VoIP Santral Çekirdek Bileşeninde Yazılım Yaması Modeli

Necip Gözüaçık[1], Fatih Ayvaz[1], Bahadır Özdemir[1], A. Belma Şahin-Kaya[1]

Oğuzhan Yavuz[1]

[1] Netaş Telekomünikasyon A.Ş, İstanbul, Türkiye
{gozuacik,fayvaz,bozdemir,belmas,oyavuz}@netas.com.tr

Özet. Bu çalışmada haberleşme santrallerinde (VoIP, PSTN) koşan yazılım mimarisinde yama kullanımının önemi ve uygulanabilirliğine ilişkin Netaş'ın geliştirdiği yama süreci paylaşılmıştır. Bu süreç Netaş'ın tüm ArGe faaliyetlerinden sorumlu olduğu yüksek kapasiteli ve çok bileşenli VoIP santrali üzerinde etkin olarak kullanılmaktadır. Bu model, santralin kalbi olan Çekirdek üzerindeki uygulaması temel alınarak anlatılmıştır.

Anahtar Kelimeler: Yama, PROTEL, VoIP Sistemleri

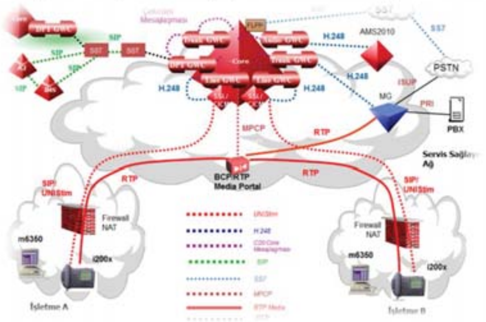
1 Giriş

Bir telekomünikasyon ağında haberleşmenin sağlanmasını, donanımsal alt yapı ile onun üzerine kurulan yazılım sağlamaktadır. Bir İnternet Protokolü Üzerinden Ses İletişimi (VoIP: Voice over Internet Protocol,) santrali genel anlamda birden çok bileşenden oluşmaktadır. Bu bileşenler temel olarak Çekirdek (Core), Ağ Geçidi (Gateway, GW), Ağ Geçidi Denetleyicisi (Gateway Controller, GWC), Oturum Trank Sunucusu (Session Server Trunk, SST), İşletim Yönetim Birimi (OAM&P) gibi modüllerden ibarettir. Her bir bileşenin kendi yapısı, işlemci özelliği gereği farklı bir yazılım dili ve ortamı olabilmektedir. Telekom santrali çözümleri sağlayan firmalar, genellikle santral yazılımlarını ya da diğer bir ifadeyle santralin işletim sistemlerini sürümler altında belirli periyodlar ile geliştirmektedirler. Bunun yanında telekom operatörleri sahada var olan santral yazılımlarında, abonelerden gelen problemlerle karşılaşabilmektedirler. Sistem çok büyük ve karmaşık olduğu için sahada karşılaşılan bir problemin çözümünü yeni bir santral yazılım sürümü ile en baştan vermek pek uygun ve mümkün değildir. Canlı konuşma trafiği altındaki bu santral yapılarına çok hızlı bir şekilde yazılım çözümü sunulmasına ihtiyaç vardır. Hali hazırda mevcut kurulu olan santrallerdeki tüm çağrı trafikleri, mesajlaşmalar vb. anlık olarak devam etmektedir. Gerçek zamanlı bu haberleşme ortamında sadece karşılaşılan problemin çözülmesini sağlayacak minik yazılım parçacıklarına ihtiyaç duyulur. Bu yazılım parçacıkları sayesinde sistemde herhangi bir köklü değişiklik yapılmadan mevcut ortama yazılım yaması uygulanarak müşterinin problemi giderilmiş olunacaktır. Yazılım yamasının [1] tercih edilme nedenleri; hızlı çözüm sunmak, tüm müşterilere aynı çözümü ulaştırabilmek, müşteriler için bakım ve işletim maliyeti kazancı sağlamaktır. Bu yayın içerisinde telekom hizmeti veren santral yapılarındaki en önemli birim olan Çekirdek üzerinde yazılım yamasının uygulamasına ilişkin yama süreci verilmiştir.

Bu yayın altı bölümden oluşmaktadır. Bölüm 2'de Netaş'ın müşterilerine sunduğu VoIP tümleşik çözüm mimarisi ve bu mimarideki bileşenler anlatılmıştır. Bölüm 3'te Çekirdek bileşenindeki yazılım mimarisi anlatılmıştır. Bölüm 4'te Çekirdek bileşeninde kullanılan yazılım yaması modeline değinilmiştir. Bir sonraki bölümde yazılım yaması uygulanırken izlenen adımlara ve iş akışına yer verilmiştir. Son bölümde ise genel değerlendirmelere ve sonuçlara değinilmiştir.

2 VoIP Tümleşik Çözüm Mimarisi

Şekil 1'de verilen çözüm, yeni teknolojilerle eski teknolojileri kullanan Zaman Bölmeli Çoğullama (TDM: Time Division Multiplexing,) abonelerine hizmet verebilmek için kullanılan IP tabanlı haberleşme altyapısıdır. Netaş'ta yapılan geliştirmeler sonucunda TDM donanımlar santral üzerinden kaldırılmıştır. Bu yenilik sonucunda santral tamamen IP olmuştur. Bu çözüm ile santrale konumdan bağımsızlık özelliği kazandırılmıştır. Başka bir deyişle bu santrali A noktasına yerleştirip birçok bölgeye ağ geçitleri üzerinden hizmet verilebilmektedir. Bu özellik sayesinde operatörler, dağınık santrallerini merkezi yüksek kapasiteli santrallerle değiştirme imkânı elde etmişlerdir. Bu çözümün en büyük getirisi işletim ve yapılandırma maliyetlerinin düşmesidir.



Şekil 1. Netaş Tümleşik Çözüm Mimarisi ve Protokoller

Şekil 1'de verilen yeni nesil VoIP santralin yazılımı 41 milyon satır koddan oluşmaktadır. Bu kadar büyük bir yazılımın sebebi bu santralin, değişik görevleri yerine getiren çok fazla alt bileşenin santral üzerinde oluşması ve 1300 civarında servisi müşterilerine sunabilmesidir. Şekil 1'de görüldüğü gibi santral tarafından birçok protokol desteklenmekte ve kullanılmaktadır. Ayrıca bu santral tarafından dünyadaki neredeyse tüm haberleşme standartları desteklenmektedir.

Şekil 1'deki yapının en önemli bileşeni Çekirdek (Core)'dur. Çekirdek aramanın kurulması, yönlendirilmesi ve sonlandırılması esnasındaki sinyalleşme gibi arama ve servislerle ilgili tüm kontrolleri üstlenir. Ayrıca hizmet verdiği abonelerin tüm kayıtlarını, faturalandırma detayları gibi bilgileri oluşturur. Bunlara ek olarak üzerinde 500 e yakın çeşitli servisler barındırır (çağrı engelleme, numara gizleme, çağrı yönlendirme, çağrı bekletme, konferans başlatma vb.)

GW transfer katmanında sinyalleşme sistem no:7 (SS7) ile IP tabanlı sinyalleşme protokolleri arasında dönüşüm yapmaktadır. Buradaki sistemde GW, TDM abonelerinin VoIP teknolojisine entegrasyonunda önemli yer tutmaktadır.

GWC, Çekirdek ile GW arasındaki bağlantıyı sağlamaktadır. Bir GWC 64 adet GW'yi destekleyebilmektedir. Çekirdek ile bağlantı kopması durumunda kendisine bağlı GW'ler arasında aramanın kurumunu yapabilmektedir. Ayrıca hatların durum güncellemeleri GWC tarafından Çekirdek' e sağlanmaktadır.

Oturum Trank Sunucusu (SST) santrali IP altyapıya bağlayan, diğer santraller ile bağlantıyı sağlayan birimdir. SST, Çekirdek'e GWC üzerinden bağlanırken sisteme özel bir protokol kullanılır. SST, IMS (IP multi-medya systems, IP çoklu-medya sistemleri) ve diğer santraller arasında SIP (Session Initiation Protocol, Oturum Başlatma Protokolü) kullanılmaktadır.

İşletim yönetim birimi çok bileşenli santralin operatörlere uzaktan kontrol ve takip imkânı sunan birimdir. Bu birim 4 farklı alt birimden oluşmaktadır:

Santral Yönetimi Araçları (CMTg/CMT): Abone tanımlama, ağ geçidi yöneticisi ve ağ geçidi tanımlama, trank uç noktaları tanımlama gibi işlevlere yerine getirmektedir.

Genview Yöneticisi Tabanı/Entegre Olmuş Eleman Yönetim Sistemi: Sistem elemanları üzerindeki hata ve olay günlüklerinin etkin takibi ve bu elemanların yönetim araçlarına ulaşılmasını sağlar.

aTCA Çağrı Sunucu Yöneticisi: Çekirdek yapısı üzerindeki hata ve olay günlüklerinin etkin bir şekilde takibini sağlayan bir uygulamadır.

Merkezi Fatura Yöneticisi (CBMg): Santral öğelerinin işletimi, yönetimi ve bakımının gerçekleştirilmesine olanak sağlayan verileri belli formatta saklayan yazılım uygulamaları bütünüdür

3 Çekirdek Yazılım Mimarisi

Netaş'ın müşterilerine sunduğu VoIP çözümündeki Çekirdek bileşeninde Destek İşletim Sistemi – Support Operating System (SOS) diye isimlendirilen bir işletim sistemi yer almaktadır. Çekirdek üzerinde koşan bu işletim sistemi 1975 yılında Bell-Northern Araştırma laboratuvarında geliştirilmiş olan Procedure Oriented Type Enforcing Language (PROTEL) [2] dili kullanılarak tasarlanmıştır. Bu dilin en büyük özelliği telekomünikasyon sisteminin ihtiyaçları göz önünde bulundurularak tasarlanmış olmasıdır. Çağrı kurulumu, sinyalleşmeler ve protokollere ilişkin özellikler bu dil içerisinde oldukça esnek ve anlaşılır bir şekilde modellenmiştir. PROTEL, PASCAL [3] ve ALGOL 68 [4] tabanlı bir dil olup C diline de oldukça benzemektedir. Bu dilin sonradan geliştirilen nesne yönelimli sürümü de PROTEL-2 olarak isimlendirilmiştir [5].

Çekirdek'in diğer bileşenlerle ve sistemin bütünü ile iletişimin sağlanması konusunda işlemci kullanımının çağrı trafiğine bağlı olarak uygun şekilde düzenlenmesi, giriş verilerine ilişkin isteklerinin kesintisiz bir biçimde karşılanması, sistem kaynaklarına aynı anda yapılmak istenen erişimlerin kontrollü bir şekilde karşılanması ve donanımsal ya da yazılımsal hatalar karşısında tüm sistemin bloke olmamasına ilişkin koruma mekanizmalarının hayata geçirilmesi görevlerini yerine getirmesi beklenir.

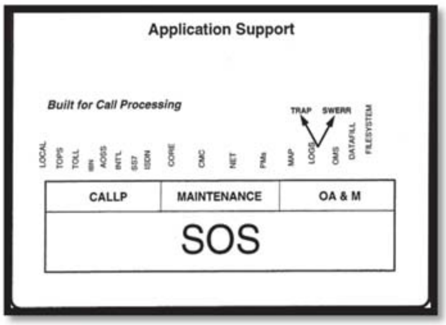
Yukarıda anlatılan görevler için SOS işletim sisteminden beklenen yetkinlikleri;bellek yönetimi (Data Store ve Program Store alanlarının kontrolü), yazılım yamasının ya da yeni bir yazılımın sisteme yüklenilmesi, kaldırılması, çevre bileşenlerden (GWC, Sinyalleşme ara yüzü, Mesajlaşma ara yüzü) ile gelen mesajların işlenmesi ve bu bileşenlere gidecek mesajların oluşturulması, zamanlayıcıların senkron bir şekilde çalışmasının sağlanması, iş parçacıklarının (Thread) kontrol altında tutulması, dosya sistemlerine erişimin sağlanması ve komut seviyesinde kullanıcı ara yüzü hizmeti verilmesi şeklinde sıralanabilir.

Şekil 2'de görüldüğü üzere Çekirdek'te koşan SOS yaklaşık 60.000 ayrı modülden/dosyadan oluşmaktadır. Bu da yaklaşık ~12 milyon satır kodu meydana getirmektedir. Görüldüğü gibi oldukça büyük bir yazılımsal veri içeren bu yapının sıfır toleransla hizmet vermesi oldukça önemlidir.

Şekil 2. SOS sisteminde modül yapısı

SOS işletim sistemi Şekil 3'te gösterilen 3 temel fonksiyonu; çağrı işleme, bakım/onarım, sürdürülebilirlik ve makine insan ara yüzü için gerekli altyapıyı sağlar.

Şekil 3'te SOS yapısının üzerindeki Çağrı işleme (CALLP), Bakım ve Onarım (MAINTENANCE) ve İşletim, Ölçme ve Ücretlendirme (OAM) bölümleri gösterilmiştir. SOS yazılımı en genel anlamda iş parçacıklarından oluşan prosedürel bir yapıya sahiptir. İşlemci, proseslerden gelen istekleri sırayla işleyip ilgili prosedürel yapıların koşmasını sağlayarak sistemin sonsuz bir döngü içerisinde çalışmasını sağlar.



Şekil 3. SOS yapısının genel mimarideki yeri

4 Çekirdek YazılımYaması Modeli

Telekom operatörlerine sunulan santral çözümlerinde, tasarım ekiplerinin belirlenmiş bir dönem süresince üzerinde geliştirme yaptıkları Çekirdek yazılım yüküne ait SOS'un son hali verilir. Üzerinde çalışılan bu yükte tasarım, fonksiyonel test ve sistem test mühendisleri gerekli çalışmaları ve testleri yaparlar. Yazılımın temel doğası gereği sahada gerçek konuşma trafiği altında müşteriler çok çeşitli problemler raporlayabilmektedirler. Raporlanan problemlerin müşterilere en kısa sürede ve en az yan etki ile verilmesi için yazılım yaması modeli kullanımı uygun görülmüştür. Bu konuda endüstride ve literatürde kullanılan diğer bir yöntem ise sistemin tamamen yeni baştan yüklenmesine dayanan modeldir. Bu yöntemde sistemin tamamına müdahale söz konusu olacağından müşteriler belirli bir süreliğine servis kesintisi yaşarlar. Ayrıca direkt olarak program belleğine müdahele ederek yazılım güncelleştirme yöntemleri de vardır [6]. Oysa yama modelinde müşteri bir yandan yazılım güncellemesini alırken bir yandan da kesintisiz bir şekilde servis hizmeti almaya devam etmektedir. SOS ve kullanılan PROTEL sayesinde bu tür haberleşme sistemlerinde merkezi bileşendeki yazılım güncelleştirmelerinin oldukça esnek ve yan etkisiz olması sağlanmıştır.

Yama, birçok yazılım uygulamasında oldukça yoğun bir şekilde kullanılan bir yöntemdir. Netaş'ın sunduğu santral çözümünde, problemlerin yazılım yaması ile giderilmesi oldukça eskilere dayanır. 1990'ların başından itibaren yazılım yama modeli başarılı bir şekilde uygulanarak günümüze kadar gelmiştir. Bir santral içerisindeki birçok bileşende aslında yama modeli uygulanmaktadır. Bu çalışmada üzerinde durulacak alan ise Çekirdek bileşenindeki yama modeli olacaktır.

Çekirdek bileşeni üzerinde yama geliştirmekle aslında kastedilen, yapılan bir yazılımsal geliştirmenin mevcut çalışan sisteme gerçek zamanlı olarak uygulanmasıdır. Yazılım yaması geliştirilmesini gerektiren nedenleri

1. Sahadan gelen müşteri problemlerinin çözülmesi
2. Ürün tasarımı sırasında bulunan problemlerin çözülmesi
3. Müşteriye özel proje geliştirmeleri

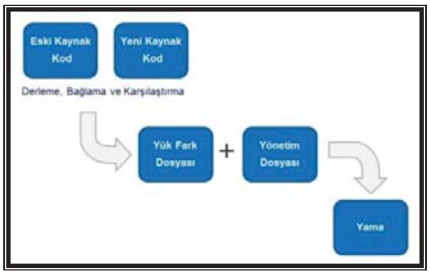
olmak üzere 3 ana başlık altında toplayabiliriz. İlk iki maddede verilen nedenlerden önceki bölümlerde bahsedilmişti. 3. maddede verilen neden müşterilerin taleplerinin yazılım yaması ile karşılanmasını ifade etmektedir. Bu istekler aboneler için yeni servisler tanıtılması, çağrı yönlendirmede numara engellemesi vb. gibi servisler veya uygulamalar olabilir. Bu tür müşteri istekleri, eğer tasarım aşaması kısa ve mimari açıdan köklü bir değişikliğe ihtiyaç yoksa, yama çözümü şeklinde müşterinin kullandığı işletim sistemi sürümüne uygulanmaktadır.

Çekirdek bileşeni üzerinde yazılım yaması yöntemi olarak "Hitless Patching" kullanılır [7]. "Hitless Patching", çalışır durumdaki bir sistemin üzerine bir yamanın sorunsuz bir şekilde uygulanması diye düşünülebilir. Bu yapıyı isterseniz "Car Analogy (Araba Analojisi)" [7] konsepti üzerinden anlatmaya çalışalım. Yamanın uygulanacağı hedefi bir otomobil olarak varsayalım. Yamanın uygulanması motoru çalışan bir otomobil üzerinde işlem yapmak gibi olacaktır. Bu açıdan yamanın uygulanması sırasında aşağıdaki noktalara dikkat etmek gerekir.

- Lastiklerin şişirilmesi, sileceklerin değiştirilmesi vb. işlemler (prosedürel değişiklikler, lokal eklemeler vb.) **kabul edilebilir**
- Benzin ilave etmek (bellekte yeni bir alan yaratma vb.) **tavsiye edilmez**
- Buji, vantilatör kayışı ya da yağ değişimi (çağrı kurulmasını engelleme, sistemi kesintiye uğratma vb.) **kesinlikle yapılmamalıdır**

Bu yama türünde motorun asla durdurulmaması (santralin yeniden başlatılması, yeniden yüklenmesi vb. gibi) gerekir. Ayrıca otomobildeki orijinal parçaların iyi saklanması gerekir çünkü yama geri çekilmek zorunda kalındığında onlara tekrar ihtiyaç olacaktır.

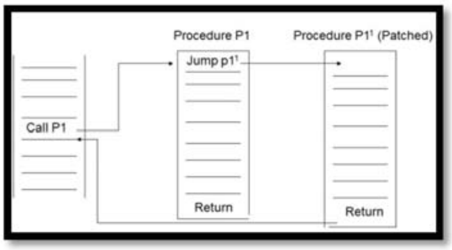
Şekil 4'te Çekirdek bileşeninde yazılım yamasının oluşturulmasına ilişkin bir şema verilmiştir. Yama oluşturulmasında temel mantık bir dosyadaki mevcut kodun içeriği ile geliştirme yapılan yeni hali arasındaki farkın belirlenerek prosedürel yapılara ilişkin adres değişimlerinin ayarlanmasıdır.



Şekil 4. Yama Oluşturulması

Eski ve yeni kaynak kodları, sırasıyla kodun orijinal halinin ve yeni halinin tutulduğu dosyaları göstermektedir. Yük Fark Dosyası ise eski ve yeni kaynak kod dosyaları arasındaki değişiklikleri tutan yapıdır. Ayrıca yamanın özellikleri ve kullanım bilgileri yönetim dosyasında saklanır.

Şekil 5'te Çekirdek bileşeni için hazırlanan yamanın santrale uygulanması sırasında işletim sisteminin nasıl davrandığı gösterilmiştir.



Şekil 5. Yamanın Çalışması

Şekil 5, yamanın çalışma prensibini göstermektedir. Uygulanan yamanın P1 metodundaki bir kodu değiştirdiğini varsayalım. Bu durumda P1 metodu çağırıldığı sırada işletim sistemi onu yeni bir adres alanına göndererek kodun yeni halinin çalışmasını sağlar.

Şekil 6'da Netaş'ın sunduğu VoIP çözümünde yama kullanımının uçtan uca dağıtımı özetlenmeye çalışılmıştır. Bu resim üç aşamada ele alınabilir. İlk aşamada yama yazılımı, gerekli yazılım araçları, kütüphaneler ve yazılım veri tabanları kullanılarak oluşturulur. 2. aşama bu yama yazılımının müşteriler tarafından ulaşılabilecek ortak bir veri tabanında tutulmasıdır. 3. ve son aşamada ise müşterinin santralindeki tüm bileşenler için hazırlanmış olan yamaların santrale gerekli araçlar sayesinde uygulanmasıdır.

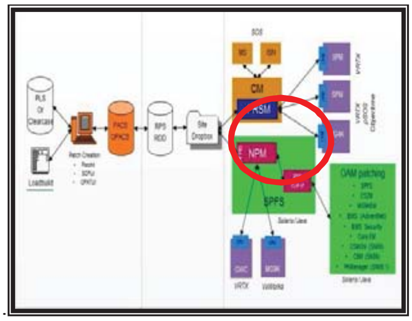
5 Çekirdek Yazılım Yama Süreci

Müşterilere güvenli ve kaliteli bir yama yazılımı verilmesi için bir proses uygulanması elzemdir [8]. Bu proses ile bir yama yazılımı geliştirirken geçilen adımlardan bahsedilecektir.

5.1 Kodlama

Yazılım yamasına ilişkin kod geliştirilmesinde, üzerinde geliştirme yapılan işletim sisteminin son sürümü için hazırlanan kod baz alınır. Yani öncelikle problemin çözümüne ilişkin kod henüz açık durumda olan işletim sistemi sürümü üzerinde geliştirilir. Burada temel amaç, henüz açık durumda bulunan işletim sistemi sürümündeki kodun, yamanın kullanılacağı işletim sistemi sürümüne göre yapılandırılmasıdır.

Örneğin sahadan problem raporlayan müşterinin kullandığı işletim sisteminin sürümü 14 olsun. Henüz geliştirilme aşamasında olan işletim sisteminin sürümü de 18 olsun. Yapılan iş, problemi 18 numaralı sürümde çözüp bu çözümü 14 numaralı sürüme yama olarak hazırlamaktır. Yazılım yaması hazırlanırken yapılan işlem şu şekilde gerçekleşir. İlk olarak son sürümdeki modülde/dosyada yapılan kod değişikliğine ilişkin bir fark dosyası oluşturulur. Oluşturulan bu fark dosyasından yararlanılarak aynı kod değişikliği yama yazılacak sürümdeki ilgili modülün/dosyanın son haline eklenir. Böylelikle hem açık olan işletim sistemi sürümünde hem de müşterinin kullandığı işletim sistemi sürümünde problem çözülmüş olacaktır. Çoğullama ile de arada kalan diğer işletim sistemi sürümlerine de aynı çözüm sunulmuş olacaktır.



Şekil 6. Uçtan Uca Yama Dağıtımı

5.2 Kod Kontrolü

Hazırlanan yama kodu ve yamanın kullanımına ilişkin rehber dosyası tecrübeli mühendisler tarafından gözden geçirilir. Kontrol sırasında bulunan hatalar yazılım yamasını hazırlayan tasarımcı tarafından giderilerek kodun en son hali için onay alınır.

5.3 Test

Hazırlanan yazılım yaması müşterideki ortama uygun şekilde konfigüre edilen laboratuvarda test edilir. Test ederken geliştirilen tüm kod parçalarının çalıştığından emin olmak gerekir. Bundan dolayı orijinal problemin testi dışında ek testlerin de koşulması gerekmektedir. Test türleri aşağıdaki gibidir.

Fonksiyonel Test: Raporlanan orijinal problemin çözülüp çözülmediğine ilişkin yapılan testtir. Bu test için müşterinin sahadaki konfigürasyonu ile birebir aynı ortamın laboratuvar koşullarında sağlanması gerekir.

Birim Test: Yazılım yamasındaki eklenen/değiştirilen kodların her bir parçasının çağırılmasını/kullanılmasını sağlayacak test senaryoları oluşturulur. Oluşturulan bu senaryolar denenirken çeşitli araçlar yardımı ile kodun olduğu yerlere izleme noktaları koyulur. Böylece tüm kodların test edilmesi garanti altına alınmış olur.

Stres Testi: Yazılım yamasındaki kod içerisinde bellek alanı kullanımına ilişkin sınır noktaların test edilmesi, işaretli/işaretsiz sayı kullanımlarına ilişkin testler vb.

Etkileşim Testi: Yazılım yamasında kod geliştirilen dosyaların ya da prosedürlerin beraber çalıştığı diğer dosyalar ve prosedürlerin kullanılmasını da içerecek şekilde test senaryolarının koşturulmasıdır.

Trafik Testi: Müşteri sahasındaki canlı haberleşme trafiğinin bir simülasyonunu, laboratuvar ortamında gerçekleştirerek herhangi bir probleme yol açmadığının kanıtlanmasıdır.

Hata Testi: Yazılım yamasındaki hata durumu olarak tasarlanan kod parçacıklarının test edilerek sistemin geneline bir yan etki olup olmadığının doğrulanması.

Sağlamlık Testi: Yazılım yamasının birden fazla yüklenip çıkarılması şeklinde yapılan testtir.

5.4 Kalite Kontrolü

Kalite bir yazılım ürününde olmazsa olmaz bir parametredir. Hazırlanan yazılım yamasının düzgün bir şekilde geliştirildiğinin ve kalite açısından gerekli yeterlilikleri sağlayıp sağlamadığına bakılması gerekmektedir. Kalite kontrolü aşamasının en önemli amacı sahaya gönderilen yazılım yamalarındaki hata oranını minimumda tutmaktır. TL9K standartları baz alınarak yapılan çalışmada hedef, 6 aylık hata ortalamasının %0,5 ten küçük olmasıdır [9]. Yani her yazılan 1000 yazılım yamasında hatalı çıkan yama sayısının en fazla 5 olması beklenmektedir.

Kalite kontrolü anlamında temel olarak üzerinde durulan noktalar

- Yazılım yamasının kullanımına ilişkin rehber dosyasının düzgün şekilde oluşturulması
- Test planı ve sonuçların düzgün şekilde dokümante edilmesi
- Sağlamlık testindeki tüm adımların (yama yükleme, çıkarma vb.) uygulanması

şeklindedir.

5.5 Müşteri Tarafından Doğrulama

Hazırlanan yama yazılımının problemi raporlayan müşteri tarafından test edilme aşamasıdır. Müşteri problemin yaşandığı telekom santraline yamayı yükler ve ilgili test senaryosunu koşarak problemin çözülüp çözülmediğini doğrular. Eğer problem çözülmezse yama yeniden hazırlanır; problem çözülürse yamanın 1 hafta daha santralde yüklü olarak kalması sağlanarak trafik altında başka bir probleme neden olup olunmadığından emin olunur. 1 hafta sonunda yazılım yaması başarılı bir şekilde sahada çalışırsa tüm müşterilere otomatik olarak gönderilir.

5.6 Çoğullama

Çoğullama olarak ifade edilmek istenen, yazılım yamasının hazırlandığı işletim sistemi sürümü ile açık olan işletim sistemi sürümü arasında kalan sürümlere aynı çözümün yama şeklinde hazırlanmasıdır. Böylece farklı müşterilerden raporlanma ihtimali olan bu problem otomatik olarak çözülmüş olacaktır.

6 Sonuçlar

Bu makalede, yüksek kapasiteli IP tabanlı telekomünikasyon santrallerindeki merkezi bileşene ilişkin yazılım mimarisi, yazılım yaması modeli ve yazılım yamasında uygulanan proses konuları anlatılmıştır. Yazılım yaması yöntemi ile telekom santralinden gelen yazılım hataları en kısa sürede çözülmüş olacaktır. Yazılım yamasında kullanılan "hitless" yöntemi sayesinde gerçek zamanlı çağrı trafiğine herhangi bir zarar verilmeyecektir. Bu da müşteri memnuniyeti açısından oldukça önemlidir.
Yazılım yaması yönteminin kolay uygulanabilirliği müşterilerden gelecek yeni taleplerin karşılanmasında da önem arz etmektedir. Yazılım yamasında bakım ve işletim maliyetinin düşük olması da müşteriler açısından önemli bir göstergedir.

Kaynakça

1. Stolikj, M., Cuijpers, P.J.L., and Lukkien, J.J. "Patching a patch - software updates using horizontal patching" IEEE trans. On Consumer Electronics, vol. 59(2), pp. 435-441, (2013)
2. Foxall, D.G., Joliat, M.L., Kamel, R.F., ve Miceli, J.J. "Protel: a high level language for telephony", The IEEE Computer Society's Third International Computer Software and Applications Conference, 1979.
3. http://en.wikipedia.org/wiki/Pascal_(programming_language)
4. http://en.wikipedia.org/wiki/ALGOL_68
5. Cashin, P. M., Joliat, M. L., Kamel,R. F. ve Lasker, D. M., "Experience with a modular typed language: PROTEL", Proceedings of the 5th international conference on Software engineering ICSE '81, pp.136-143, (1981).
6. Ali, S.R., "Software patching in the SPC environment and its impact on switching system reliability" IEEE Journal on Selected Areas in Communications, vol. 9(4), pp. 626-631, (1991)
7. Utas G., "Robust Communications Software: Extreme Availability, Reliability and Scalability for Carrier-Grade Systems" John Wiley & Son, England (2004)
8. Byers, D., and Shahmehri, N., "Design of a Process for Software Security," The Second International Conference on Availability, Reliability and Security (ARES 2007), pp. 301 – 309, Vienna, (2007)
9. http://www.tl9000.org/tl_resources/meas_lib/Peer_Review_Defect_Tracking.doc