlanmış olunur.

3.3.2 DHCPv6 Opsiyonları

Tasarlanan DHCPv6 kütüphanesi, RFC-3315 [Droms at al, 2003] ve RFC-3646'da [Droms, 2003] tanımlı aşağıdaki opsiyonel bilgileri içerir:

- OPTION_CLIENTID
- OPTION_SERVERID
- OPTION_IA_NA
- OPTION_IAADDR
- OPTION_ORO
- OPTION_PREFERENCE
- OPTION_RELAY_MSG
- OPTION_AUTH
- OPTION_STATUS_CODE
- OPTION_RECONF_MSG
- OPTION_DNS_SERVERS

Tanımlı opsiyonlar, nesneye yönelik programlama metodolojine uygun olarak tasarlanan DHCPOption sınıfından türetilmişlerdir. Yeni opsiyon tanımlanması durumunda, ilgili opsiyon DHCPOption sınıfından türetilerek temel fonksiyonlar sağlanmış olunur. Şekil 3.10'da, gerçeklenen opsiyonlar arasındaki hiyerarşik yapı verilmiştir.

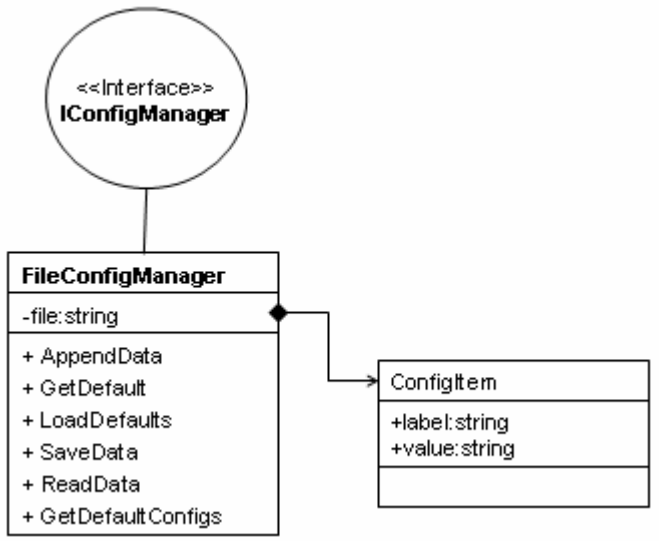


Şekil 3.9 DHCPv6 opsiyonları sınıf diyagramı

3.3.3 Konfigürasyon Yönetimi

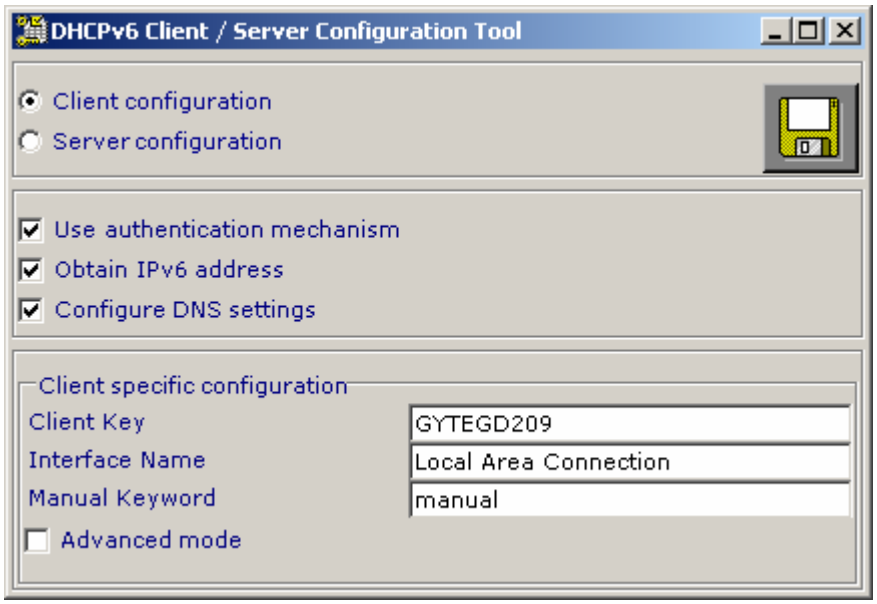
DHCPv6 kütüphanesi, tasarlanan DHCPv6 sisteminde ihtiyaç duyulan konfigürasyon bilgilerinin istemci ve sunumcu tarafında ortak yönetimi için ihtiyaç duyulan konfigürasyon yönetim sınıflarını içerir.

Tasarlanan sistemde, konfigürasyon bilgileri metin dosyasında tutulduğu için tanımlanan konfigürasyon yönetimi arayüzü için sadece metin dosyası kullanılarak konfigürasyonu sağlayan sınıf tasarlanmıştır. İhtiyaç halinde konfigürasyon bilgilerinin saklanması için veritabanı veya XML kullanımı söz konusu olduğunda konfigürasyon yönetimi arayüzünden türetilecek sınıflarda istenen fonksiyonların tanımlanması mevcut sistemde değişikliğe neden olmadan yeni konfigürasyon sisteminin etkinleştirilmesini sağlayacaktır. Şekil 3.10'da konfigürasyon yönetimi için gerçeklenen sınıflar ve aralarındaki ilişkiler gösterilmiştir.



Şekil 3.10 DHCPv6 konfigürasyon yönetimi sınıf diyagramı

İstemci ve sunumcu konfigürasyonlarının kolaylıkla yapılabilmesi için kullanıcı arayüzleri tasarlanmıştır. Konfigürasyon yönetimi nesnelerini kullanan uygulama ile kullanıcı arayüz işlemleri ve konfigürasyon yönetim işlemleri birbirinden ayrılmıştır. Şekil 3.11'de, istemci konfigürasyonu için tasarlanan kullanıcı arayüz ekranı verilmiştir.



Şekil 3.11 DHCPv6 istemci konfigürasyonu kullanıcı arayüzü

3.3.4 DHCPv6 Sabitleri

RFC-3315 [Droms at al, 2003] ve RFC-3646'da [Droms, 2003] tanımlı sabit değişkenler özelliklerine göre sınıflandırılarak, DHCPv6 kütüphanesi içerisinden kullanımı kolaylaştıracak şeklide tanımlanmışlardır. Bu yöntem ile kod içerisinde sabit değer kullanımı merkezi hale getirilmiştir. Sabit değerler şu başlıklar altında sınıflandırılmıştır:

- DHCPv6 mesaj tipleri
- DHCPv6 opsiyon tipleri
- Azami değerler
- Zaman aşımı değerleri
- IP adresleri
- Port numaraları
- Durum kodları

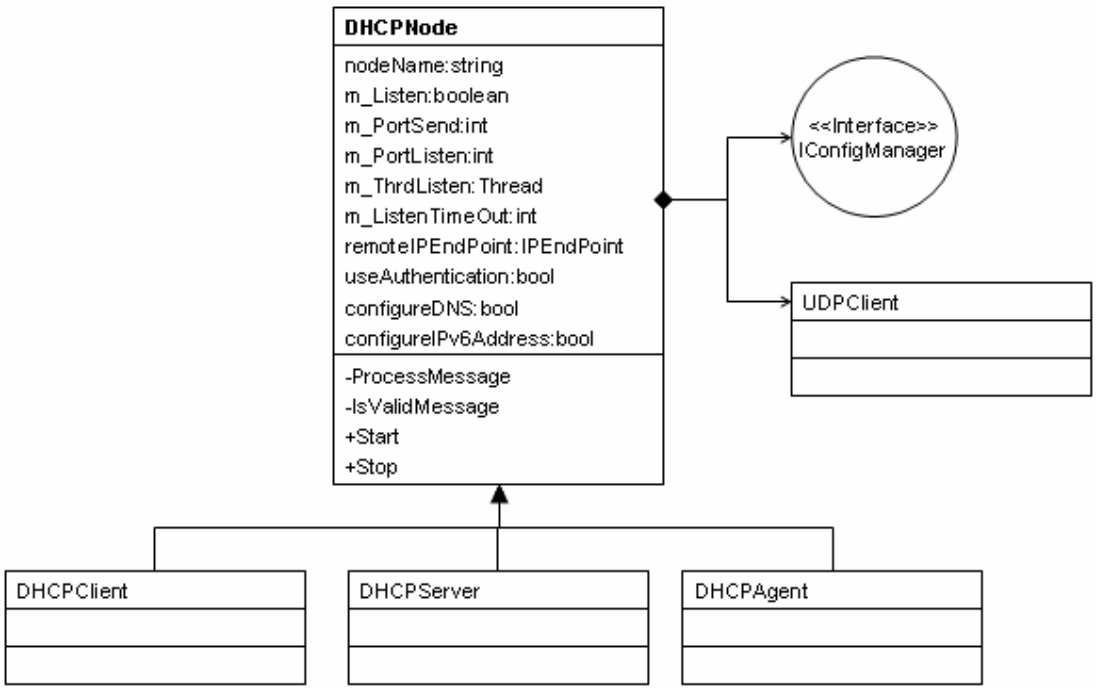
3.3.5 Yardımcı Sınıflar

DHCPv6 kütüphanesi, istemci ve sunumcu uygulamaları ve diğer DHCP6 kütüphanesi alt bileşenlerinde kullanılmak üzere tasarlanan yardımcı sınıflara sahiptir. Bunlar;

- Günlük (log) takibi metotlarını içeren sınıf,
- Yardımcı fonksiyonları (komut çalıştırma, IP adresi temini ve hesaplamaları vs.) içeren sınıf,
- Güvenlik metotlarını (şifreleme, özet değer üretme) içeren sınıflar olarak sıralanabilir.

3.3.6 İstemci ve Sunumcu Mantığı

DHCP sistemlerinde istemci ve sunumcu mesaj alma / gönderme, port açma / kapama gibi benzer ortak davranışlara sahiptir. Tasarlanan sistemde, benzer özellikler, nesneye yönelik programlama ilkelerine göre bir üst sınıf içerisinde tanımlanarak kodlamada basitliğe gidilmiştir (Şekil 3.12). Bu yöntemle ayrıca, geliştirilecek istemci ve sunumcunun lojiği kütüphane içerisinde tutularak, geliştirilecek uygulama tipinden (Windows form uygulaması, Windows konsol uygulaması veya Windows servisi) bağımsız hale getirilmiştir. Tasarlanan DHCPv6 kütüphanesi kullanılarak, istemci veya sunumcu uygulamaları kolaylıkla farklı uygulama tiplerinde hizmete sunulabilir.



Şekil 3.12 DHCPv6 uç birim sınıfların ilişkileri

3.4 Mesajlaşma Senaryoları

Tasarlanan güvenli DHCPv6 sisteminde, RFC-3315'te tanımlanan 4'lü ve 2'li mesajlaşma ile istemci konfigürasyonu senaryoları ile istemcinin IP adresi iade etmesi, istemcini IP adresi geçerlilik süresini artırması ve sunumcunun istemcilerin yeniden konfigürasyon bilgilerini almaları için tetiklemesi senaryoları ele alınmıştır.

İlgili senaryoların tümünde, güvenlik mekanizması devreye alınmış ve her bir senaryo için düzenlenecek saldırılara karşı sistemin korunmalı olduğu gözlenmiştir.

3.4.1 Dörtlü Mesajlaşma ile İstemci Konfigürasyonu

RFC-3315'te [Droms at al, 2003] tanımlı, 4'lü mesajlaşma ile istemcinin konfigürasyonunu tamamlaması senaryosu, gerçeklenen sistemde Şekil 3.13'te verilen iş akışı diyagramında olduğu gibi tasarlanmıştır.

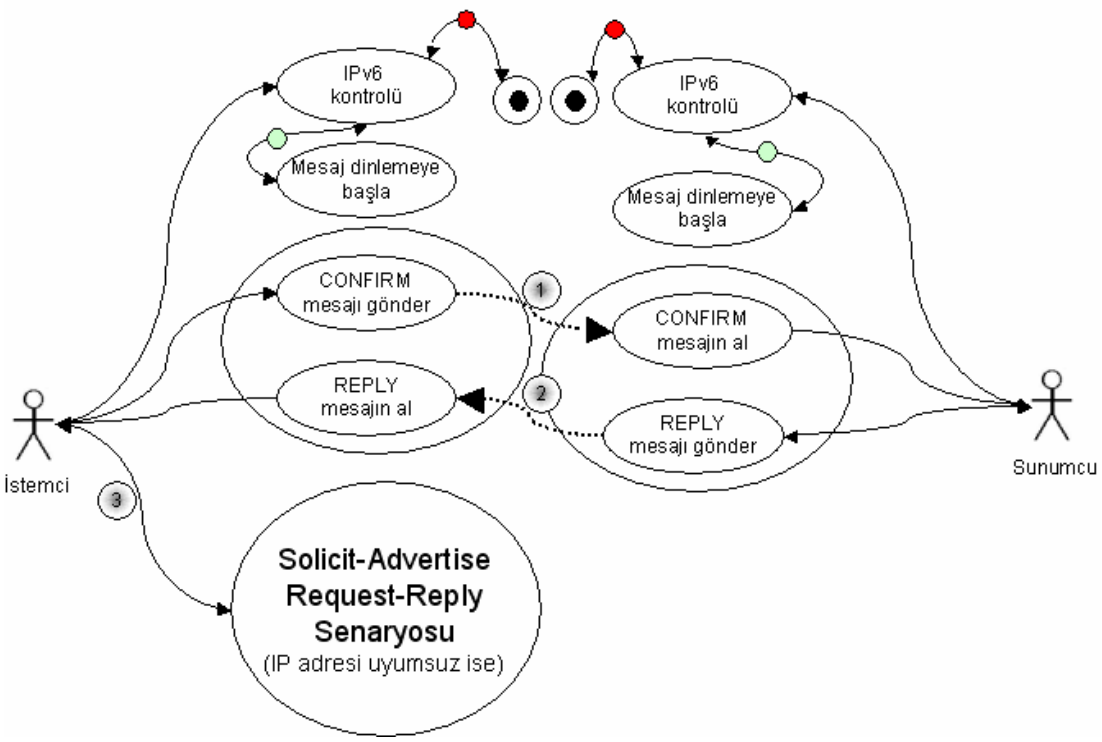
Şekil 3.13 4'lü mesajlaşma

İstemci ve sunumcu uygulamaları başladığında mevcut sistemin IPv6 uyumlu olup olmadığını kontrol eder. Sistemin uyumlu olmaması halinde uyarı mesajı vererek işlem sonlandırılır.

İstemci başlatıldığında, DHCP sunumcudan aldığı bir IPv6 adresi yoksa IPv6 adresi alacağı sunumcuyu tespit etmek için SOLICIT mesajını gönderir. SOLICIT mesajını alan sunumcu(lar), varlığını ve önceliğini istemciye bildirmek için ADVERTISE mesajını gönderirler. İstemci geçerli sunumculardan gelen ADVERTISE mesajlarında en yüksek önceliği belli bir süre boyunca bekler. En yüksek öncelikli sunucuyu seçerek, konfigürasyon bilgisini kendisine göndermesi için REQUEST mesajını gönderir. REQUEST mesajını alan sunumcu, istemciye IPv6 adres havuzunda tanımlı adreslerden birini ve tanımlı DNS ayarlarını REPLY mesajı ile istemciye gönderir. REPLY mesajını alan istemci, IPv6 adresinin kullanılabilirliliğini kontrol ettikten sonra IP ayarlarını yaparak konfigürasyon sürecini tamamlar ve sunumcunun göndereceği olası RECONFIGUE mesajlarını beklemeye başlar.

3.4.2 İkili Mesajlaşma ile İstemci Konfigürasyonu

RFC-3315'te [Droms at al, 2003] tanımlı, 2'li mesajlaşma ile istemcinin konfigürasyonunu tamamlaması senaryosu, gerçeklenen sistemde Şekil 3.14'te verilen iş akışı diyagramında olduğu gibi tasarlanmıştır.



Şekil 3.14 2'li mesajlaşma

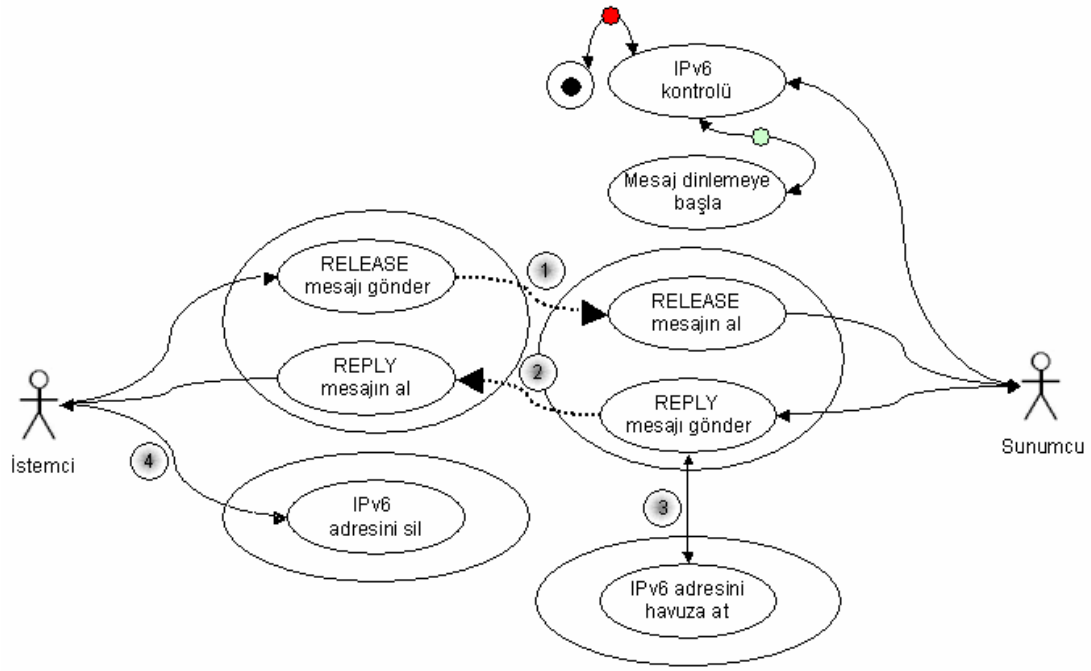
İstemci ve sunumcu uygulamaları başladığında mevcut sistemin IPv6 uyumlu olup olmadığını kontrol eder. Sistemin uyumlu olmaması halinde uyarı mesajı vererek işlem sonlandırılır.

İstemci başladığında, daha önce DHCP sunucudan aldığı IP adresine sahip ise mevcut adreslerinin geçerliliğini ağ değişimi durumuna karşı kontrol etmek için bu senaryoyu başlatır. İstemci, mevcut IPv6 adresleri için CONFIRM mesajını sunumcuya gönderir. Sunumcu gelen CONFIRM mesajından IPv6 adresini elde ederek, havuzunda tanımlı bir IPv6 adresi olup olmadığını kontrol eder. Adres havuzda tanımlı ise REPLY mesajına ekleyeceği durum bilgisine SUCCESS değerini ekler ve gönderir. SUCCESS durum bilgisine sahip REPLY mesajını alan istemci mevcut IPv6 adresini kullanmaya devam eder. IPv6 adresinin sunumcu havuzunda

tanımlı olmaması durumunda sunumcu durum bilgisine NOTONLINK değerini ekler ve istemciye gönderir. NOTONLINK durum bilgili REPLY mesajını alan istemci IPv6 adresini konfigürasyonundan siler ve yeni IPv6 adresi temini için 4'lü mesajlaşma senaryosunu başlatır.

3.4.3 İstemcinin IP Adresini Sunumcuya İade Etmesi

RFC-3315'te tanımlı, istemcinin kullanmakta olduğu IPv6 adresini sunumcuya iade etmesi, gerçeklenen sistemde Şekil 3.15'te verilen iş akışı diyagramında olduğu gibi tasarlanmıştır.

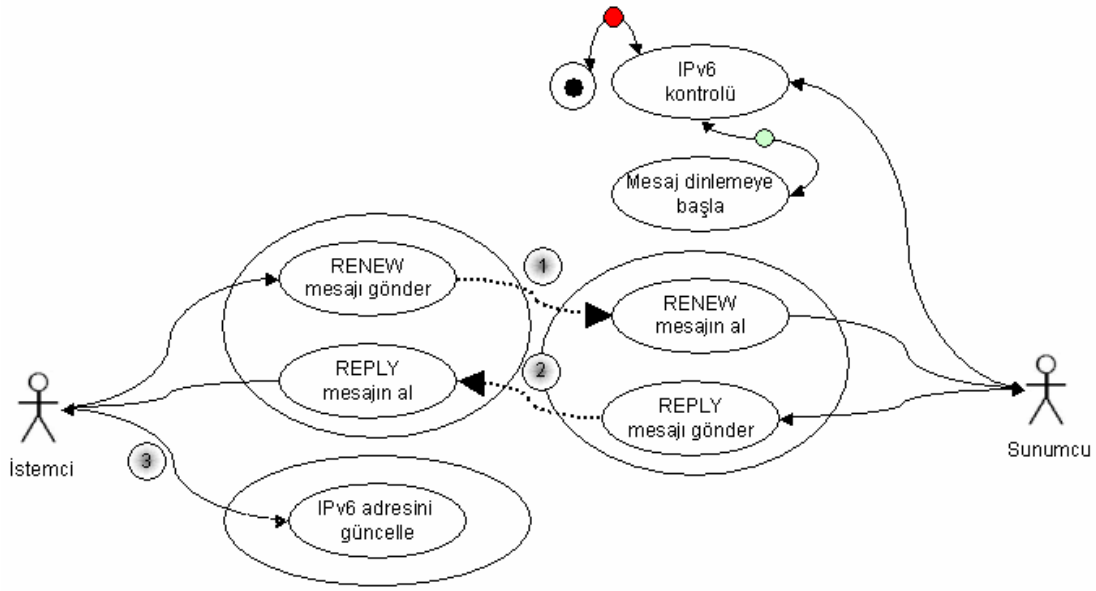


Şekil 3.15 İstemcinin IPv6 adresini sunumcuya iade etmesi

İstemcinin kullandığı IPv6 adreslerinden birini sunumcuya iade edilmesi istendiğinde, istemci IPv6 adresini RELEASE mesajına ekleyerek sunumcuya gönderir. Gelen RELEASE mesajını alan sunumcu, IPv6 adresinin kendi havuzunda tanımlı IPv6 adreslerinden biri olması durumunda IPv6 adresini daha sonra istekte bulunacak istemcilere göndermek üzere havuzuna atar ve istemciye REPLY mesajı gönderir. Gelen REPLY mesajını alan istemci IPv6 adresini konfigürasyonundan silerek işlemi tamamlamış olur.

3.4.4 İstemcinin IP Adresi Geçerlilik Süresini Artırması

RFC-3315'te [Droms at al, 2003] tanımlı, istemcinin kullanmakta olduğu IPv6 adresinin geçerlilik süresini uzatması, gerçeklenen sistemde Şekil 3.16'da verilen iş akışı diyagramında olduğu gibi tasarlanmıştır.

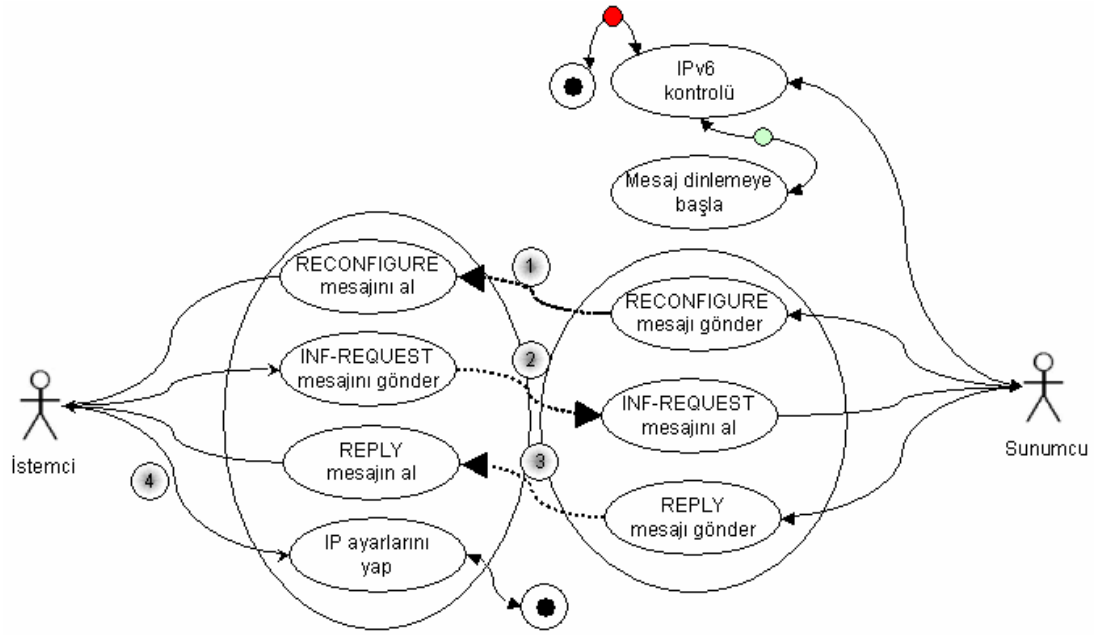


Şekil 3.16 İstemcinin IPv6 adresinin geçerlilik süresini uzatması

İstemci kullanmakta olduğu IPv6 adresini RENEW mesajına ekleyerek sunumcuya gönderir. RENEW mesajını alan sunumcu IPv6 adresinin kendi IPv6 adres havuzunda tanımlı olması durumunda istemciye REPLY mesajı ile adresin yeni geçerlilik süresini gönderir. REPLY mesajını alan istemci, IPv6 adresinin geçerlilik süresini güncelleyerek işlemini tamamlamış olur.

3.4.5 Sunumcunun İstemcileri Tetiklemesi

RFC-3315'te tanımlı, sunumcunun istemcileri yeniden konfigüre olmaları için tetiklemesi, gerçeklenen sistemde Şekil 3.17'de verilen iş akışı diyagramında olduğu gibi tasarlanmıştır.



Şekil 3.17 Sunumcunu istemcileri tetiklemesi

Ağ servis bilgilerinin değişikliği durumunda sunumcu, daha önceden IPv6 adresi tahsisi yaptığı istemcilere RECONFIGURE mesajını göndererek istemcilerin yeniden konfigürasyon bilgilerini almaları için onları uyarır. RECONFIGURE mesajını alan istemciler değişen konfigürasyon bilgilerini almak için INFORMATION-REQUEST mesajı göndererek sunumcudan yeni değerleri isterler. REPLY mesajı ile sunumcu yeni değerleri istemciye gönderir. Gelen REPLY mesajın alan istemci konfigürasyon bilgilerini güncelleyerek sunumcunun başlattığı işlemin tamamlanmasını sağlar.

3.5 Tasarlanan DHCPv6 Sunumcusunun Özellikleri

Tasarlanan DHCPv6 sunumcusu, DHCPv6 istemcisinden gelen mesajları işlemekte, istemciye tanımlı ağ konfigürasyon bilgileri ile birlikte IPv6 adresi tahsis edebilmektedir. Genel özellikleri şunlardır:

- Kısmen RFC-3315 [Droms at al, 2003], RFC-3646 [Droms, 2003] ve RFC-3118 [Droms and Arbaugh, 2001] uyumludur.
- SOLICIT, REQUEST, DECLINE, RELEASE, RENEW, CONFIRM ve INFORMATION-REQUEST mesajlarını işleyebilir.

- Ağ erişim bilgilerinde değişiklik olması durumunda istemcilere RECONFIGURE mesajı göndererek istemcilerin yeniden yapılandırılmalarını sağlayabilir.
- IPv6 adres tahsisi yapabilir.
- Tanımlı DNS sunumcularının listesini istemciye iletebilir.
- Sistem yönetici tarafından güvenlik modülü devre dışı bırakılabilir.
- Sistem yöneticisi tarafından sadece ağ konfigürasyon bilgilerini gönderecek şekilde ayarlanabilir.
- Konfigürasyon için sistem yöneticisine kullanıcı arayüzü sağlar.

3.6 Tasarlanan DHCPv6 İstemcisinin Özellikleri

Tasarlanan DHCPv6 istemcisi, ağa bağlanacak konak için otomatik IP adres tahsisi ve ağ konfigürasyonu işlemlerini gerçekleştirebilir. Genel özellikleri şunlardır:

- Kısmen RFC-3315 [Droms at al, 2003], RFC-3646 [Droms, 2003] ve RFC-3118 [Droms and Arbaugh, 2001] uyumludur.
- ADVERTISE, REPLY ve RECONFIGURE mesajlarını işler.
- Mevcut IP ayarlarının doğruluğu için CONFIRM-REPLY mesajlaşması yapabilir.
- Kullandığı IPv6 adresinin sunumcuya iadesi için RELEASE-REPLY mesajlaşması yapabilir.
- Mevcut IP adresinin geçerlilik süresini artırmak için RENEW-REPLY mesajlaşması yapabilir.
- Birden fazla DHCP sunumcu arasından en yüksek öncelikli sunumcudan istekte bulunabilir.
- IPv6 adres ataması yapabilir.
- DNS sunumcusu konfigürasyonu yapabilir.
- Sistem yönetici tarafından güvenlik modülü devre dışı bırakılabilir.
- Sistem yöneticisi tarafından sadece ağ konfigürasyonunun yapılandırılması için ayarlanabilir (IP adresi almadan DNS vb. ağ ayarlarının yapılandırılması).

- Konfigürasyon için sistem yönetisine kullanıcı arayüzü sağlar.

3.7 Dokümantasyon

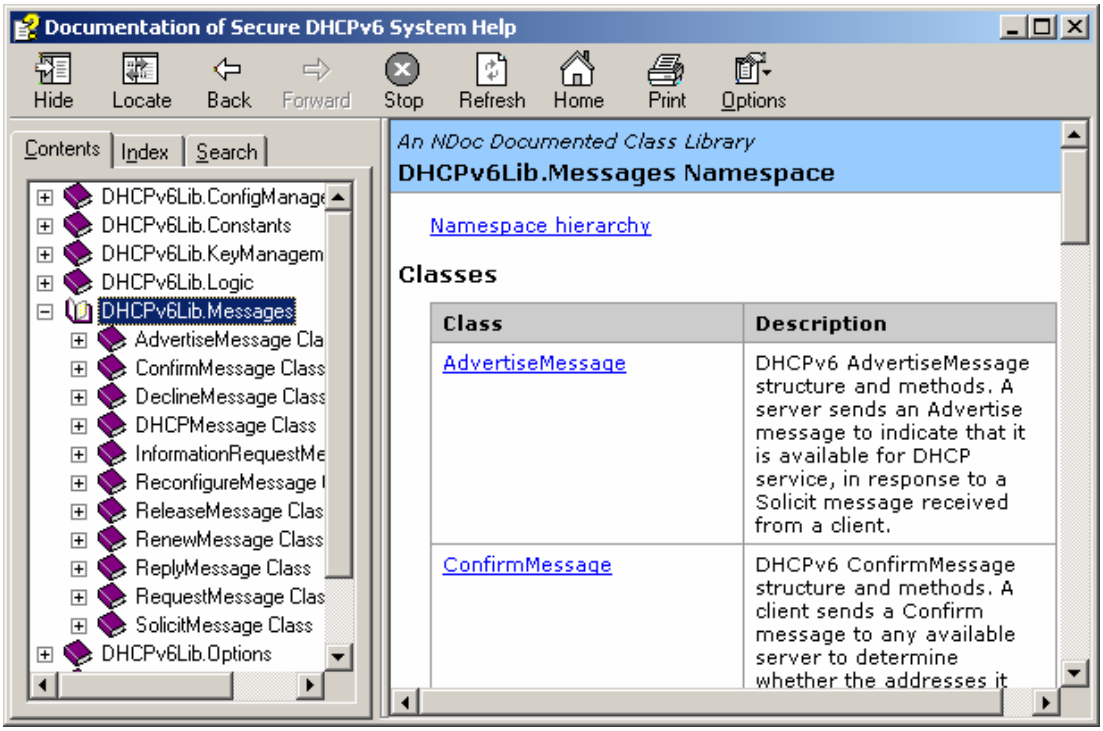
Gerçeklenen "Güvenli DHCPv6 Sistemi" projesinde, dokümantasyona önem verilmiş olup, dokümantasyonla ilgili çalışmalar yapılmıştır. Bu çalışmalar:

- Kodlama esnasında, gerekli yerlerde açıklama bilgilerinin girilmesi,
- Her nesne ve o nesneye ait metotların özet bilgilerinin girilmesi,
- Sistem geliştiriciler için MSDN standardında yardım dosyalarının üretilmesi,
- İstemci ve sunumcu için mesaj alma / gönderme akış diyagramlarının (Şekil 3.3, 3.4, 3.5 ve 3.6) hazırlanması,
- İlgili mesaj, opsiyon ve DHCP öğeleri için UML diyagramlarının (Şekil 3.8, 3.9, 3.10 ve 3.12) hazırlanması,
- İstemci ve sunumcu arasında mesajlaşma senaryoları için akış diyagramlarının (Şekil 3.13, 3.14, 3.15, 3.16 ve 3.17) hazırlanması olarak sıralanabilir.

Şekil 3.18'de, kaynak kod içerisinde yapılan açıklamalardan örnek verilmiştir.

```
namespace DHCPv6Lib.Messages
{
    /// <summary>
    /// DHCPv6 SolicitMessage structure and methods.
    ///     A client sends a Solicit message to locate
    ///     servers.
    /// </summary>
    public class SolicitMessage : DHCPMessage
    {
        /// <summary>
        /// Default constructer
        /// </summary>
        public SolicitMessage(int transactionID)
            :base(DHCPMessages.SOLICIT,transactionID)
        {
            //add OptionClientID option to options list...
            string linkLayerAddr =Utility.GetMACAddress();
            DUID_LL duid = new DUID_LL(linkLayerAddr);
            OptionClientID optClientID = new OptionClientID(duid);
            this.AddOption(optClientID);
        }
```

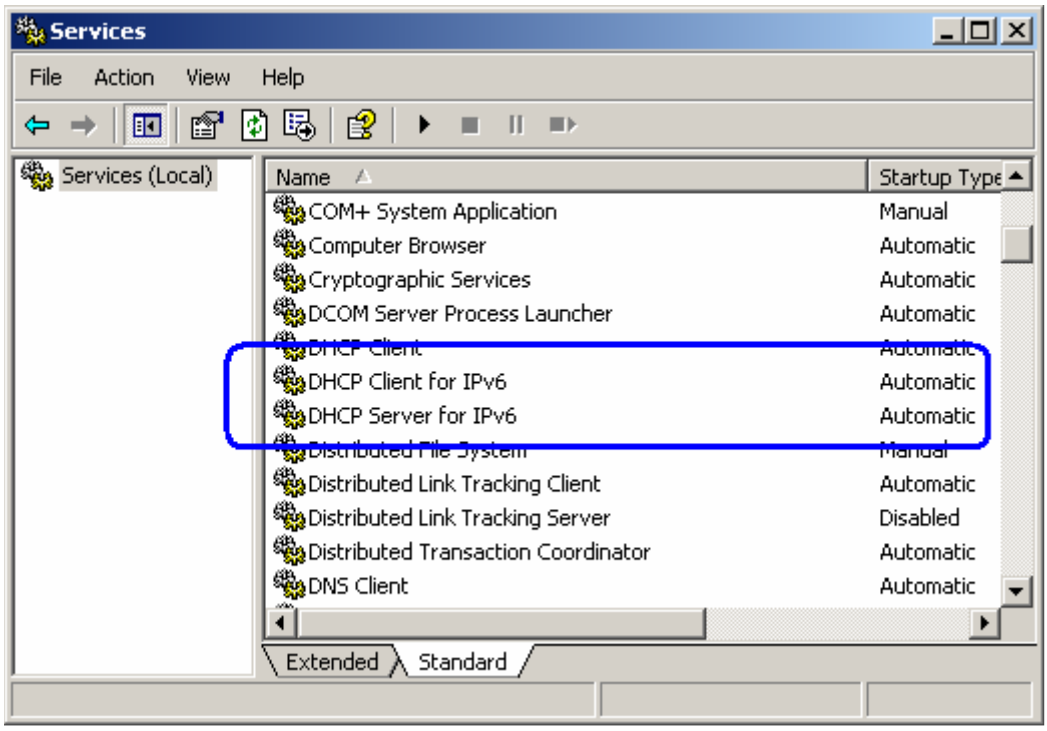
MSDN formatında yardım dosyası oluşturmak için, kod içerisinde gerekli yerlerde özel etikler kullanılarak açıklama bilgileri girilmiş ve açık kaynak kodlu Ndoc [NDoc] uygulaması kullanılarak bu açıklamalar doğrultusunda MSDN formatında yardım dosyaları üretilmiştir (Şekil 3.19).



Şekil 3.19 MSDN formatında yardım dosyası (örnek bölüm)

4 GERÇEKLEME VE TEST SONUÇLARI

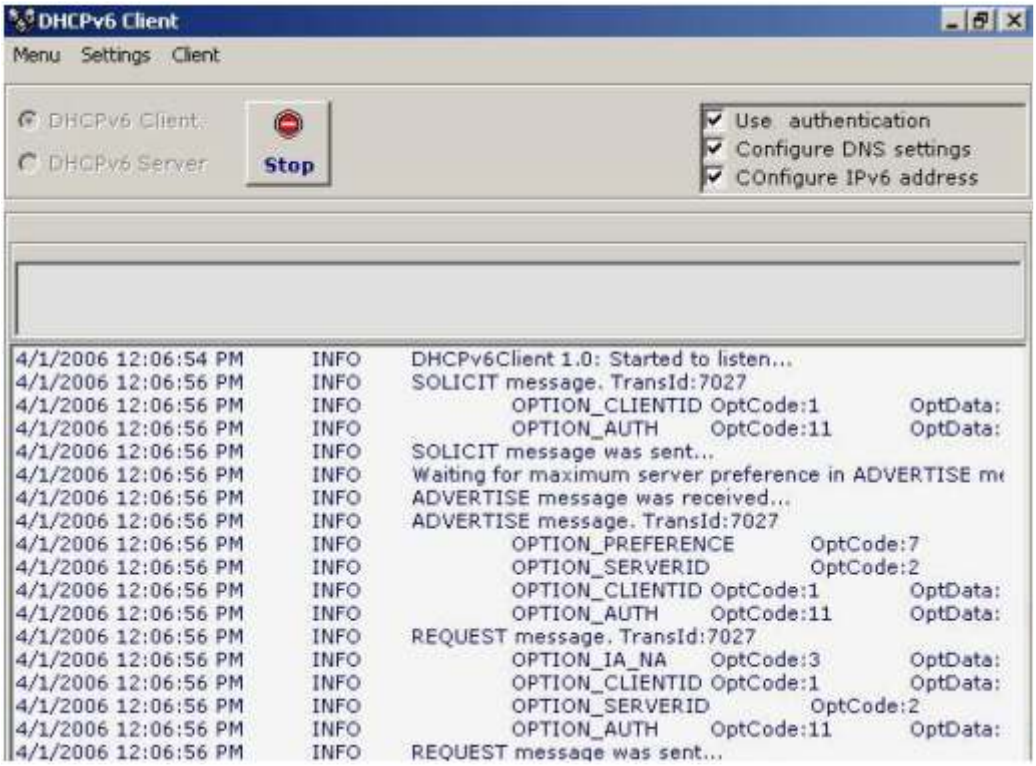
Tasarlanan güvenli DHCPv6 istemci ve sunumcu yazılımları iki farklı uygulama tipinde gerçeklenmiştir. Birinci uygulama tipi Windows servisi olup, istemci ve sunumcu yazılımlarının sistem seviyesinde çalıştırılması sağlanmıştır. İkinci uygulama tipi ise kullanıcı arayüzlü Windows uygulamasıdır.



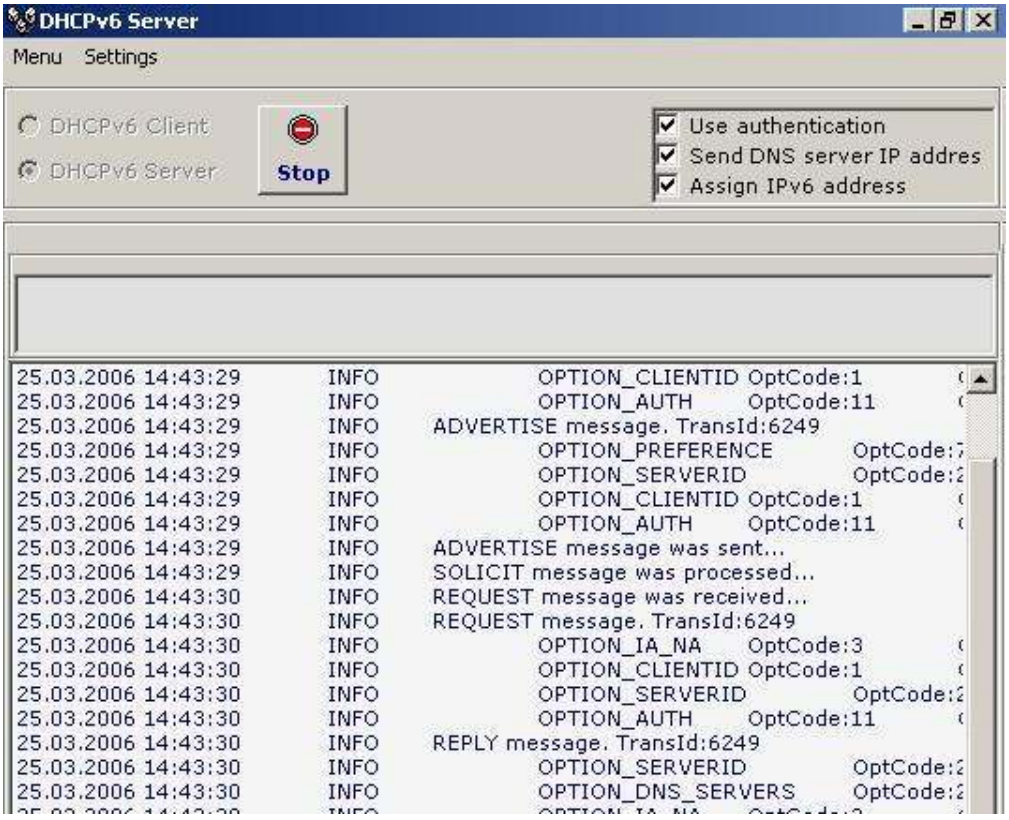
Şekil 4.1 İstemci ve sunumcu yazılımlarının servis olarak çalıştırılması

İstemci ve sunumcu yazılımları, ortak kullanılan portlar nedeni ile aynı bilgisayar üzerinde birlikte çalışamazlar. Şekil 4.1, geliştirilen DHCPv6 Windows servislerinin kurulum durumunu göstermektedir.

Windows uygulaması tipindeki istemci ve sunumcu yazılımları üzerinde sistemin çalışması test edilmiştir. Kullanıcı arayüzünde sağlanan istemci ve sunumcuya ait seçenekler ile konfigürasyon dosyalarına müdahale etmeden istemci ve sunumcu uygulamaları girilen parametrelerle anlık çalıştırılmış ve test edilmiştir. Uygulamaların çalışması esnasında çıktı olarak üretilen zaman ve işlem durum bilgileri performans ve uyumluluk testlerinde veri olarak kullanılmıştır.



Şekil 4.2 DHCPv6 istemcisi kullanıcı arayüzü



Şekil 4.3 DHCPv6 sunumcusu kullanıcı arayüzü

4.1 Geliştirme ve Test Ortamı

Güvenli DHCPv6 sistemi, Visual Studio .NET 2003 uygulama geliştirme ortamında, .NET Framework 1.1 üzerinde, C# programlama dili kullanılarak, nesneye yönelik programlama metodolojisine uygun olarak geliştirilmiştir.

Güvenli DHCPv6 sistemi, IPv6 uyumlu, Microsoft tabanlı işletim sistemleri üzerinde çalışabilmektedir. Sistem bileşenleri, aşağıdaki işletim sistemleri üzerinde test edilmiştir.

- MS Windows XP Professional (SP2)
- MS Windows 2003 Server

Gerçeklenen DHCPv6 sunumcu ve istemci yazılımları için, Windows XP (SP2) işletim sistemi üzerinde, 512 MB Ram'li, 100 Mbps'lık ağ arayüzüne sahip Pentium-IV işlemcili kişisel bilgisayarlar üzerinde, Windows form uygulaması tipindeki sürümleri ile sınanmıştır.

4.2 DHCPv6 Uyumluluk Testleri

Gerçeklenen güvenilir DHCPv6 sisteminin DHCPv6 uyumluluk testleri, New Hampshire Üniversitesi bünyesinde kurulan IPv6 Konsorsiyumu tarafından, IPv6 uygulamalarının uyumluluk testleri için hazırlanan uyumluluk test dokümanları göz önüne alınarak yapılmıştır [UNH]. DHCPv6 uyumluluk test sonuçları Çizelge 4.1'de verilmiştir.

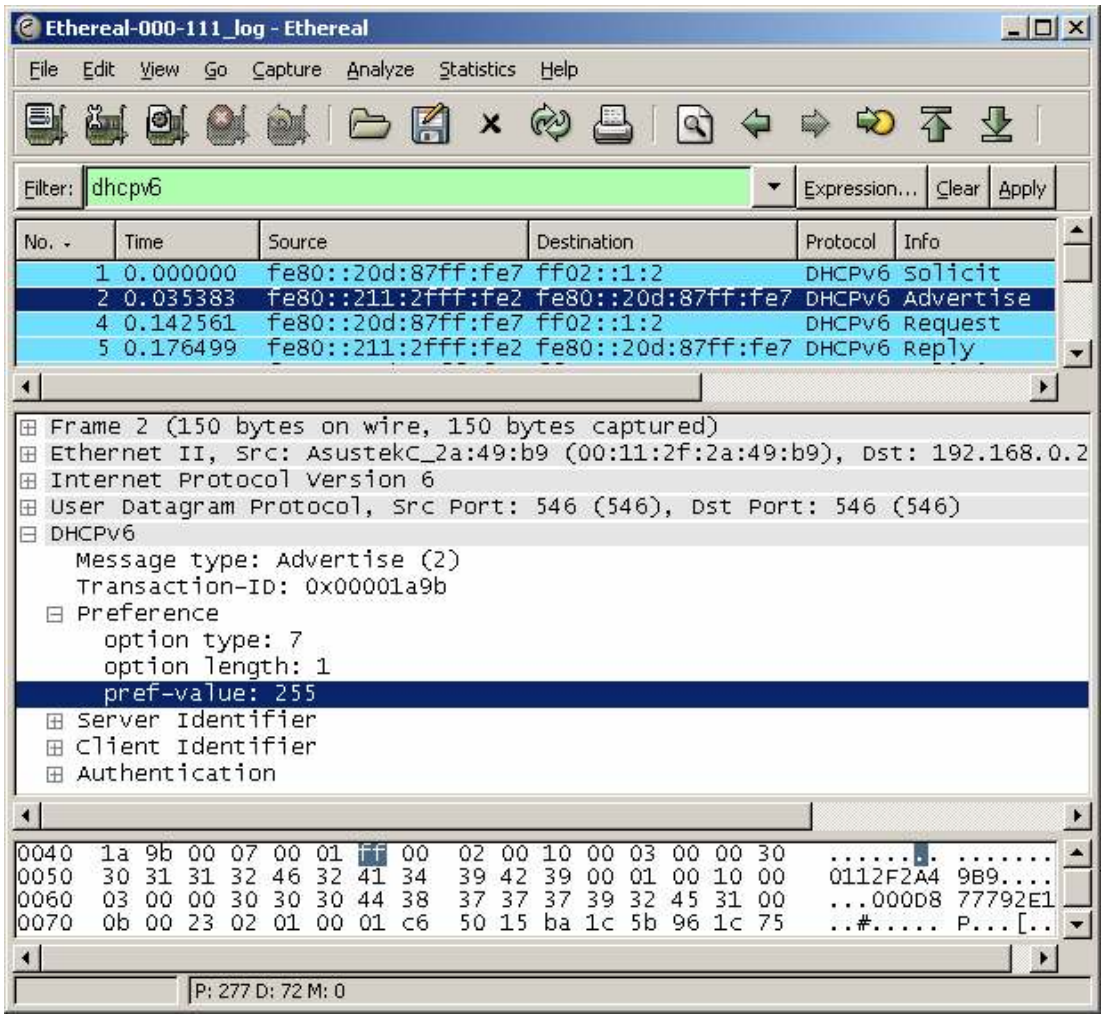
Çizelge 4.1 DHCPv6 uyumluluk test sonuçları

Uyumluluk test senaryoları	**Test sonuçları**
İstemci göndereceği mesajları RFC-3315'te belirtilen FF02::1:2 çoklu yayın adresini kullanarak göndermeye çalışır.	**Başarılı** (İstemci gönderdiği bütün mesajlar için hedef adres olarak FF02::1:2 adresini göstermektedir.)
İstemci, 546 nolu portu dinleyerek kendine gelen mesajları işler.	**Başarılı** (İstemcinin 546 nolu portu dinlemektedir.)
Sunumcu, 547 nolu portu dinleyerek kendine gelen mesajları işler.	**Başarılı** (Sunumcunun 547 nolu portu dinlemektedir.)

İstemci, kendine gelen Solicit, Request, Confirm, Renew, Rebind, Decline, Release, Information-Request, Relay-forward ve Relay-reply mesajlarını dikkate almaz ve cevaben hiç bir paket göndermez.	**Başarılı** (İstemci, ilgili mesajları aldığında herhangi bir işlem yapmamaktadır.)
Sunumcu kendine gelen, Advertise, Reply ve Reconfigure mesajlarını dikkate almaz ve cevaben hiç bir paket göndermez.	**Başarılı** (Sunumcu, ilgili mesajları aldığında hiç bir işlem yapmamaktadır)
İstemci, kendine gönderilen Advertise mesajının doğruluğunu ve geçerliliğini kontrol eder.	**Başarılı** (İstemci, hatalı veya geçersiz Advertise mesajını dikkate almamaktadır.)
İstemci, kendine gönderilen Reply mesajının doğruluğunu ve geçerliliğini kontrol eder.	**Başarılı** ((İstemci, hatalı veya geçersiz Reply mesajını işleme almamaktadır.)
İstemci, kendine gönderilen geçersiz Reconfigure mesajlarını dikkate almaz.	**Başarılı** (İstemci, geçersiz Reconfigure mesajlarını gözardı etmektedir)
İstemci, sunumcu keşfi için geçerli Solicit mesajı gönderir.	**Başarılı** (İstemci, sunumcu tespiti için Solicit mesajı göndermektedir.)
İstemci, Advetise mesajlarını öncelik değeri 255 olan mesaj gelene kadar veya zaman aşımına uğrayıncaya kadar biriktirir ve gelen Advertise mesajları içerisinde en büyük öncelik değerine sahip mesaj üzerinden işlemine devam eder.	**Başarılı** (İstemci, kendine gönderilen Advertise mesajlarından önceliği büyük olanı işleme almaktadır. Öncelik değeri 255 ten küçük olan Advertise mesajları için gelen mesajları biriktirmekte ve en büyük öncelikli mesajı işleme almaktadır.)
İstemci, geçerli Request mesajı oluşturur ve gönderir.	**Başarılı** (İstemci geçerli Request mesajları göndermektedir.)
İstemci, ağını değiştirdiği durumda geçerli bir Confirm mesajı göndererek ayarlarını kontrol eder.	**Başarılı** (İstemci yeniden başlatıldığında mevcut IPv6 adresinin içinde bulunduğu ağa uygun olup olmadığını kontrol etmekte, uygun değilse yeni adres talebinde bulunmaktadır.)
İstemci, sahip olduğu IP adresinin geçerlilik süresini artırmak için geçerli bir Renew mesajı gönderir.	**Başarılı** (İstemcinin Renew mesajı göndermekte ve gelen mesajı işleme almaktadır.)
İstemci, IP adresi almadan konfigürasyon bilgilerini elde etmek için geçerli bir Information-Request mesajı gönderir.	**Başarılı** (İstemci sadece konfigürasyon bilgilerini alacak şekilde ayarlanmış ve Information-Request mesajı gönderdiği gözlemlenmiştir.)
İstemci, kullanmakta olduğu IP adresinin iadesi için geçerli bir Release mesajı gönderir.	**Başarılı** (İstemcinin Release mesajı gönderdiği ve sunumcunun gelen mesajı IP adres havuzuna geri koyduğu gözlemlenmiştir.)
İstemci, kendine gönderilen IP adreslerinin geçerliliğini kontrol eder, geçersiz IP adresleri için Decline mesajını gönderir.	**Başarılı** (İstemci kendine gönderilen IPv6 adresinin kullanımda olup olmadığını kontrol eder, kullanımda ise başka bir adres talebinde bulunur.)

İstemci, gönderilen geçerli Advertise mesajlarını işler.	**Başarılı** (İstemci geçerli Advertise mesajlarını işlemektedir.)
İstemci, kendine gönderilen geçerli Reply mesajlarını işler.	**Başarılı** (İstemci geçerli Reply mesajlarını işlemektedir.)
İstemci, kendine gönderilen geçerli Reconfigure mesajlarını işler.	**Başarılı** (İstemci geçerli Reconfigure mesajlarını işlemektedir.)

Uyumluluk testlerinde, açık kaynak kodlu Ethereal programı [Ethereal] kullanılarak DHCP mesaj paketleri incelenmiş, RFC-3315 [Droms at al, 2003] ve RFC-3646 [Droms, 2003] için uyumluluk kontrolleri yapılmıştır. Şekil 4.4'te DHCPv6 Advertise mesajına ait paketin Ethereal programı ile incelenmesi sunulmuş olup benzer mesaj inceleme işlemleri gerçeklenen bütün mesaj tipleri için uygulanmıştır.



Şekil 4.4 Ethereal programı ile DHCPv6 Advertise mesajının incelenmesi

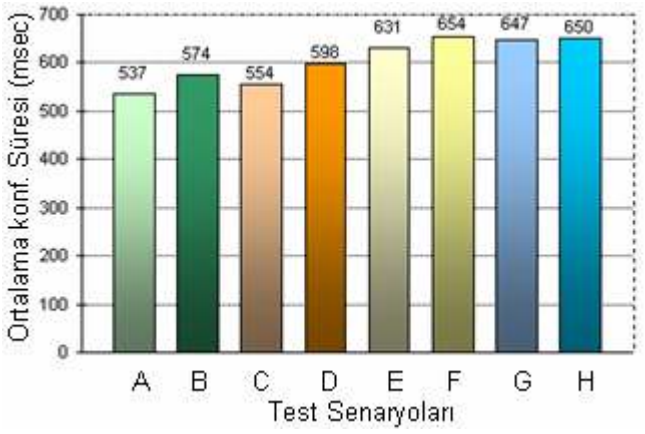
İncelenen mesaj paketlerinin, Ethereal programı tarafından otomatik olarak alanlarına ayrıştırılması sonucu gerçeklenen DHCPv6 mesajlarının standartlara uygun olduğu gözlemlenmiştir.

4.3 Performans Testleri

Gerçeklenen istemci ve sunumcu uygulamaları 3 adet fonksiyonel seçeneğe sahiptir. Bunlar IPv6 adresi gönder/al, DNS adreslerini gönder/al ve güvenli mesajlaş seçenekleridir. Bu 3 opsiyon ile 8 farklı test senaryosu elde edilebilir.

İstemci tarafında, düşük seviyeli işler (IP adresi ataması ve DNS ayarlarının yapılması) işletim sistemi komutları ile yapıldığından performans testleri, mesajlaşma performansı ve istemci konfigürasyon performansı olarak ikiye ayrılmış, her test için yukarıda bahsedilen 8 farklı test senaryosu 100'er kez tekrarlanarak performans verileri elde edilmiştir. Çizelge 4.2 ve 4.3'te verilen A, B, C, D, E, F, G ve H test senaryolarından B, D, F ve H test senaryoları sırası ile A, C, E ve G senaryolarının güvenlik modülü devreye alınmış durumlarıdır.

Mesajlaşma performanslarının ölçülmesinde işletim sistemi seviyesinde çalıştırılan komutlar devre dışı bırakılarak, istemci ve sunumcu arasındaki mesajlaşma performansları ve güvenlik modülünün etkileri incelenmiştir.



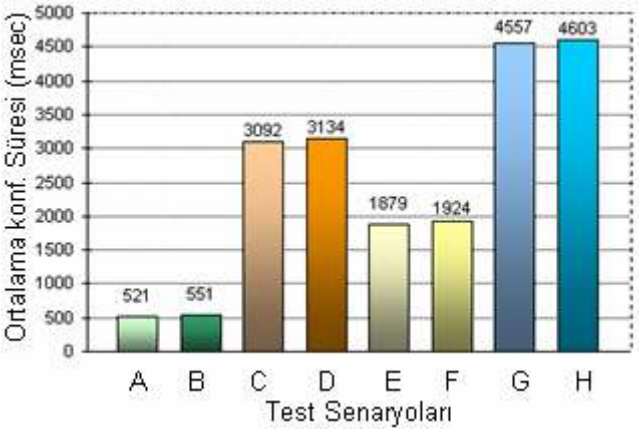
Şekil 4.5 Ortalama istemci-sunumcu mesajlaşma süresi

Mesajlaşma performans testlerine ilişkin konfigürasyon ve test verileri Çizelge 4.2'de verilmiştir.

Çizelge 4.2 Mesajlaşma testleri konfigürasyon ve test verileri

Senaryo	Fonksiyonel seçenekler			Test verileri (Konfigürasyon süreleri)			
	IP adresi al	DNS ayarı yap	Güvenli mesajlaş	Max süre	Min süre	Ortalama süre	Standart sapma
A	-	-	-	1021	441	537	80
B	-	✓	✓	1072	491	574	91
C	-	-	-	1062	491	554	61
D	-	✓	✓	1783	451	598	137
E	✓	-	-	1392	511	631	112
F	✓	✓	✓	1242	601	654	66
G	✓	-	-	741	561	647	45
H	✓	✓	✓	921	591	650	39

Konfigürasyon performans testlerinde, işletim sistemi seviyesindeki komutlar devreye alınarak istemcinin ortalam konfigürasyon süresi ve güvenlik modülünün etkileri incelenmiştir.



Şekil 4.6 Ortalama istemci konfigürasyon süresi (fonksiyonel)

Konfigürasyon performans testlerine ilişkin konfigürasyon ve test verileri Çizelge 4.3'te verilmiştir.

Çizelge 4.3 Fonksiyonel test konfigürasyonları ve test verileri

Senaryo	Fonksiyonel seçenekler			Test verileri (Konfigürasyon süreleri)			
	IP adresi al	DNS ayarı yap	Güvenli mesajlaş	Max süre	Min süre	Ortalama süre	Standart sapma
A	-	-	-	971	431	521	69
B	-	✓	✓	851	461	551	38
C	-	-	-	3876	2984	3092	86
D	-	✓	✓	4046	3024	3134	96
E	✓	-	-	2313	1783	1879	64
F	✓	✓	✓	2594	1833	1924	82
G	✓	-	-	6528	4386	4557	209
H	✓	✓	✓	6684	4426	4603	215

Her iki performans testinde de, B, D, F ve H test test sonuçları sırası ile A, C, E ve G testlerinin sonuçları ile karşılaştırıldığında, ortalama konfigürasyon sürelerinin birbirlerinden çok farklı olmadıkları ve aralarındaki farkın standart sapma değerlerinden küçük oldukları gözlemlenmiştir. Bu durum, güvenlik mekanizmasının performansa olan etkisinin göz ardı edilebilecek değerlerde olduğunu göstermektedir.

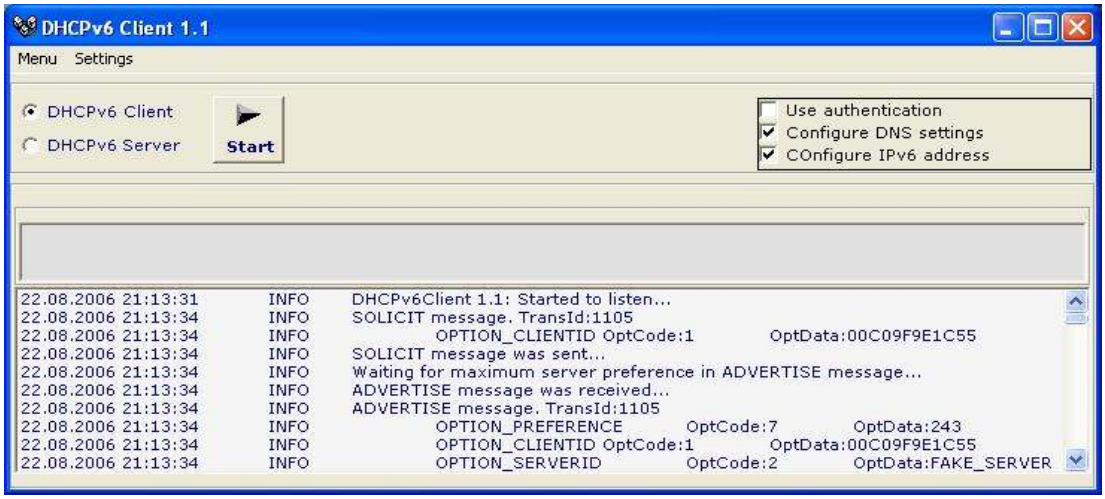
Performans testleri sonucunda, güvenlik mekanizmasının devreye alınmasının sistem performansına olan etkisinin gözardı edilebilecek değerlerde olduğu gözlemlenmiştir.

4.4 Güvenlik Testleri

Güvenlik testleri esnasında, DHCP istemcilerine saldırmak için, gerçeklenen DHCPv6 sunumculardan biri hatalı konfigüre edilerek sahte DHCP sunumcusu olarak çalıştırılmıştır. DHCP sunumcusuna saldırıda bulmak için de DHCPv6 mesajları üreten bir uygulama geliştirilmiş ve bazı istemciler geçersiz istemci olarak tanımlanarak çalıştırılmıştır.

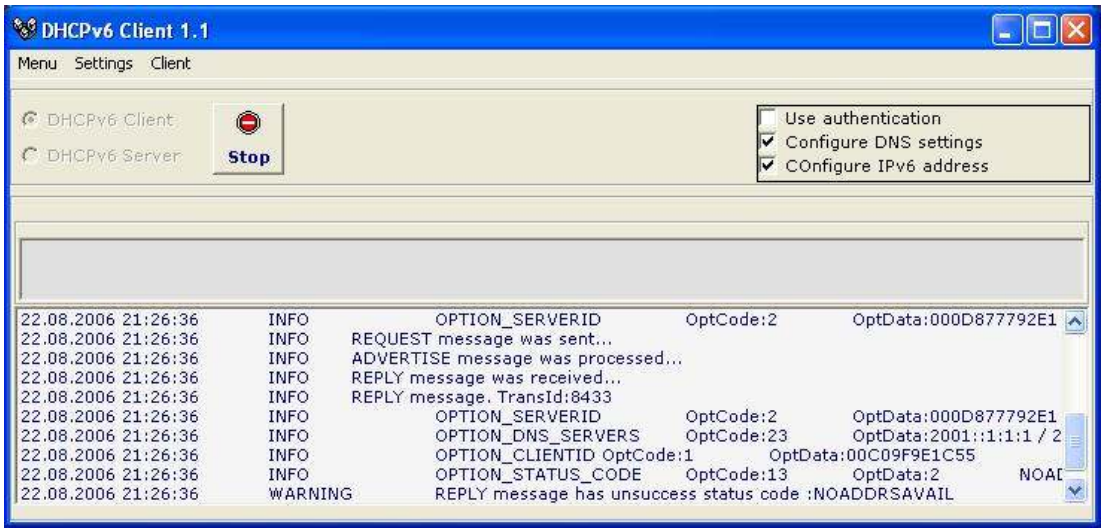
İstemci tarafında güvenlik mekanizması devre dışı bırakıldığında, sahte DHCP sunumcusunun asıl DHCP sunucusundan hızlı cevap döndüğü durumlarda, istemci sahte DHCP sunumcusundan hizmet almıştır. İstemci, sahte DHCP sunumcusundan

gelen RECONFIGURE mesajını kabul etmiş, yeni ağ ayarlarını sahte DHCP sunumcusundan aldığı bilgilere göre değiştirmiştir (Şekil 4.7).



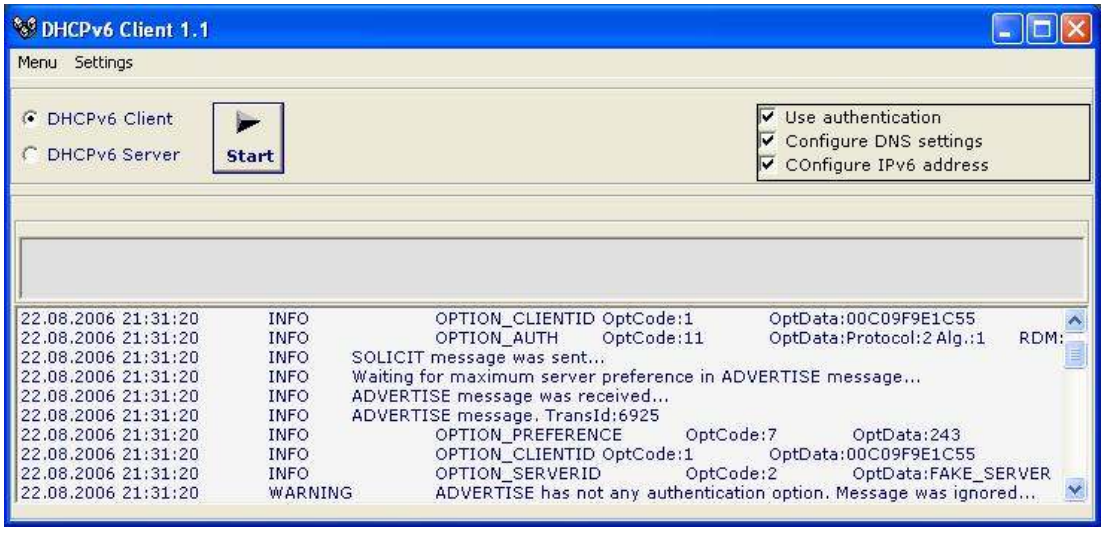
Şekil 4.7 İstemcinin sahte sunumcudan gelen mesajları kabul etmesi

Sunumcu tarafında güvenlik mekanizması devre dışı bırakıldığında, sunumcu ağdaki geçersiz istemcilere de hizmet vermiştir. DHCP mesaj üretici uygulaması tarafından sunumcu IP havuzundaki IP adreslerini tüketmek için gönderilen REQUEST mesajları işleyerek IP adres havuzundaki IP adreslerini tüketmiş, geçerli istemcilerin IP adres taleplerine cevap veremeyecek hale gelmiştir (Şekil 4.8). Geçerli bir istemciye gönderilen IP adresi için DHCP mesaj üretici uygulaması tarafından gönderilen RELEASE mesajı sunumcu tarafından işlenerek IP adres havuzuna geri bırakılmış, daha sonra kendinden IP adresi isteyen istemciye kullanımda olan IP adresini göndererek istemcinin konfigürasyon süresinin uzamasına neden olmuştur.



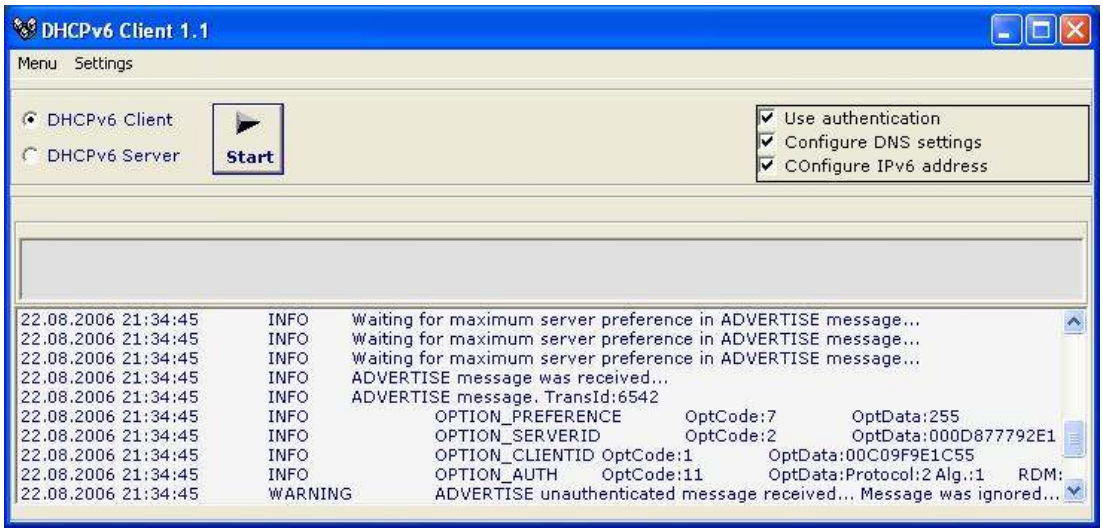
Şekil 4.8 Sunumcunun IP adres havuzunun boşaltılması

İstemci tarafında güvenlik mekanizması devreye alındığında, sahte sunumcudan gelen mesajlar doğrulanamadığı için iptal edilmiş ve geçerli sunumcu ile mesajlaşma yapıldığı gözlemlenmiştir. Sahte sunumcu tarafında güvenlik modülü devre dışı bırakılmış ise gönderdiği mesajlarda, mesaj doğrulama bilgisi olmadığı için mesaj istemci tarafında iptal edilmiştir (Şekil 4.9).



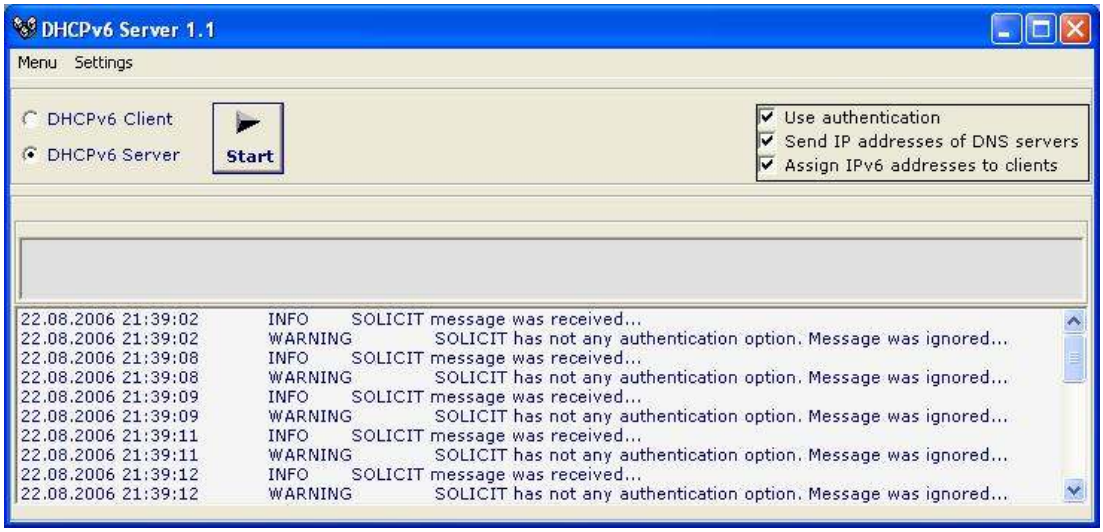
Şekil 4.9 İstemcinin geçersiz sunumcudan gelen mesajı iptal etmesi

Sahte sunumcu tarafında güvenlik modülü devrede olduğu durumda, istemciye gönderilen mesajdaki mesaj doğrulama bilgisi ile istemcinin hesapladığı mesaj doğrulama bilgileri birbiri ile uyuşmadığı için istemci bu durumda da gelen mesajı iptal etmiştir (Şekil 4.10).



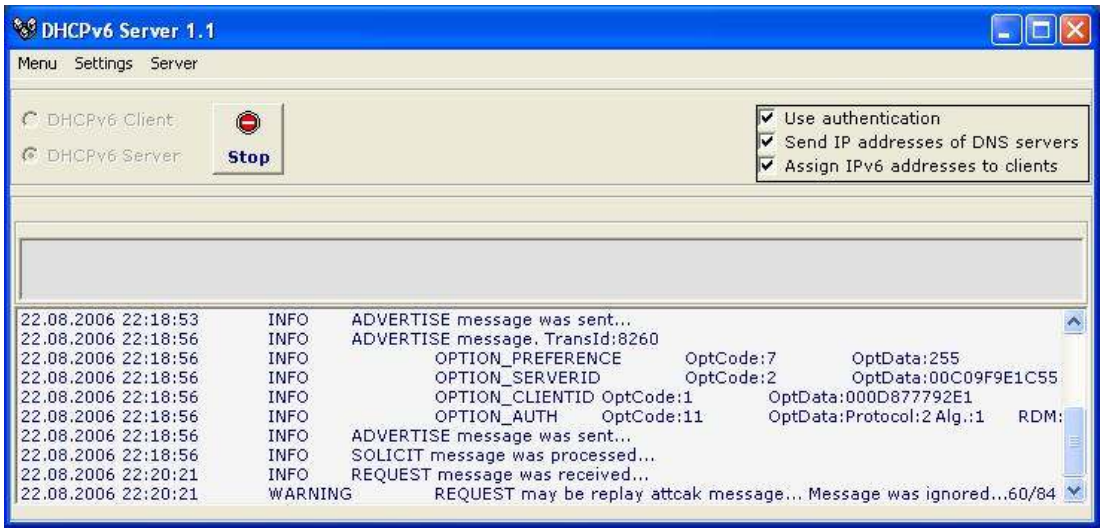
Şekil 4.10 İstemcinin sahte DHCP sunumcudan gelen mesajı iptal etmesi

Sunumcu tarafında güvenlik mekanizması devreye alındığında, geçersiz istemcilerden ve DHCP mesaj üreticisi tarafından gönderilen mesajlar doğrulanamadıkları için iptal edilmiştir (Şekil 4.11). Sunumcu sadece geçerli istemcilerden gelen mesajları işlemiştir.



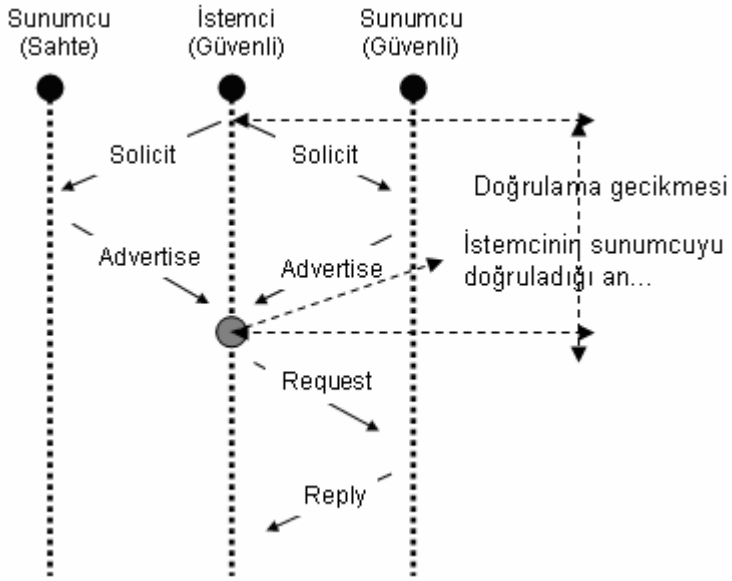
Şekil 4.11 Sunumcunun geçersiz istemciden gelen mesajları iptal etmesi

Sunumcuya daha önce geçerli bir istemci tarafından gönderilen mesaj, mesaj üretici tarafından kopyalanarak gönderildiğinde, sunumcu gelen mesajın doğrulamasını yapmış fakat mesaj tekrarlama yöntemi ile gönderilen bir mesaj olabileceği hesap edilerek mesaj iptal edilmiştir (Şekil 4.12).



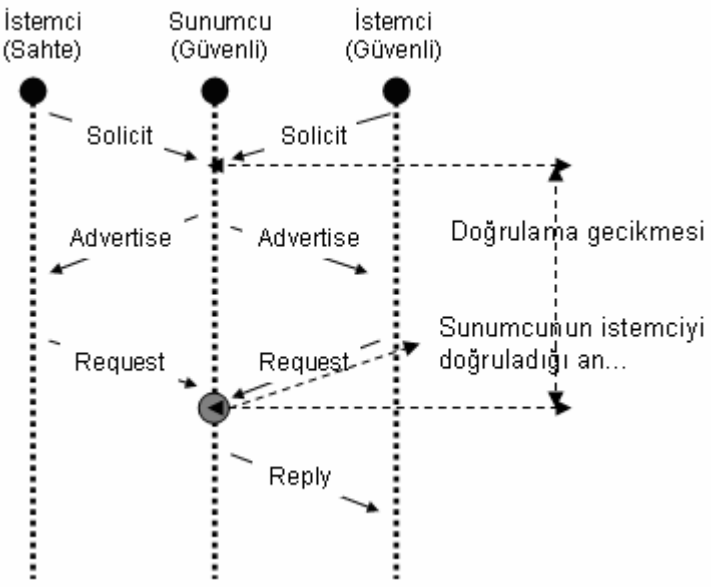
Şekil 4.12 Sunumcunun olası mesaj tekrarlama saldırılarına karşı tedbir alması

Özet olarak, Şekil 4.13'te güvenli DHCPv6 istemcisinin, ağdaki güvenli ve sahte DHCPv6 sunumcuları ile olan mesajlaşmaları verilmiştir. İstemcinin, sahte sunumcuyu tespit etmesi ve güvenli sunumcudan hizmet alması gösterilmiştir.



Şekil 4.13 Güvenli DHCPv6 istemcisinin mesajlaşmaları

Şekil 4.14'te güvenli DHCPv6 sunumcusunun ağdaki güvenli ve sahte DHCPv6 istemcileri ile olan mesajlaşmaları, sunumcunun sahte istemcileri tespit etmesi ve güvenli istemcilere hizmet vermesi gösterilmiştir.



Şekil 4.14 Güvenli DHCPv6 sunumcusunun mesajlaşmaları

Güvenlik testleri sonucunda, istemci ve sunumcu tarafında güvenlik mekanizması devre dışı bırakıldığında Bölüm 2.3'te bahsedilen güvenlik açıkları kullanılarak yapılan saldırılara karşı sistemin, diğer sistemler gibi savunmasız olduğu gözlemlenmiştir. Güvenlik mekanizmasının devreye alınması durumunda, yapılan saldırılar tespit edilerek, saldırı maksatlı gönderilen mesajlar istemci ve sunumcu tarafından iptal edilerek sistemin korunduğu tespit edilmiştir.

4.5 Mevcut DHCPv6 Sistemleri ile Karşılaştırma

Gerçeklenen "Güvenli DHCPv6 Sistemi" ile incelenen diğer açık kaynak kodlu DHCPv6 sistemleri arasında yapılan karşılaştırmalar Çizelge 4.4'te verilmiştir.

Çizelge 4.4 Mevcut DHCPv6 sistemlerinin karşılaştırılması

Sistem Özellikleri	**WIDE-DHCPv6 (KAME-DHCPv6)**	**Dibbler**	**Güvenli DHCPv6 Sistemi**
RFC uyumluluğu (Kısmen)	RFC-3315, RFC-3319, RFC-3633, RFC-3646	RFC-3315, RFC-3736, RFC-3898, RFC-3646	RFC-3315, RFC-3118, RFC-3646
DHCP elemanları	İstemci, Sunumcu ve Ajan	İstemci, Sunumcu ve Ajan	İstemci ve Sunumcu
Uyumlu işletim sistemleri	Linux, BSD (FreeBSD, OpenBSD, NetBSD)	Linux, Windows XP, Windows 2003	Windows XP, Windows 2003
Açık kaynak kod	✓	✓	✓
Çoklu sunumcu desteği	✓	✓	✓
IPv6 adres tahsisi	–	✓	✓
Güvenlik mekanizması	–	–	✓
Konfigürasyon arayüzü	–	–	✓
Yardımcı dokümanlar	–	✓ * Geliştirme klavuzu	✓ * Geliştirme klavuzu * Kod içi açıklamalar * MSDN formatlı yardım * UML şemaları * Akış şemaları

Gerçeklenen “Güvenli DHCPv6 Sistemi”, güvenlik mekanizması ve istemci-sunumcu konfigürasyonu için sağladığı kullanıcı arayüzleri ile diğer sistemlerde olmayan özelliklere sahiptir. Ayrıca, uygulama geliştiriciler için sunduğu yardımcı dokümanlar, sağlam altyapı ve anlaşılır kaynak kodu ile geliştirilmeye uygun bir sistemdir.

5 SONUÇLAR

DHCP protokolü, birçok güvenlik açığına sahip olmasına rağmen sağladığı kolaylıklar nedeni ile günümüzde yaygın olarak kullanılmaktadır. Fiziksel güvenliğin sağlandığı yerel alan ağlarında DHCP istemci ve sunumcuları için iç tehditler dışında çok fazla risk olmamasına karşın, özellikle kablosuz ağlarda fiziksel güvenliğin olmaması nedeni ile dış unsurlar çok büyük riskler doğurmaktadır.

Günümüzde, DHCP sistemlerinin birçoğunda kontrol-denetim maksatlı geliştirilen özellikler mevcut güvenlik açıklarını kapatmamaktadır. DHCP sistemlerinde güvenlik ancak protokol bazında uygulanacak güvenlik yöntemleri ile sağlanabilir.

Her sistemde olduğu gibi, bu sistemde de güvenlik unsuru gerek performans gerekse yönetilebilirlik açısından sıkıntılar doğurmasına karşılık, gerçeklenen sistemde bu sıkıntılar en aza indirgenmiştir. Varsayılan anahtar yöntemi ile anahtar paylaşımına ilişkin sorunlar giderilmiştir. Güvenlik mekanizmasının performansa olan etkisi göz ardı edilebilecek değerlerdedir.

Gerçeklenen “Güvenli DHCPv6 Sistemi”, ilave ek opsiyonel bilgilerin ve mesajların geliştirilmesi için gerekli olan alt yapıya sahip olduğundan; yeni opsiyonel bilgilerin ve mesajların sisteme dâhil edilmesinde büyük kolaylıklar sağlamaktadır. DHCPv6 protokolünde opsiyonel bilgilerin çeşitliliği ile ilgili çalışmalar devam etmekte olup, gerek mevcut opsiyonların ve gerekse yeni tasarlanan opsiyonların sisteme eklenmesi, sistemi daha kullanışlı hale getirecektir.

Gerçeklenen “Güvenli DHCPv6 Sistemi”, protokol bazında güvenliği sağlayarak mevcut güvenlik açıklarını kapatıp DHCP saldırılarına karşı korunaklıdır.

KAYNAKLAR

1. W. J. Croft, J. Gilmore, “Bootstrap Protocol”, RFC 951, IETF, September 1985
2. R. Droms, “Dynamic Host Configuration Protocol”, RFC 1541, IETF, October 1993
3. R. Droms, “Dynamic Host Configuration Protocol”, RFC 2131, IETF, March 1997
4. D. Comer, “Bootstrapping with BOOTP and DHCP”, The Internet Protocol Journal, Vol. 5, No.2, Page(s):24-31, June 2002
5. S. Deering, R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification” RFC 1883, December 1995
6. O. Annala, “The Hot Topics in DHCP Protocol Development”, May 2004, http://www.cs.helsinki.fi/u/kraatika/courses/IPsem04s/DHCP_HotTopics.pdf
7. R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, M. Carney, “Dynamic Host Configuration Protocol for IPv6” RFC 3315, July 2003
8. R. Droms, W. Arbaugh, “Authentication for DHCP Messages”, RFC 3118, IETF, June 2001.
9. C. E. Perkins, J. Bound, “DHCP for IPv6”, Computers and Communications, 1998. ISCC '98. Proceedings. Third IEEE Symposium, Page(s): 493-497, July 1998
10. KAME Project, http://www.tahi.org, http://wide-dhcpv6.sourceforce.net, http://www.freshports.org/net/dhcpv6
11. T. Mrugalski, “Dibbler – a portable Dynamic Host Configuration Protocol for IPv6 implementation”, The 8th International Conference on Telecommunications, Contel’2005, June 2005
12. R. Droms, Ed., “DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, RFC 3646, IETF, December 2003
13. S. Thomson, T. Narten, “IPv6 Stateless Address Autoconfiguration” RFC 2462, December 1998
14. R. Hibbs, C. Smith, B. Volz, M. Zohar, “Dynamic Host Configuration Protocol for IPv4 (DHCPv4) Threat Analysis”, Internet-Draft, IETF, June 2003

15. N. Tanaka, K. Fukuda, H. Nakada and H. Shimokawa “Development of PC Quarantine System”, NEC Journal of Advanced Technology, Vol.2, No.1, Page(s):47-52, 2005
16. M. Patrick, “DHCP Relay Agent Information Option.”, RFC 3046, IETF, January 2001.
17. K. Hornstein, T. Lemon, B. Aboba, J. Trostle “DHCP Authentication via Kerberos V.”, Internet Draft (c) The Internet Society, November 2000.
18. G. Glazer, C. Hussey, R. Shea “Certificate-Based Authentication for DHCP”, March 2003, http://www.cs.ucla.edu/ chussey/proj/dhcp cert/cbda.pdf
19. T. Komori, T. Saito, “The Secure DHCP System with User Authentication”, Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference. Page(s): 123 -131, ISBN 0-7695-1591-6, November 2002
20. H. Ju, J. Han, “DHCP Message Authentication with an Effective Key Management”, Transactions on Engineering, Computing and Technology, Enformatika V8, ISSN 1305-5313, October 2005
21. R. Rivest, “The MD5 Message-Digest Algorithm", RFC 1321, IETF, April 1992
22. NDoc Code Documentation Generator for .NET, http://ndoc.sourceforge.net
23. IPv6 Consortium, Interoperability Laboratory, Research Computing Center, University of New Hampshire, http://www.iol.unh.edu/testsuites/IPv6
24. Ethereal: A Network Protocol Analyzer, http://www.ethereal.com

ÖZGEÇMİŞ

9 Eylül 1978'de Ardahan'ın Çıldır ilçesine bağlı Kayabeyi köyünde dünyaya geldi. İlk ve orta öğrenimini Çıldır'da, lise öğrenimini Kartal'da tamamladı. 1995 yılında başladığı İstanbul Üniversitesi Mühendislik Fakültesi, Bilgisayar Bilimleri Mühendisliği Bölümünden Temmuz 1999'da mezun oldu. 1998–2000 yılları arasında Transvaro Elektron Aletleri A.Ş.'de Ar-Ge mühendisi olarak çalıştı. 2000–2003 yılları arasında YBS A.Ş.'de yazılım mühendisi olarak çalıştı. 2001 yılında Gebze Yüksek Teknoloji Enstitüsü, Mühendislik ve Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği bölümünde yüksek lisans öğrenimine başladı. 2004 yılında askerlik hizmetini Genelkurmay Bilgi Sistemleri Daire Başkanlığı'nda, OBİ subayı olarak tamamladıktan sonra Havelsan A.Ş.'de yazılım uzman mühendisi olarak çalışmaya başladı.

EKLER

Ek 1, DHCPv6 İstemcisi Kaynak Kodları

```
/*-------------------------------------------------------------------+
| Gursoy DURMUS                                                      |
| gdurmus@yahoo.com, gdurmus@gmail.com                               |
| Gebze Institude of Technology, Computer Engineering Department     |
| http://bilmuh.gyte.edu.tr                                          |
| ================================================================== |
|                                                                    |
| Revisions:                                                         |
|                                                                    |
+-------------------------------------------------------------------*/

using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Collections;

using DHCPv6Lib.Messages;
using DHCPv6Lib.Constants;
using DHCPv6Lib.Utilities;
using DHCPv6Lib.Options;
using DHCPv6Lib.ConfigManager;
using DHCPv6Lib.KeyManagement;

namespace DHCPv6Lib.Logic
{
      /// <summary>
      /// Implementation of the DHCPv6Client logic.
      /// </summary>
      public class DHCPClient:DHCPNode
      {
            //constants...
            private const byte REPLY_FOR_REQUEST =
DHCPMessages.REQUEST;
            private const byte REPLY_FOR_CONFIRM =
DHCPMessages.CONFIRM;
            private const byte REPLY_FOR_RELEASE =
DHCPMessages.RELEASE;
            private const byte REPLY_FOR_RENEW   =
DHCPMessages.RENEW;


            //class members
            private ClientInfo cInfo = new ClientInfo();
            private IConfigManager configManager;
            private ArrayList alAdvertises = new ArrayList();
            private Thread thrdProcessAdvertise;
            private Thread thrdProcessConfirm;
            private bool waitForMaxServerPref = true;
            private bool confirmApproved = false;


            /// <summary>
```

```
            /// Default constructer
            /// </summary>
            public DHCPClient(IConfigManager configManager)
            {
                  try
                  {
                        //load client settings...
                        Initialize();

                        //set client threads
                        thrdProcessAdvertise = new Thread(new
ThreadStart(WaitForMaxServerPref));
                        thrdProcessConfirm = new Thread(new
ThreadStart(WaitForConfirmReply));

                        //log config
                        Logger.appName ="Client";

                        //logic config
                        Utility.configMan = configManager;
                        KeyManager.cnfMangr = configManager;
                        this.configManager = configManager;
                        //authentication and other configs
                        this.useAuthentication =
(configManager.ReadData(ConfigLabels.COMMON_USE_AUTHENTICATION,true)
=="true");
                        this.configureDNS =
configManager.ReadData(ConfigLabels.COMMON_CONFIGURE_DNS,true)=="tru
e";
                        this.configureIPv6Address =
configManager.ReadData(ConfigLabels.COMMON_CONFIGURE_IPV6_ADDRESS,tr
ue)=="true";
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Warning,"DHCPv6Client
object could not created properly...");
                        Logger.Log(LogType.Error, e.ToString());
                  }

            }

            /// <summary>
            /// Loads client settings...
            /// </summary>
            private void Initialize()
            {
                  this.nodeName = "DHCPv6Client 1.0";
                  //port config
                  this.m_PortListen = UDPPorts.CLIENT_PORT;
                  this.m_PortSend = UDPPorts.SERVER_PORT;
                  //udpClient config
                  this.m_UdpClientListener = new UdpClient
(m_PortListen,AddressFamily.InterNetworkV6);
                  this.m_UdpClientSender = new UdpClient
(m_PortSend,AddressFamily.InterNetworkV6);

                  //join multicast group
                  IPAddress listenIPAddress =
IPAddress.Parse(MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_SERVERS
);
```

```
                  IPv6MulticastOption ipv6MulticastOption = new
IPv6MulticastOption(listenIPAddress,0);
                  IPAddress group =  ipv6MulticastOption.Group;
                  long interfaceIndex =
ipv6MulticastOption.InterfaceIndex;

                  //join the specified multicast group for receiving
datagrams

      m_UdpClientListener.JoinMulticastGroup((int)interfaceIndex,
listenIPAddress);

                  //remoteIPEndPoint config
                  remoteIPEndPoint = new
IPEndPoint(IPAddress.IPv6Any,m_PortListen);
            }

            /// <summary>
            /// Processes the incoming DHCPMessages
            /// </summary>
            /// <param name="dhcpMessage"></param>
            /// <param name="server"></param>
            protected override void ProcessMessage(DHCPMessage
msg,string source)
            {
                  //show valid message info
                  msg.ShowInfo();

                  //check message is valid
                  if(!IsValidMessage(msg))
                        return;

                  //process the valid message
                  switch(msg.GetMsgType())
                  {
                        case(DHCPMessages.ADVERTISE):
                              //get OPTION_PREFERENCE
                              DHCPOption optRet;
                              OptionPreference optPref=null;
                              optRet =
msg.GetOption(DHCPOptions.OPTION_PREFERENCE);
                              if(optRet==null)
                              {
                                    Logger.Log(LogType.Warning,
msg.GetName() + " does not have any OPTION_PREFERENCE option...
Message ignored...");
                              }
                              else{
                                    //-------------------------------
-------------------------------
                                    //select first received advertise
message, and process it...
                                    //ProcessAdvertiseMessage(msg);
                                    //Logger.Log(LogType.Info,
msg.GetName() +" message was processed...");
                                    //-------------------------------
-------------------------------
                                    //multiple server support
                                    //collect advertise messages
until time out occurs or max pref receives
                                    alAdvertises.Add(msg);
```

```
                                    optPref = new
OptionPreference(optRet);

      if(optPref.GetPreferenceValue()>=MaxValues.MAX_SERVER_PREF)
                                    {
                                          waitForMaxServerPref =
false;
                                    }
                                    /*
                                    //start thread if thread is not
started...

      if(thrdProcessAdvertise.ThreadState!=ThreadState.Running &&
thrdProcessAdvertise.ThreadState !=ThreadState.WaitSleepJoin)

                                                      {

      thrdProcessAdvertise.Start();
                                    }
                                    //wait for thread starts...

      while(thrdProcessAdvertise.ThreadState!=ThreadState.Running &&
thrdProcessAdvertise.ThreadState !=ThreadState.WaitSleepJoin);
                                    //-------------------------------
-------------------------------
                                    */
                              }
                              break;
                        case(DHCPMessages.REPLY):
                              ProcessReplyMessage(msg);
                              Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                              break;
                        case(DHCPMessages.RECONFIGURE):
                              ProcessReconfigureMessage(msg);
                              Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                              break;
                        default:
                              Logger.Log(LogType.Info, msg.GetName()
+" message process logic is not implemented, so message will not be
replied...");
                              Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                              break;
                  }
            }

            /// <summary>
            /// starts client process
            /// </summary>
            public override void Start()
            {
                  //start to listen...
                  m_Listen = true;
                  m_ThrdListen = new Thread(new
ThreadStart(Listen));
                  m_ThrdListen.Start();

                  //send solicit
                  //SendSolicitMessage();
```

```
      //.....................................................
.........
                  ArrayList alManualIPs =
Utility.GetManualIPvAddresses();
                  if(alManualIPs.Count>0)
                  {
                        //start confirm control thread
                        thrdProcessConfirm.Start();
                        cInfo.iTransID =
Utility.GetNewTransactionID();
                        //send confirm message for each ipv6 addr.
                        foreach(IPAddress ipv6Addr in alManualIPs)
                              SendConfirmMessage(ipv6Addr);
                  }
                  else{
                        SendSolicitMessage();
                  }

      //.....................................................
.........
            }


            /// <summary>
            /// stops client process
            /// </summary>
            public override void Stop()
            {
                  ShowStatus();

                  m_Listen = false;
                  this.m_UdpClientListener.Close();
                  this.m_UdpClientListener=null;
                  this.m_UdpClientSender.Close();
                  this.m_UdpClientSender=null;

                  //stop m_ThrdListen thread
                  try
                  {
                        if(m_ThrdListen!=null &&
m_ThrdListen.ThreadState ==ThreadState.Running)
                        {
                              m_ThrdListen.Abort();
                              m_ThrdListen =null;
                        }
                  }
                  catch(Exception e)
                  {
                        //TODO: ThreadAbortException: Thread was
being aborted.
                        //Logger.Log (LogType.Error,e.ToString());
                  }

                  //stop thrdProcessAdvertise thread
                  try
                  {
                        if(thrdProcessAdvertise!=null &&
(thrdProcessAdvertise.ThreadState ==ThreadState.Running ||
thrdProcessAdvertise.ThreadState ==ThreadState.WaitSleepJoin))
```

```
                        {
                              thrdProcessAdvertise.Abort();
                              thrdProcessAdvertise =null;
                        }
                  }
                  catch(Exception e)
                  {
                        //Logger.Log (LogType.Error,e.ToString());
                  }
                  //stop thrdProcessConfirm thread
                  try
                  {
                        if(thrdProcessConfirm!=null &&
(thrdProcessConfirm.ThreadState ==ThreadState.Running ||
thrdProcessConfirm.ThreadState ==ThreadState.WaitSleepJoin))
                        {
                              thrdProcessConfirm.Abort();
                              thrdProcessConfirm =null;
                        }
                  }
                  catch(Exception e)
                  {
                        //Logger.Log (LogType.Error,e.ToString());
                  }
            }

            /// <summary>
            /// checks message is valid for client
            /// </summary>
            /// <param name="msg"></param>
            /// <returns></returns>
            protected override bool IsValidMessage(DHCPMessage msg)
            {
                  DHCPOption opt;
                  OptionClientID optClntID;
                  OptionServerID optSrvID;
                  OptionAuthentication optAuth;

                  //check message is valid for Client
                  if(   !( msg.GetMsgType() ==
DHCPMessages.ADVERTISE
                        || msg.GetMsgType() == DHCPMessages.REPLY
                        || msg.GetMsgType() ==
DHCPMessages.RECONFIGURE
                        ))
                  {
                        Logger.Log(LogType.Warning, msg.GetName()+ "
message is not acceptable for the client and will be discarded...");
                        alIgnoredMessages.Add(msg);
                        return false;
                  }

                  //check message is valid
                  if(!msg.IsValid())
                  {
                        Logger.Log(LogType.Warning, msg.GetName()+ "
message is invalid...");
                        alIgnoredMessages.Add(msg);
                        return false;
                  }
```

```
                  //check ClientIDs
                  opt= msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                  if(opt==null)
                  {
                        Logger.Log(LogType.Warning, msg.GetName()+ "
message has not OPTION_CLIENTID option...");
                        alIgnoredMessages.Add(msg);
                        return false;
                  }
                  else
                  {
                        optClntID =  new OptionClientID (opt);

      if(optClntID.GetDUID()!=cInfo.optClntID.GetDUID())
                        {
                              Logger.Log(LogType.Warning,
msg.GetName()+ " message has not valid OPTION_CLIENTID option...");
                              alIgnoredMessages.Add(msg);
                              return false;
                        }
                  }

                  //check ServerIDs
                  opt= msg.GetOption(DHCPOptions.OPTION_SERVERID);
                  if(opt==null)
                  {
                        Logger.Log(LogType.Warning, msg.GetName()+ "
message has not OPTION_SERVERID option...");
                        alIgnoredMessages.Add(msg);
                        return false;
                  }
                  else
                  {
                        optSrvID =  new OptionServerID (opt);
                        if(cInfo.optSrvID!=null &&
cInfo.optSrvID.GetDUID()!=optSrvID.GetDUID())
                        {
                              Logger.Log(LogType.Warning,
msg.GetName()+ " message has not valid OPTION_SERVERID option...");
                              alIgnoredMessages.Add(msg);
                              return false;
                        }
                  }

                  //check transactionIDs
                  if(cInfo.iTransID !=msg.GetTransactinID())
                  {
                        Logger.Log(LogType.Warning, msg.GetName()+ "
message's transactionID is invalid...");
                        alIgnoredMessages.Add(msg);
                        return false;
                  }

                  //check message is expected message
                  if(cInfo.waitingFor != msg.GetMsgType())
                  {
                        Logger.Log(LogType.Warning, msg.GetName()+ "
unexpected message ignored...");
                        alIgnoredMessages.Add(msg);
                        return false;
                  }
```

```
                  //control authentication option
                  if(useAuthentication){
                        opt =
msg.GetOption(DHCPOptions.OPTION_AUTH);
                        if(opt==null){
                              Logger.Log(LogType.Warning,
msg.GetName()+ " has not any authentication option. Message was
ignored...");
                              alIgnoredMessages.Add(msg);
                              return false;
                        }
                        optAuth = new OptionAuthentication(opt);
                        //check replay detection...

      if(OptionAuthentication.GetCurrReplyDetection()-
optAuth.GetReplyDetection() > 60*10000000){
                              //replay attack may be in more than 1
minute
                              Logger.Log(LogType.Warning,
msg.GetName()+ " may be replay attcak message... Message was
ignored...");
                              alIgnoredMessages.Add(msg);
                              return false;
                        }
                        //check authentication is valid...
                        if(!msg.IsAuthenticated(cInfo.key)){
                              Logger.Log(LogType.Warning,
msg.GetName()+ " unauthenticated message received... Message was
ignored...");
                              alIgnoredMessages.Add(msg);
                              return false;
                        }
                  }
                  //message is valid
                  return true;
            }


            /// <summary>
            /// process the incoming advertise message
            /// </summary>
            /// <param name="advMsg"></param>
            /// <returns></returns>
            private int ProcessAdvertiseMessage(DHCPMessage msg)
            {
                  try
                  {
                        DHCPOption optRet;
                        //analyse the advertise
                        OptionServerID optSrvID=null;
                        OptionClientID optClntID=null;
                        //get OPTION_CLIENTID
                        optRet =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(optRet!=null)
                              optClntID = new
OptionClientID(optRet);
                        //get OPTION_SERVERID
                        optRet =
msg.GetOption(DHCPOptions.OPTION_SERVERID);
```

```
                        if(optRet!=null)
                              optSrvID = new OptionServerID(optRet);

                        //save selected ServerID
                        cInfo.optSrvID = optSrvID;

                        //create and send Request/InformationRequest
message
                        DHCPMessage reqMsg = null;
                        if(configureIPv6Address){
                              reqMsg = new RequestMessage
(msg.GetTransactinID());
                              OptionIA_NA optIANA = new
OptionIA_NA(0,OptionIA_NA.TIME_INFINITE,OptionIA_NA.TIME_INFINITE);
                              reqMsg.AddOption(optIANA);
                        }
                        else{
                              reqMsg = new InformationRequestMessage
(msg.GetTransactinID());
                        }
                        reqMsg.AddOption(optClntID);
                        reqMsg.AddOption(optSrvID);

                        //create authentcation option if necessary
                        if(useAuthentication){
                              reqMsg.AddAuthOption(cInfo.key);
                        }

                        //send request message

      Send(reqMsg,MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_SERVE
RS);

                        //change client status
                        cInfo.waitingFor = DHCPMessages.REPLY;
                        cInfo.reply_for = REPLY_FOR_REQUEST;
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }


            /// <summary>
            /// processes the incoming reply message
            /// </summary>
            /// <param name="rplyMsg"></param>
            /// <returns></returns>
            private int ProcessReplyMessage(DHCPMessage msg)
            {
                  try
                  {
                        DHCPOption optRet;
                        //check the Reply message has success code,
or not in message
                        OptionStatusCode optStat=null;
                        optRet =
msg.GetOption(DHCPOptions.OPTION_STATUS_CODE);
```

```
                        if(optRet!=null)
                              optStat = new
OptionStatusCode(optRet);

                        //--------------- REPLY_FOR_CONFIRM --------
------------------------
                        if(cInfo.reply_for==REPLY_FOR_CONFIRM)
                        {
                              //check confirm message is approved...
                              confirmApproved = optStat!=null &&
optStat.GetStatusCode()==StatusCodes.SC_RFC_SUCCESS;
                              if(!confirmApproved)
                              {

      DeleteCurrentIPv6Address(msg,false);
                              }
                              cInfo.iTransID=0;
                              cInfo.reply_for = REPLY_FOR_CONFIRM;
                              return 0;
                        }

                        //--------------- REPLY_FOR_RELEASE --------
------------------------
                        if(cInfo.reply_for==REPLY_FOR_RELEASE)
                        {
                              bool release_success = optStat!=null
&& optStat.GetStatusCode()==StatusCodes.SC_RFC_SUCCESS;
                              if(release_success)

      DeleteCurrentIPv6Address(msg,true);
                              cInfo.iTransID=0;
                              cInfo.reply_for = REPLY_FOR_CONFIRM;
                              return 0;
                        }

                        //--------------- REPLY_FOR_RENEW ----------
-------------------------
                        if(cInfo.reply_for==REPLY_FOR_RENEW)
                        {
                              //check message has success code
                              if(optStat!=null &&
optStat.GetStatusCode()==StatusCodes.SC_RFC_NOBINDING)
                              {
                                    //if address is not binded,
delete address

      DeleteCurrentIPv6Address(msg,false);
                                    //check IPv6 address assigned,
otherwise send Solicit message to obtain IPv6 address
                                    ArrayList al =
Utility.GetManualIPvAddresses();
                                    if(al.Count==0)
                                          SendSolicitMessage();
                              }
                              cInfo.iTransID=0;
                              cInfo.reply_for = REPLY_FOR_CONFIRM;
                              return 0;
                        }

                        //--------------- REPLY_FOR_REQUEST --------
------------------------
```

```
                        if(cInfo.reply_for==REPLY_FOR_REQUEST)
                        {
                              //check message has success code
                              if(optStat!=null &&
optStat.GetStatusCode()!=StatusCodes.SC_RFC_SUCCESS)
                              {

      Logger.Log(LogType.Warning,msg.GetName()+" message has
unsuccess status code :"+ optStat.GetStatusMessage());
                                    return 0;
                              }
                        }

                        //analyse the reply message and configure
the client ip settings
                        OptionIA_NA optIA_NA = null;
                        OptionDNSServers optDNS = null;
                        //get OptionIA_NA
                        optRet =
msg.GetOption(DHCPOptions.OPTION_IA_NA);
                        if(optRet!=null)
                              optIA_NA = new OptionIA_NA(optRet);
                        //get OPTION_DNS_SERVERS
                        optRet =
msg.GetOption(DHCPOptions.OPTION_DNS_SERVERS);
                        if(optRet!=null)
                              optDNS = new OptionDNSServers(optRet);

                        //get OptionIAAddr option
                        OptionIAAddr optIAAddr = null;
                        if(optIA_NA!=null)
                        {
                              foreach(DHCPOption opt in
optIA_NA.GetSubOptions())
                              {
                                    if(opt.GetOptionCodeValue() ==
DHCPOptions.OPTION_IAADDR)
                                          optIAAddr = new
OptionIAAddr(opt);
                              }
                        }

                        //process OptionIAAddr option
                        if(optIAAddr!=null && configureIPv6Address)
                        {
                              string sRet;
                              IPAddress ipAddr =
optIAAddr.GetIPv6Address();
                              //check IPAddress is already in use by
another node
                              if(!Utility.IsAddrAvailable(ipAddr))
                              {
                                    SendDeclineMessage(optIA_NA);
                                    Logger.Log(LogType.Info,"Declined
IPv6 Address : " + ipAddr.ToString() + " Because it is inuse by
another node");
                                    return 0;
                              }
                              //configure IPv6Address
                              //delete if exist on node (deteced in
upper line...)(BUG on detect)
```

```
                              //sRet=
DeleteIPv6Address(IPv6Context.Interface,ipAddr.ToString());
                              //Logger.Log(LogType.Info,"Deleted
IPv6 Address : " + ipAddr.ToString() + "\tRetCode:"+sRet);
                              //add address
                              sRet=
SetIPv6Address(IPv6Context.Interface,ipAddr.ToString());
                              Logger.Log(LogType.Info,"Configured
IPv6 Address : " + ipAddr.ToString() + "\tRetCode:"+sRet);
                        }

                        //process OptionDNSServers option
                        if(optDNS!=null && configureDNS)
                        {
                              string sRet;
                              //clear dns settings...
                              sRet =
DeleteIPv6Address(IPv6Context.DNS,"");
                              Logger.Log(LogType.Info,"Deleted DNS
Server settings..." + "\tRetCode:"+sRet);
                              foreach(IPAddress ipAddr in
optDNS.GetDNSServers())
                              {
                                    //add dns server
                                    sRet =
SetIPv6Address(IPv6Context.DNS,ipAddr.ToString());

      Logger.Log(LogType.Info,"Configured DNS Server : " +
ipAddr.ToString() + "\tRetCode:"+sRet);
                              }
                        }

                        //show configuration info
                        cInfo.dtEnd = DateTime.Now;
                        Logger.Log(LogType.Info,"Client operation
completed.... Start:\t"+cInfo.dtStart + "\tFinish:\t"+cInfo.dtEnd);
                        //change client status
                        cInfo.iTransID=0;
                        cInfo.waitingFor = DHCPMessages.RECONFIGURE;
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }

            /// <summary>
            /// defines IPv6 realms
            /// </summary>
            public enum IPv6Context
            {
                  Interface = 1,
                  DNS=2
            }

            /// <summary>
            /// Adds IPv6 address to interface or adds DNS server
IPv6 address to DNS list
            /// </summary>
```

```
            /// <param name="enum_dest"></param>
            /// <param name="ipv6Address"></param>
            /// <returns></returns>
            private string SetIPv6Address(IPv6Context
enum_dest,string ipv6Address)
            {
                  try
                  {
                        string cmd="",process="",parameters="";

                        //get command text
                        if(enum_dest == IPv6Context.Interface)
                              cmd =
configManager.ReadData(ConfigLabels.CLIENT_CMD_ADD_IPV6_ADDRESS,true
);
                        else if (enum_dest == IPv6Context.DNS)
                              cmd =
configManager.ReadData(ConfigLabels.CLIENT_CMD_ADD_DNS_ADDRESS,true)
;

                        if(cmd=="")
                              return "Unknown.";
                        //calculate process and parameters
                        cmd = cmd.Replace("INTERFACE_NAME",
"\""+configManager.ReadData(ConfigLabels.CLIENT_INTERFACE_NAME,true)
+"\"");
                        cmd =
cmd.Replace("IPv6_ADDRESS",ipv6Address);
                        process = cmd.Substring(0,cmd.IndexOf(" "));
                        parameters = cmd.Replace(process+" ","");
                        //run command
                        string temp =
Utility.RunProcess(process,parameters,false);
                        temp = temp.Replace("\r","\t");
                        temp = temp.Replace("\n","\t");
                        return temp;
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Info,e.ToString());
                        //return failed
                        return "Failed.";
                  }
            }


            /// <summary>
            /// Deletes the given IPv6 address from related context
            /// </summary>
            /// <param name="enum_dest"></param>
            /// <param name="ipv6Address"></param>
            /// <returns></returns>
            private string DeleteIPv6Address(IPv6Context
enum_dest,string ipv6Address)
            {
                  try
                  {
                        string cmd="",process="",parameters="";

                        //get command text
                        if(enum_dest == IPv6Context.Interface)
```

```
                              cmd =
configManager.ReadData(ConfigLabels.CLIENT_CMD_DEL_IPV6_ADDRESS,true
);
                        else if (enum_dest == IPv6Context.DNS)
                              cmd =
configManager.ReadData(ConfigLabels.CLIENT_CMD_DEL_DNS_ADDRESS,true)
;

                        if(cmd=="")
                              return "Unknown.";
                        //calculate process and parameters
                        cmd = cmd.Replace("INTERFACE_NAME",
"\""+configManager.ReadData(ConfigLabels.CLIENT_INTERFACE_NAME,true)
+"\"");
                        cmd =
cmd.Replace("IPv6_ADDRESS",ipv6Address);
                        process = cmd.Substring(0,cmd.IndexOf(" "));
                        parameters = cmd.Replace(process+" ","");
                        //run command
                        string temp =
Utility.RunProcess(process,parameters,false);
                        temp = temp.Replace("\r","\t");
                        temp = temp.Replace("\n","\t");
                        return temp;
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Info,e.ToString());
                        //return failed
                        return "Failed.";
                  }
            }

            /// <summary>
            /// Create and send SOLICIT message
            /// </summary>
            private void SendSolicitMessage(){
                  cInfo.dtStart = DateTime.Now;
                  cInfo.waitingFor = DHCPMessages.ADVERTISE;
                  cInfo.reply_for = REPLY_FOR_REQUEST;
                  //create solicit message
                  cInfo.iTransID = Utility.GetNewTransactionID();
                  SolicitMessage slctMsg = new
SolicitMessage(cInfo.iTransID);
                  DUID_LL duid = new
DUID_LL(Utility.GetMACAddress());
                  cInfo.optClntID = new OptionClientID(duid);
                  slctMsg.AddOption(cInfo.optClntID);
                  //check authentication and add authentication
option
                  if(useAuthentication)
                  {
                        cInfo.key =
KeyManager.GetKey(cInfo.optClntID.GetDUID(),true);
                        slctMsg.AddAuthOption(cInfo.key);
                  }
                  //send solicit message

      Send(slctMsg,MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_SERV
ERS);
```

```
                  //-------------------------------------------------
-----------------------------------------------------------------
-----------------
                  //start thread if thread is not started...
                  if(thrdProcessAdvertise==null)
                        thrdProcessAdvertise = new Thread(new
ThreadStart(WaitForMaxServerPref));


      if(thrdProcessAdvertise.ThreadState==ThreadState.Stopped){
                        thrdProcessAdvertise.Abort();
                        thrdProcessAdvertise = new Thread(new
ThreadStart(WaitForMaxServerPref));
                  }


      if(thrdProcessAdvertise.ThreadState!=ThreadState.Running &&
thrdProcessAdvertise.ThreadState !=ThreadState.WaitSleepJoin)
                  {
                        thrdProcessAdvertise.Start();
                  }
                  //wait for thread starts...

      while(thrdProcessAdvertise.ThreadState!=ThreadState.Running &&
thrdProcessAdvertise.ThreadState !=ThreadState.WaitSleepJoin);
                  //-------------------------------------------------
-----------------------------------------------------------------
-----------------

            }

            /// <summary>
            /// Creates and send the decline message
            /// </summary>
            /// <param name="optIA_NA"></param>
            private void SendDeclineMessage(OptionIA_NA optIA_NA){
                  //If a client detects that one or more addresses
assigned to it by a
                  //server are already in use by another node, the
client sends a Decline
                  //message to the server to inform it that the
address is suspect.
                  DeclineMessage dclnMsg = new DeclineMessage
(cInfo.iTransID);
                  dclnMsg.AddOption(cInfo.optClntID);
                  dclnMsg.AddOption(cInfo.optSrvID);
                  dclnMsg.AddOption(optIA_NA);
                  cInfo.waitingFor = DHCPMessages.REPLY;
                  if(useAuthentication)
                        dclnMsg.AddAuthOption(cInfo.key);
                  //send decline message

      Send(dclnMsg,MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_SERV
ERS);
            }

            /// <summary>
            /// waits for max server pref in ADVERTISE message
            /// </summary>
            private void WaitForMaxServerPref(){
                  int i =0;
```

```
                  //wait until time out or max pref receives...
                  while (i< TimeOuts.SOL_MAX_RT &&
waitForMaxServerPref){
                        i += 250;
                        Logger.Log(LogType.Info,"Waiting for maximum
server preference in ADVERTISE message...");
                        Thread.Sleep(250);
                  }

                  //check received advertise messages...
                  if(alAdvertises.Count==0)
                  {
                        //if have not any advertise message
                        Logger.Log(LogType.Warning,"Client has not
received any ADVERTISE message and will send a new SOLICIT
message!...");
                        Thread threadSolicit = new Thread(new
ThreadStart(SendSolicitMessage));
                        threadSolicit.Start();
                  }
                  else{
                        //select the advertise message contains max
pref value, and then process it...
                        DHCPMessage msgMaxPref=null;
                        int iMaxPref = 0;
                        foreach(DHCPMessage msg in alAdvertises){
                              DHCPOption opt =
msg.GetOption(DHCPOptions.OPTION_PREFERENCE);
                              if(opt==null)
                                    continue;
                              OptionPreference optPref = new
OptionPreference (opt);

      if(optPref.GetPreferenceValue()>iMaxPref){
                                    iMaxPref =
optPref.GetPreferenceValue();
                                    msgMaxPref = msg;
                              }
                        }
                        ProcessAdvertiseMessage(msgMaxPref);
                        Logger.Log(LogType.Info,
msgMaxPref.GetName() +" message was processed...");
                  }
            }

            /// <summary>
            /// waits for confirm replays and if not success
receives sends solicit...
            /// </summary>
            private void WaitForConfirmReply()
            {
                  int i =0;
                  //wait until time out
                  while (i< TimeOuts.CNF_MAX_RD && !confirmApproved)
                  {
                        i += TimeOuts.CNF_MAX_RD/5;
                        Logger.Log(LogType.Info,"Waiting for REPLY
message for CONFIRM message...");
                        Thread.Sleep(TimeOuts.CNF_MAX_RD/5);
                  }
```

```
                  //if not Confirm message approved...
                  if(!confirmApproved)
                  {
                        //send solicit
                        SendSolicitMessage();
                  }
                  cInfo.waitingFor = DHCPMessages.RECONFIGURE;
            }


            /// <summary>
            /// Processes the Reconfigure message and sends INF-REQ.
messsage
            /// </summary>
            /// <param name="msg"></param>
            /// <returns></returns>
            private int ProcessReconfigureMessage(DHCPMessage msg)
            {
                  try
                  {
                        DHCPOption optRet;
                        //analyse the advertise
                        OptionServerID optSrvID=null;
                        OptionClientID optClntID=null;
                        //get OPTION_CLIENTID
                        optRet =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(optRet!=null)
                              optClntID = new
OptionClientID(optRet);
                        //get OPTION_SERVERID
                        optRet =
msg.GetOption(DHCPOptions.OPTION_SERVERID);
                        if(optRet!=null)
                              optSrvID = new OptionServerID(optRet);

                        //save selected ServerID
                        cInfo.optSrvID = optSrvID;

                        //check reconfigure message type (RENEW or
INF-REQ)
                        optRet =
msg.GetOption(DHCPOptions.OPTION_RECONF_MSG);
                        OptionReconfigureMessage optReconf = new
OptionReconfigureMessage(optRet);

                        //check reconfigure message type

      if(optReconf.GetMsgTypeValue()==DHCPMessages.RENEW)
                        {

      Logger.Log(LogType.Warning,"Reconfiguration for Renew request
was not implemented. Message will be ignored...");
                              return 0;
                        }
                        else
if(optReconf.GetMsgTypeValue()!=DHCPMessages.INFORMATION_REQUEST){

      Logger.Log(LogType.Warning,"Reconfiguration message has
unknown reconfigure request. Message will be ignored...");
                              return 0;
```

```
                        }

                        //create and send InformationRequest message
                        DHCPMessage reqMsg = null;
                        cInfo.iTransID =
Utility.GetNewTransactionID();
                        reqMsg = new InformationRequestMessage
(cInfo.iTransID);
                        reqMsg.AddOption(optClntID);
                        reqMsg.AddOption(optSrvID);

                        //create authentcation option if necessary
                        if(useAuthentication)
                        {
                              reqMsg.AddAuthOption(cInfo.key);
                        }

                        //send request message

      Send(reqMsg,MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_SERVE
RS);

                        //change client status
                        cInfo.waitingFor = DHCPMessages.REPLY;
                        cInfo.reply_for = REPLY_FOR_REQUEST;
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }

            private void SendConfirmMessage(IPAddress ipv6Addr)
            {
                  cInfo.waitingFor = DHCPMessages.REPLY;
                  cInfo.reply_for = REPLY_FOR_CONFIRM;
                  //create Confirm message
                  ConfirmMessage confirmMsg = new
ConfirmMessage(cInfo.iTransID);
                  //add OptionClientID option
                  DUID_LL duid = new
DUID_LL(Utility.GetMACAddress());
                  cInfo.optClntID = new OptionClientID(duid);
                  confirmMsg.AddOption(cInfo.optClntID);
                  //add OptionIA_NA option
                  OptionIA_NA optIA_NA = new
OptionIA_NA(0,OptionIA_NA.TIME_INFINITE,OptionIA_NA.TIME_INFINITE);
                  OptionIAAddr optIAAddr = new OptionIAAddr
(ipv6Addr,OptionIAAddr.TIME_INFINITE,OptionIAAddr.TIME_INFINITE);
                  optIA_NA.AddOption(optIAAddr);
                  confirmMsg.AddOption(optIA_NA);
                  //check authentication and add authentication
option
                  if(useAuthentication)
                  {
                        cInfo.key =
KeyManager.GetKey(cInfo.optClntID.GetDUID(),true);
                        confirmMsg.AddAuthOption(cInfo.key);
                  }
```

```
                //send solicit message

      Send(confirmMsg,MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_S
ERVERS);
            }



            /// <summary>
            /// Deletes the IPv6 address from client ipv6 address
list
            /// </summary>
            /// <param name="msg"></param>
            /// <returns></returns>
            private int DeleteCurrentIPv6Address(DHCPMessage
msg,bool isRelease){
                  try
                  {
                        DHCPOption optRet=null;
                        OptionIA_NA optIA_NA = null;
                        //get OptionIA_NA
                        optRet =
msg.GetOption(DHCPOptions.OPTION_IA_NA);
                        if(optRet!=null)
                              optIA_NA = new OptionIA_NA(optRet);

                        //get OptionIAAddr option
                        OptionIAAddr optIAAddr = null;
                        if(optIA_NA!=null)
                        {
                              foreach(DHCPOption opt in
optIA_NA.GetSubOptions())
                              {
                                    if(opt.GetOptionCodeValue() ==
DHCPOptions.OPTION_IAADDR)
                                          optIAAddr = new
OptionIAAddr(opt);
                              }
                        }

                        //delete current IPv6 address
                        if(optIAAddr!=null)
                        {
                              string sRet;
                              IPAddress ipAddr =
optIAAddr.GetIPv6Address();
                              sRet=
DeleteIPv6Address(IPv6Context.Interface,ipAddr.ToString());
                              if(isRelease)
                                    Logger.Log(LogType.Info,"Released
IPv6 Address : " + ipAddr.ToString() + "\tRetCode:"+sRet);
                              else
                                    Logger.Log(LogType.Info,"Deleted
IPv6 Address : " + ipAddr.ToString() + "\tRetCode:"+sRet);
                        }
                  }
                  catch(Exception e){
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
```

```
                return 0;
            }


            /// <summary>
            /// Sends release message for the given IPv6 address
            /// </summary>
            /// <param name="ipAddr"></param>
            /// <returns></returns>
            public int SendReleaseMessage(IPAddress ipAddr){
                cInfo.waitingFor = DHCPMessages.REPLY;
                cInfo.reply_for = REPLY_FOR_RELEASE;
                //create release message
                cInfo.iTransID = Utility.GetNewTransactionID();
                ReleaseMessage rlsMsg = new
ReleaseMessage(cInfo.iTransID);
                //add client id
                rlsMsg.AddOption(cInfo.optClntID);
                //add server id
                rlsMsg.AddOption(cInfo.optSrvID);
                //add ia_na
                OptionIA_NA optIA_NA= new
OptionIA_NA(0,OptionIA_NA.TIME_INFINITE,OptionIA_NA.TIME_INFINITE);
                OptionIAAddr optIA = new
OptionIAAddr(ipAddr,OptionIA_NA.TIME_INFINITE,OptionIA_NA.TIME_INFIN
ITE);
                optIA_NA.AddOption(optIA);
                rlsMsg.AddOption(optIA_NA);
                //check authentication and add authentication
option
                if(useAuthentication)
                {
                    cInfo.key =
KeyManager.GetKey(cInfo.optClntID.GetDUID(),true);
                    rlsMsg.AddAuthOption(cInfo.key);
                }
                //send release message

      Send(rlsMsg,MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_SERVE
RS);
                return 0;
            }

            /// <summary>
            /// Send renew message for the given ipv6 address
            /// </summary>
            /// <param name="ipAddr"></param>
            /// <returns></returns>
            public int SendRenewMessage(IPAddress ipAddr)
            {
                cInfo.waitingFor = DHCPMessages.REPLY;
                cInfo.reply_for = REPLY_FOR_RENEW;
                //create release message
                cInfo.iTransID = Utility.GetNewTransactionID();
                RenewMessage rnwMsg = new
RenewMessage(cInfo.iTransID);
                //add client id
                rnwMsg.AddOption(cInfo.optClntID);
                //add server id
                rnwMsg.AddOption(cInfo.optSrvID);
```

```
                  //add ia_na
                  OptionIA_NA optIA_NA= new
OptionIA_NA(0,OptionIA_NA.TIME_INFINITE,OptionIA_NA.TIME_INFINITE);
                  OptionIAAddr optIA = new
OptionIAAddr(ipAddr,OptionIA_NA.TIME_INFINITE,OptionIA_NA.TIME_INFIN
ITE);
                  optIA_NA.AddOption(optIA);
                  rnwMsg.AddOption(optIA_NA);
                  //check authentication and add authentication
option
                  if(useAuthentication)
                  {
                        cInfo.key =
KeyManager.GetKey(cInfo.optClntID.GetDUID(),true);
                        rnwMsg.AddAuthOption(cInfo.key);
                  }
                  //send release message

      Send(rnwMsg,MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_SERVE
RS);
                  return 0;
            }
      }
}
```

Ek 2, DHCPv6 Sunumcusu Kaynak Kodları

```
/*-------------------------------------------------------------+
| Gursoy DURMUS                                                |
| gdurmus@yahoo.com, gdurmus@gmail.com                         |
| Gebze Institude of Technology, Computer Engineering Department |
| http://bilmuh.gyte.edu.tr                                    |
| ============================================================ |
|                                                              |
| Revisions:                                                   |
|                                                              |
+-------------------------------------------------------------*/

using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Collections;

using DHCPv6Lib.Messages;
using DHCPv6Lib.Constants;
using DHCPv6Lib.Utilities;
using DHCPv6Lib.Options;
using DHCPv6Lib.ConfigManager;
using DHCPv6Lib.KeyManagement;

namespace DHCPv6Lib.Logic
{
      /// <summary>
      /// Implementation of the DHCPv6Server logic.
      /// </summary>
      public class DHCPServer:DHCPNode
      {
            //class members
            private IPAddress lastSentIPv6Address;
            private IPAddress maxSentIPv6Address;
            private ArrayList alDNSs = new ArrayList();
            private byte preference ;
            private OptionServerID optSrvID = null;
            private OptionPreference optPref = null;
            private OptionDNSServers optDNS = null;
            private IConfigManager configManager;
            private string sentIPv6Addresses ="";
            private bool addDefaultClientKeys ;
            private bool impersonateMultipleServer;

            /// <summary>
            /// Default constructer
            /// </summary>
            /// <param name="configManager"></param>
            /// <remarks>alDNSs contains IPAddresses</remarks>
            public DHCPServer(IConfigManager configManager)
            {
                  try
                  {
                        Initialize();
```

```
                        Utility.configMan = configManager;
                        KeyManager.cnfMangr = configManager;
                        this.configManager = configManager;

                        //authentication and other config
                        this.useAuthentication =
(configManager.ReadData(ConfigLabels.COMMON_USE_AUTHENTICATION,true)
=="true");
                        this.configureDNS =
configManager.ReadData(ConfigLabels.COMMON_CONFIGURE_DNS,true)=="tru
e";
                        this.configureIPv6Address =
configManager.ReadData(ConfigLabels.COMMON_CONFIGURE_IPV6_ADDRESS,tr
ue)=="true";
                        this.addDefaultClientKeys =
(configManager.ReadData(ConfigLabels.SERVER_ADD_CLIENT_KEY_AUTOMATIC
,true)=="true");
                        this.impersonateMultipleServer =
(configManager.ReadData(ConfigLabels.SERVER_IMPERSONATE_MULTISERVER,
true)=="true");

                        //get LastSentIPv6Address and maxIPv6Address
                        string sTemp;
                        sTemp =
configManager.ReadData(ConfigLabels.SERVER_LAST_SENT_IPV6_ADDRESS,tr
ue);
                        this.lastSentIPv6Address =
IPAddress.Parse(sTemp);
                        sTemp =
configManager.ReadData(ConfigLabels.SERVER_MAX_SENT_IPV6_ADDRESS,tru
e);
                        this.maxSentIPv6Address =
IPAddress.Parse(sTemp);

                        //get DNSServers
                        string[] aTemp;
                        sTemp =
configManager.ReadData(ConfigLabels.SERVER_DNS_SERVERS,true);
                        aTemp = sTemp.Split(';');
                        foreach(string s in aTemp)
                        {
                              if(s!=null && s!="")

      this.alDNSs.Add(IPAddress.Parse(s));
                        }

                        //get server preference
                        sTemp =
configManager.ReadData(ConfigLabels.SERVER_PREFERENCE,true);
                        preference= Convert.ToByte(sTemp);

                        //create default options
                        //serverID option
                        DUID_LL duid = new DUID_LL
(Utility.GetMACAddress());
                        optSrvID = new OptionServerID (duid);

                        //preference option
                        optPref = new OptionPreference(preference);

                        //DNSs option
```

```
                        optDNS = new OptionDNSServers ();
                        foreach(IPAddress ipAddr in alDNSs)
                              optDNS.AddDNSServer(ipAddr);
                  }
                  catch(Exception e){
                        Logger.Log(LogType.Warning,"DHCPv6Server
object could not created properly...");
                        Logger.Log(LogType.Error, e.ToString());
                  }
            }

            /// <summary>
            /// configures DHCPv6Server object's defaults...
            /// </summary>
            private void Initialize(){
                  this.nodeName = "DHCPv6Server 1.0";
                  //log config
                  Logger.appName ="Server";
                  //port config
                  this.m_PortListen = UDPPorts.SERVER_PORT;
                  this.m_PortSend = UDPPorts.CLIENT_PORT;
                  //udpClient config
                  this.m_UdpClientListener = new UdpClient
(m_PortListen,AddressFamily.InterNetworkV6);
                  this.m_UdpClientSender = new UdpClient
(m_PortSend,AddressFamily.InterNetworkV6);

                  //join multicast group
                  IPAddress listenIPAddress =
IPAddress.Parse(MulticastAddresses.ALL_DHCP_RELAY_AGENTS_AND_SERVERS
);
                  IPv6MulticastOption ipv6MulticastOption = new
IPv6MulticastOption(listenIPAddress,0);
                  IPAddress group =  ipv6MulticastOption.Group;
                  long interfaceIndex =
ipv6MulticastOption.InterfaceIndex;

                  //join the specified multicast group for receiving
datagrams

      m_UdpClientListener.JoinMulticastGroup((int)interfaceIndex,
listenIPAddress);

                  //assign port and multicast address to
remoteIPEndPoint object
                  remoteIPEndPoint = new
IPEndPoint(group,m_PortListen);
            }



            /// <summary>
            /// Process message in server's logic
            /// </summary>
            /// <param name="msg"></param>
            /// <param name="source"></param>
            protected override void ProcessMessage(DHCPMessage
msg,string source)
            {
                  //check message is valid
                  if(!IsValidMessage(msg))
```

```
                        return;

                    //show valid message info
                    msg.ShowInfo();

                    //process the valid message
                    ArrayList alOpts = msg.GetOptions();
                    switch(msg.GetMsgType())
                    {
                          //reply SOLICIT message by ADVERTISE
                          case(DHCPMessages.SOLICIT):
                                ProcessSolicitMessage(msg,source);
                                Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                                break;
                          //reply REQUEST message by REPLY
                          case(DHCPMessages.REQUEST):
                                ProcessRequestMessage(msg,source);
                                Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                                break;
                          //reply INFORMATION_REQUEST message by REPLY
                          case(DHCPMessages.INFORMATION_REQUEST):
                                ProcessRequestMessage(msg,source);
                                Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                                break;
                          //reply DECLINE message by REPLY
                          case(DHCPMessages.DECLINE):
                                ProcessDeclineMessage(msg,source);
                                Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                                break;
                          //reply CONFIRM message by REPLY
                          case(DHCPMessages.CONFIRM):
                                ProcessConfirmMessage(msg,source);
                                Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                                break;
                          //reply RELEASE message by REPLY
                          case(DHCPMessages.RELEASE):
                                ProcessReleaseMessage(msg,source);
                                Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                                break;
                          //reply RENEW message by REPLY
                          case(DHCPMessages.RENEW):
                                ProcessRenewMessage(msg,source);
                                Logger.Log(LogType.Info, msg.GetName()
+" message was processed...");
                                break;
                          default:
                                Logger.Log(LogType.Info, msg.GetName()
+" message process logic is not implemented, so message will not be
replied...");
                                break;
                    }

              }

              /// <summary>
```

```
            /// starts server operation
            /// </summary>
            public override void Start()
            {
                  m_Listen = true;
                  m_ThrdListen = new Thread(new
ThreadStart(Listen));
                  m_ThrdListen.Start();
            }


            /// <summary>
            /// stops server operation
            /// </summary>
            public override void Stop()
            {
                  ShowStatus();

                  m_Listen = false;
                  this.m_UdpClientListener.Close();
                  this.m_UdpClientListener=null;
                  this.m_UdpClientSender.Close();
                  this.m_UdpClientSender=null;

                  try
                  {
                        if(m_ThrdListen!=null &&
m_ThrdListen.ThreadState ==ThreadState.Running)
                        {
                              m_ThrdListen.Abort();
                              m_ThrdListen =null;
                        }
                  }
                  catch(Exception e)
                  {
                        //Logger.Log (LogType.Error,e.ToString());
                  }
            }

            /// <summary>
            /// check message is valid for server
            /// </summary>
            /// <param name="msg"></param>
            /// <returns></returns>
            protected override bool IsValidMessage(DHCPMessage msg)
            {
                  //if message is not valid, add it to
IgnoredMessages list
                  if(   !( msg.GetMsgType() != DHCPMessages.SOLICIT
                        || msg.GetMsgType() != DHCPMessages.REQUEST
                        || msg.GetMsgType() != DHCPMessages.CONFIRM
                        || msg.GetMsgType() != DHCPMessages.RENEW
                        || msg.GetMsgType() != DHCPMessages.REBIND
                        || msg.GetMsgType() != DHCPMessages.DECLINE
                        || msg.GetMsgType() != DHCPMessages.RELEASE
                        || msg.GetMsgType() !=
DHCPMessages.INFORMATION_REQUEST
                        || msg.GetMsgType() !=
DHCPMessages.RELAY_FORW
                        ))
                  {
```

```
                        Logger.Log(LogType.Warning, msg.GetName()+ "
message is not acceptable for the server, so message will be
discarded...");
                        alIgnoredMessages.Add(msg);
                        return false;
                  }

                  //check ServerID option if exist
                  DHCPOption opt;
                  opt = msg.GetOption(DHCPOptions.OPTION_SERVERID);
                  if(opt!=null)
                  {
                        OptionServerID optCurrSrvID= new
OptionServerID(opt);
                        if(optCurrSrvID!=null &&
optCurrSrvID.GetDUID()!=optSrvID.GetDUID())
                        {
                              Logger.Log(LogType.Warning,
msg.GetName()+ " message was not sent for this server and will be
ignored...");
                              alIgnoredMessages.Add(msg);
                              return false;
                        }
                  }

                  //check clientID option if exist
                  OptionClientID optClntID;
                  opt = msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                  if(opt==null)
                  {
                        Logger.Log(LogType.Warning, msg.GetName()+ "
message has not clientID option and will be ignored...");
                        alIgnoredMessages.Add(msg);
                        return false;
                  }
                  else
                  {
                        optClntID= new OptionClientID(opt);
                  }

                  //authenticate message
                  if(useAuthentication)
                  {
                        opt =
msg.GetOption(DHCPOptions.OPTION_AUTH);
                        if(opt==null)
                        {
                              Logger.Log(LogType.Warning,
msg.GetName()+ " has not any authentication option. Message was
ignored...");
                              alIgnoredMessages.Add(msg);
                              return false;
                        }
                        OptionAuthentication optAuth = new
OptionAuthentication(opt);
                        //check replay detection...
                        bool bReplyAttack = true;
                        bReplyAttack = bReplyAttack &&
msg.GetMsgType()!=DHCPMessages.SOLICIT;
```

```
                        bReplyAttack = bReplyAttack &&
(OptionAuthentication.GetCurrReplyDetection()-
optAuth.GetReplyDetection() > 60*10000000);
                        if(bReplyAttack)
                        {
                              //replay attack may be in more than 1
minute
                              Logger.Log(LogType.Warning,
msg.GetName()+ " may be replay attcak message... Message was
ignored...");
                              alIgnoredMessages.Add(msg);
                              return false;
                        }
                        //check authentication is valid...
                        if(msg.GetMsgType()!=DHCPMessages.SOLICIT)
                        {
                              string sKey =
KeyManager.GetKey(optClntID.GetDUID(),addDefaultClientKeys);
                              if(sKey==null)
                              {

      Logger.Log(LogType.Warning,optClntID.GetDUID()+" unautorized
client...");
                                    alIgnoredMessages.Add(msg);
                                    return false;
                              }
                              //control authentication informations
                              if(!msg.IsAuthenticated(sKey))
                              {
                                    Logger.Log(LogType.Warning,
msg.GetName()+ " unauthenticated message received... Message was
ignored...");
                                    alIgnoredMessages.Add(msg);
                                    return false;
                              }
                        }
                  }

                  //message is valid
                  return  true;
            }


            /// <summary>
            /// process the incoming solicit message
            /// </summary>
            /// <param name="msg"></param>
            /// <param name="source"></param>
            /// <returns></returns>
            private int ProcessSolicitMessage(DHCPMessage msg,string
source){
                  try{
                        //-----------------------------------------
-----------------------
                        //multiple server support TEST code...
                        //send fake adv message if server
impersonates multiple server
                        //implemented for showing multiple server
support...
                        if(impersonateMultipleServer)
                        {
```

```
                              SendFakeAdvertiseMessage(msg,source);
                              Thread.Sleep(3000);
                        }
                        //-----------------------------------------
-----------------------

                        DHCPOption opt;
                        //create advertise message
                        AdvertiseMessage advMsg = new
AdvertiseMessage(msg.GetTransactinID());
                        //add OptionPreference
                        advMsg.AddOption(optPref);
                        //add OptionServerID
                        advMsg.AddOption(optSrvID);
                        //add ClientID
                        OptionClientID optClntID;
                        opt =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(opt!=null)
                        {
                              //add ClientID
                              optClntID = new OptionClientID(opt);
                              advMsg.AddOption(optClntID);

                              //check authentication
                              if(useAuthentication){
                                    string sKey =
KeyManager.GetKey(optClntID.GetDUID(),addDefaultClientKeys);
                                    if(sKey==null)
                                    {

      Logger.Log(LogType.Warning,optClntID.GetDUID()+" unautorized
client...");
                                          return 0;
                                          //OptionStatusCode optSc =
new OptionStatusCode
(StatusCodes.SC_DEV_UNAUTHORIZED_CLIENT,StatusCodes.DEF_DEV_UNAUTHOR
IZED_CLIENT);
                                          //advMsg.AddOption(optSc);
                                    }
                                    //add authentication option
                                    advMsg.AddAuthOption(sKey);
                              }
                        }
                        //send Advertise message
                        Send(advMsg,source);
                  }
                  catch(Exception e){
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }


            /// <summary>
            /// process the request message
            /// </summary>
            /// <param name="msg"></param>
            /// <param name="source"></param>
            /// <returns></returns>
```

```
            private int ProcessRequestMessage(DHCPMessage msg,string
source)
            {
                  try
                  {
                        DHCPOption opt;
                        //create Reply message
                        ReplyMessage rplyMsg = new
ReplyMessage(msg.GetTransactinID());
                        //add OptionServerID
                        rplyMsg.AddOption(optSrvID);
                        //add OptionDNS
                        if(configureDNS)
                              rplyMsg.AddOption(optDNS);
                        //add ClientID
                        OptionClientID optClntID;
                        opt =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(opt!=null)
                        {
                              //add ClientID
                              optClntID = new OptionClientID(opt);
                              rplyMsg.AddOption(optClntID);
                        }
                        else
                        {
                              Logger.Log(LogType.Warning,
msg.GetName() +" hasn't got any ClientID option...");
                              return 0;
                        }
                        //add OptionIA_NA
                        opt =
msg.GetOption(DHCPOptions.OPTION_IA_NA);
                        if(opt!=null && configureIPv6Address)
                        {
                              //get available IPvAddress
                              lastSentIPv6Address =
Utility.GetNextIPv6Address(lastSentIPv6Address);
                              //control netxIPv6Address less than
maxIPv6Address

      if(Utility.CompareIPAddresses(lastSentIPv6Address,maxSentIPv6A
ddress)>0)
                              {
                                    //no addr avail
                                    //create status code option for
no addr avail
                                    OptionStatusCode optSc = new
OptionStatusCode
(StatusCodes.SC_RFC_NOADDRSAVAIL,StatusCodes.DEF_RFC_NOADDRSAVAIL);
                                    rplyMsg.AddOption(optSc);
                                    Logger.Log
(LogType.Warning,StatusCodes.DEF_RFC_NOADDRSAVAIL);
                              }
                              else{
                                    SaveLastSentIPv6Address();

                                    //save IPv6-MAC address pairs

      configManager.SaveData(ConfigLabels.SERVER_IPv6_MAC_ADDRESS_+l
astSentIPv6Address,optClntID.GetDUID());
```

```
                                    this.sentIPv6Addresses +=
lastSentIPv6Address.ToString() + " / ";
                                    //create OptionIA_NA
                                    OptionIA_NA optIA_NA = new
OptionIA_NA(opt);
                                    OptionIAAddr optIAAddr = new
OptionIAAddr
(lastSentIPv6Address,OptionIAAddr.TIME_INFINITE,OptionIAAddr.TIME_IN
FINITE);
                                    optIA_NA.AddOption(optIAAddr);
                                    rplyMsg.AddOption(optIA_NA);

                                    //save IPv6Address and ClientID
pairs for RECONFIGURE process

      //configManager.SaveData(ConfigLabels.SERVER_IPv6ADDR_CLIENTID
_+lastSentIPv6Address.ToString(),optClntID.GetDUID());
                              }
                        }

                        //check authentication
                        if(useAuthentication)
                        {
                              string sKey =
KeyManager.GetKey(optClntID.GetDUID(),addDefaultClientKeys);
                              if(sKey==null)
                              {

      Logger.Log(LogType.Warning,optClntID.GetDUID()+" unautorized
client...");
                                    return 0;
                              }
                              else
                              {
                                    //add authentication option
                                    rplyMsg.AddAuthOption(sKey);
                              }
                        }

                        //send REPLY message
                        Send(rplyMsg,source);
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }

            /// <summary>
            /// saves the last sent ipv6Address
            /// </summary>
            private void SaveLastSentIPv6Address()
            {
                  configManager.SaveData(
      ConfigLabels.SERVER_LAST_SENT_IPV6_ADDRESS,
lastSentIPv6Address.ToString());
            }
```

```
            /// <summary>
            /// shows the incoming and outgoing messages...
            /// </summary>
            protected override void ShowStatus()
            {
                  //show sent ipv6Addresses
                  Logger.Log(LogType.Info, "Sent IPv6 Addresses:
"+this.sentIPv6Addresses);

                  base.ShowStatus();
            }



            /// <summary>
            /// Send fake advertise message
            /// Implemented for showing the multiple server
support....
            /// </summary>
            /// <param name="msg"></param>
            /// <param name="source"></param>
            /// <returns></returns>
            private int SendFakeAdvertiseMessage(DHCPMessage
msg,string source)
            {
                  try
                  {
                        DHCPOption opt;
                        //create advertise message
                        AdvertiseMessage advMsg = new
AdvertiseMessage(msg.GetTransactinID());
                        //add fake OptionPreference
                        advMsg.AddOption(new OptionPreference(243));

                        //add ClientID
                        OptionClientID optClntID;
                        opt =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(opt!=null)
                        {
                              //add ClientID
                              optClntID = new OptionClientID(opt);
                              advMsg.AddOption(optClntID);

                              //add fake OptionServerID
                              advMsg.AddOption(new
OptionServerID(new DUID_LL("FAKE_SERVER")));

                              //check authentication
                              if(useAuthentication)
                              {
                                    string sKey =
KeyManager.GetKey(optClntID.GetDUID(),addDefaultClientKeys);
                                    if(sKey==null)
                                    {

      Logger.Log(LogType.Warning,optClntID.GetDUID()+" unautorized
client...");
                                          return 0;
                                          //OptionStatusCode optSc =
new OptionStatusCode
```

```
(StatusCodes.SC_DEV_UNAUTHORIZED_CLIENT,StatusCodes.DEF_DEV_UNAUTHOR
IZED_CLIENT);
                                        //advMsg.AddOption(optSc);
                                    }
                                    //add authentication option
                                    advMsg.AddAuthOption(sKey);
                              }
                        }
                        //send Advertise message
                        Send(advMsg,source);
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }


            /// <summary>
            /// Sends Reconfigure Message to related client
            /// </summary>
            /// <param name="toIPv6Addr"></param>
            /// <param name="client_duid"></param>
            /// <returns></returns>
            public int SendReconfigure(IPAddress toIPv6Addr, DUID_LL
client_duid){
                  try
                  {
                        DHCPMessage reconfMsg = new
ReconfigureMessage();
                        //add serverID option
                        reconfMsg.AddOption(optSrvID);
                        //add clientID option
                        OptionClientID optClntID = new
OptionClientID(client_duid);
                        reconfMsg.AddOption(optClntID);
                        //add Reconfigure Message Option
                        OptionReconfigureMessage optReconf = new
OptionReconfigureMessage(DHCPMessages.INFORMATION_REQUEST);
                        reconfMsg.AddOption(optReconf);


                        //check authentication
                        if(useAuthentication)
                        {
                              string sKey =
KeyManager.GetKey(optClntID.GetDUID(),addDefaultClientKeys);
                              if(sKey==null)
                              {

      Logger.Log(LogType.Warning,optClntID.GetDUID()+" unautorized
client...");
                                    return 0;
                                    //OptionStatusCode optSc = new
OptionStatusCode
(StatusCodes.SC_DEV_UNAUTHORIZED_CLIENT,StatusCodes.DEF_DEV_UNAUTHOR
IZED_CLIENT);
                                    //advMsg.AddOption(optSc);
                              }
```

```
                              //add authentication option
                              reconfMsg.AddAuthOption(sKey);
                        }

                        //send Reconfigure Message
                        Send(reconfMsg,toIPv6Addr.ToString());
                  }
                  catch(Exception e){
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }


            /// <summary>
            /// Processes the confirm message
            /// </summary>
            /// <param name="msg"></param>
            /// <param name="source"></param>
            /// <returns></returns>
            private int ProcessConfirmMessage(DHCPMessage msg,string
source)
            {
                  try
                  {
                        DHCPOption opt;
                        //create Reply message
                        ReplyMessage rplyMsg = new
ReplyMessage(msg.GetTransactinID());
                        //add OptionServerID
                        rplyMsg.AddOption(optSrvID);
                        //add ClientID
                        OptionClientID optClntID=null;
                        opt =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(opt!=null)
                        {
                              //add ClientID
                              optClntID = new OptionClientID(opt);
                              rplyMsg.AddOption(optClntID);
                        }

                        //get OptionIA_NA
                        OptionIA_NA optIA_NA = null;
                        opt =
msg.GetOption(DHCPOptions.OPTION_IA_NA);
                        if(opt!=null)
                              optIA_NA = new OptionIA_NA(opt);

                        //get OptionIAAddr option
                        OptionIAAddr optIAAddr = null;
                        if(optIA_NA!=null)
                        {
                              foreach(DHCPOption optX in
optIA_NA.GetSubOptions())
                              {
                                    if(optX.GetOptionCodeValue() ==
DHCPOptions.OPTION_IAADDR)
                                          optIAAddr = new
OptionIAAddr(optX);
```

```
                        }
                    }

                    //check current IPv6 address
                    if(optIAAddr!=null)
                    {
                        string sPrefix =
configManager.ReadData(ConfigLabels.SERVER_IPv6_ADDRESS_PREFIX,true)
;
                        string sIPv6Addr;
                        IPAddress ipAddr =
optIAAddr.GetIPv6Address();
                        sIPv6Addr = ipAddr.ToString();

                        short
statusCode=StatusCodes.SC_RFC_SUCCESS;
                        string
statusMsg=StatusCodes.DEF_RFC_SUCCESS;

                        //check address starts with network
prefix
                        if(!sIPv6Addr.StartsWith(sPrefix)){

      statusCode=StatusCodes.SC_RFC_NOTONLINK;

      statusMsg=StatusCodes.DEF_RFC_NOTONLINK;
                        }

                        OptionStatusCode optStat = new
OptionStatusCode(statusCode,statusMsg);
                        rplyMsg.AddOption(optStat);

                    }
                    else{
                        Logger.Log(LogType.Info,msg.GetName()
+" has not any OptionIAAddr option...");
                        return 0;
                    }

                    //check authentication
                    if(useAuthentication)
                    {
                        string sKey =
KeyManager.GetKey(optClntID.GetDUID(),addDefaultClientKeys);
                        if(sKey==null)
                        {

      Logger.Log(LogType.Warning,optClntID.GetDUID()+" unautorized
client...");
                            return 0;
                            //OptionStatusCode optSc = new
OptionStatusCode
(StatusCodes.SC_DEV_UNAUTHORIZED_CLIENT,StatusCodes.DEF_DEV_UNAUTHOR
IZED_CLIENT);
                            //advMsg.AddOption(optSc);
                        }
                        //add authentication option
                        rplyMsg.AddAuthOption(sKey);
                    }

                    //send reply message
```

```
                        Send(rplyMsg,source);
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }


            /// <summary>
            /// Processes the RELEASE message
            /// saves the released ipv6 address in released pool for
send another clients
            /// </summary>
            /// <param name="msg"></param>
            /// <param name="source"></param>
            /// <returns></returns>
            private int ProcessReleaseMessage(DHCPMessage msg,string
source)
            {
                  try
                  {
                        DHCPOption opt;
                        //create Reply message
                        ReplyMessage rplyMsg = new
ReplyMessage(msg.GetTransactinID());
                        //add OptionServerID
                        rplyMsg.AddOption(optSrvID);
                        //add ClientID
                        OptionClientID optClntID=null;
                        opt =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(opt!=null)
                        {
                              //add ClientID
                              optClntID = new OptionClientID(opt);
                              rplyMsg.AddOption(optClntID);
                        }

                        //get OptionIA_NA
                        OptionIA_NA optIA_NA = null;
                        opt =
msg.GetOption(DHCPOptions.OPTION_IA_NA);
                        if(opt!=null)
                              optIA_NA = new OptionIA_NA(opt);

                        //get OptionIAAddr option
                        OptionIAAddr optIAAddr = null;
                        if(optIA_NA!=null)
                        {
                              foreach(DHCPOption optX in
optIA_NA.GetSubOptions())
                              {
                                    if(optX.GetOptionCodeValue() ==
DHCPOptions.OPTION_IAADDR)
                                          optIAAddr = new
OptionIAAddr(optX);
                              }
                        }
```

```
                        //check current IPv6 address
                        if(optIAAddr!=null)
                        {
                              string sPrefix =
configManager.ReadData(ConfigLabels.SERVER_IPv6_ADDRESS_PREFIX,true)
;
                              string sIPv6Addr;
                              IPAddress ipAddr =
optIAAddr.GetIPv6Address();
                              sIPv6Addr = ipAddr.ToString();

                              short
statusCode=StatusCodes.SC_RFC_SUCCESS;
                              string
statusMsg=StatusCodes.DEF_RFC_SUCCESS;

                              //check address starts with network
prefix
                              if(!sIPv6Addr.StartsWith(sPrefix))
                              {

      statusCode=StatusCodes.SC_RFC_NOTONLINK;

      statusMsg=StatusCodes.DEF_RFC_NOTONLINK;
                              }
                              else{
                                    //save ipv6 address in released
pool
                                    string sReleasePool =
configManager.ReadData(ConfigLabels.SERVER_RELEASED_IPv6_ADDRESSES,t
rue);
                                    sReleasePool+=";"+sIPv6Addr+";";
                                    sReleasePool =
sReleasePool.Replace(";;",";");

      configManager.SaveData(ConfigLabels.SERVER_RELEASED_IPv6_ADDRE
SSES,sReleasePool);
                                    //delete IPv6-MAC address pairs
                                    ArrayList alMACs =
configManager.ReadAllData(ConfigLabels.SERVER_IPv6_MAC_ADDRESS_+sIPv
6Addr);
                                    foreach(ConfigItem ci in alMACs)

      configManager.DeleteData(ConfigLabels.SERVER_IPv6_MAC_ADDRESS_
+ci.configLabel,optClntID.GetDUID());

                                    Logger.Log(LogType.Info,sIPv6Addr
+ " address released...");

                                    //add IA_NA option
                                    rplyMsg.AddOption(optIA_NA);
                              }

                              OptionStatusCode optStat = new
OptionStatusCode(statusCode,statusMsg);
                              rplyMsg.AddOption(optStat);

                        }
                        else
                        {
```

```
                              Logger.Log(LogType.Info,msg.GetName()
+" has not any OptionIAAddr option...");
                              return 0;
                        }

                        //check authentication
                        if(useAuthentication)
                        {
                              string sKey =
KeyManager.GetKey(optClntID.GetDUID(),addDefaultClientKeys);
                              if(sKey==null)
                              {

      Logger.Log(LogType.Warning,optClntID.GetDUID()+" unautorized
client...");
                                    return 0;
                                    //OptionStatusCode optSc = new
OptionStatusCode
(StatusCodes.SC_DEV_UNAUTHORIZED_CLIENT,StatusCodes.DEF_DEV_UNAUTHOR
IZED_CLIENT);
                                    //advMsg.AddOption(optSc);
                              }
                              //add authentication option
                              rplyMsg.AddAuthOption(sKey);
                        }

                        //send reply message
                        Send(rplyMsg,source);
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }


            /// <summary>
            /// Processes the RENEW message
            /// </summary>
            /// <param name="msg"></param>
            /// <param name="source"></param>
            /// <returns></returns>
            private int ProcessRenewMessage(DHCPMessage msg,string
source)
            {
                  try
                  {
                        DHCPOption opt;
                        //create Reply message
                        ReplyMessage rplyMsg = new
ReplyMessage(msg.GetTransactinID());
                        //add OptionServerID
                        rplyMsg.AddOption(optSrvID);
                        //add ClientID
                        OptionClientID optClntID=null;
                        opt =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(opt!=null)
                        {
```

```
                              //add ClientID
                              optClntID = new OptionClientID(opt);
                              rplyMsg.AddOption(optClntID);
                        }

                        //get OptionIA_NA
                        OptionIA_NA optIA_NA = null;
                        opt =
msg.GetOption(DHCPOptions.OPTION_IA_NA);
                        if(opt!=null)
                              optIA_NA = new OptionIA_NA(opt);

                        //get OptionIAAddr option
                        OptionIAAddr optIAAddr = null;
                        if(optIA_NA!=null)
                        {
                              foreach(DHCPOption optX in
optIA_NA.GetSubOptions())
                              {
                                    if(optX.GetOptionCodeValue() ==
DHCPOptions.OPTION_IAADDR)
                                          optIAAddr = new
OptionIAAddr(optX);
                              }
                        }

                        //check current IPv6 address
                        if(optIAAddr!=null)
                        {
                              string sPrefix =
configManager.ReadData(ConfigLabels.SERVER_IPv6_ADDRESS_PREFIX,true)
;
                              string sIPv6Addr;
                              IPAddress ipAddr =
optIAAddr.GetIPv6Address();
                              sIPv6Addr = ipAddr.ToString();

                              short
statusCode=StatusCodes.SC_RFC_SUCCESS;
                              string
statusMsg=StatusCodes.DEF_RFC_SUCCESS;

                              //check address starts with network
prefix
                              if(!sIPv6Addr.StartsWith(sPrefix))
                              {

      statusCode=StatusCodes.SC_RFC_NOTONLINK;

      statusMsg=StatusCodes.DEF_RFC_NOTONLINK;
                              }
                              else
                              {
                                    //control ipv6-mac address pairs
                                    string sMACofIP =
configManager.ReadData(ConfigLabels.SERVER_IPv6_MAC_ADDRESS_+sIPv6Ad
dr,false);
                                    if(sMACofIP==null ||
sMACofIP.Trim()!=optClntID.GetDUID()){

      statusCode=StatusCodes.SC_RFC_NOBINDING;
```

```
      statusMsg=StatusCodes.DEF_RFC_NOBINDING;
                                    }
                                    //add IA_NA option
                                    rplyMsg.AddOption(optIA_NA);
                              }
                              OptionStatusCode optStat = new
OptionStatusCode(statusCode,statusMsg);
                              rplyMsg.AddOption(optStat);
                        }
                        else
                        {
                              Logger.Log(LogType.Info,msg.GetName()
+" has not any OptionIAAddr option...");
                              return 0;
                        }

                        //check authentication
                        if(useAuthentication)
                        {
                              string sKey =
KeyManager.GetKey(optClntID.GetDUID(),addDefaultClientKeys);
                              if(sKey==null)
                              {

      Logger.Log(LogType.Warning,optClntID.GetDUID()+" unautorized
client...");
                                    return 0;
                                    //OptionStatusCode optSc = new
OptionStatusCode
(StatusCodes.SC_DEV_UNAUTHORIZED_CLIENT,StatusCodes.DEF_DEV_UNAUTHOR
IZED_CLIENT);
                                    //advMsg.AddOption(optSc);
                              }
                              //add authentication option
                              rplyMsg.AddAuthOption(sKey);
                        }

                        //send reply message
                        Send(rplyMsg,source);
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }

            /// <summary>
            /// Processes the decline message, and sends REPLY
message
            /// </summary>
            /// <param name="msg"></param>
            /// <param name="source"></param>
            /// <returns></returns>
            private int ProcessDeclineMessage(DHCPMessage msg,string
source)
            {
                  try
                  {
```

```
                        //get client id
                        DHCPOption optRet =
msg.GetOption(DHCPOptions.OPTION_CLIENTID);
                        if(optRet==null)
                              return 0;
                        OptionClientID optClnt = new
OptionClientID(optRet);

                        //get ignored ipv6 address
                        OptionIA_NA optIA_NA = null;
                        OptionIAAddr optIAAddr = null;
                        //get OptionIA_NA
                        optRet =
msg.GetOption(DHCPOptions.OPTION_IA_NA);
                        if(optRet==null)
                              return 0;
                        optIA_NA = new OptionIA_NA(optRet);

                        //get OptionIAAddr option
                        if(optIA_NA!=null)
                        {
                              foreach(DHCPOption opt in
optIA_NA.GetSubOptions())
                              {
                                    if(opt.GetOptionCodeValue() ==
DHCPOptions.OPTION_IAADDR)
                                          optIAAddr = new
OptionIAAddr(opt);
                              }
                        }

                        //get ipv6 address
                        IPAddress ipAddr=null;
                        if(optIAAddr!=null)
                        {
                              ipAddr = optIAAddr.GetIPv6Address();
                              if(ipAddr==null)
                                    return 0;
                        }

                        //delete ipv6-mac pair

      configManager.DeleteData(ConfigLabels.SERVER_IPv6_MAC_ADDRESS_
+ipAddr.ToString(),optClnt.GetDUID());

                        //send new ipv6 address
                        ProcessRequestMessage(msg,source);
                  }
                  catch(Exception e)
                  {
                        Logger.Log(LogType.Error,e.ToString());
                        return -1;
                  }
                  return 0;
            }

      }
}
```