BMBAD BJESR
Mühendislik Bilimleri ve Araştırmaları Dergisi
Journal of Engineering Sciences and Researches

Tag: Blokzincirinden Esinlenilen Veri Yapısı

Tag: A Blockchain Inspired Data Structure

[1*]Alpay DORUK, [2]Savaş TAKAN, [3]Fatih SOYGAZİ, [4]Zeynep DEMİRTAŞ, [5]Elnur AKKURT

[1]Bandırma Onyedi Eylül Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi, Bandırma/Balıkesir, Türkiye,
[2]Ankara Üniversitesi, Mühendislik Fakültesi Yapay Zekâ ve Veri Mühendisliği Bölümü, Ankara, Türkiye
[3]Aydın Adnan Menderes Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Aydın/TÜRKİYE
[4,5]Bursa Uludağ Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü. Bursa/TÜRKİYE

[1]adoruk@bandirma.edu.tr, [2]stakan@ankara.edu.tr, [3]fatih.soygazi@adu.edu.tr,
[4]031990080@ogr.uludag.edu.tr, [5]031990053@ogr.uludag.edu.tr

Araştırma Makalesi/Research Article

ARTICLE INFO

Article history

Received : 5 December 2022
Accepted : 12 January 2023

Keywords:

Data Structure, Blockchain, Software Engineering, Modifiability, Reliability

ABSTRACT

In this study, the data structure of the blockchain is discussed from a software engineering perspective. The traceable, immutable, and provable nature of the data structure of the blockchain is among the issues to be dealt with in terms of software engineering. However, it is difficult to use in software development because the conventional data structures used in blockchain are intertwined with other blockchain-based technologies. In this article, the data structure is separated from blockchain technologies and made accessible to software engineers for various purposes. In doing so, numerous software engineering problems related to the blockchain data structure have been addressed and solutions developed. The proposed data structure has been validated and evaluated by comparing it with the traditional blockchain data structure. It is observed that the time/space complexity of our proposed data structure reduces on a logarithmic scale.

MAKALE BİLGİSİ

Makale Tarihleri

Gönderim : 5 Aralık 2022
Kabul : 12 Ocak 2023

Anahtar Kelimeler:

Veri yapısı, Blokzinciri, Yazılım Mühendisliği, Değişmezlik, Güvenilirlik.

ÖZET

Bu çalışmada blokzincirinin veri yapısı yazılım mühendisliği bakış açısından ele alınmaktadır. Blokzincirinin veri yapısının izlenebilir, değiştirilemez ve kanıtlanabilir doğası, yazılım mühendisliği açısından ilgilenilecek konular arasında görülmektedir. Ancak blokzincirinde kullanılan geleneksel veri yapıları diğer blockzincir tabanlı teknolojilerle iç içe olduğu için yazılım geliştirmede kullanımı zordur. Bu makalede, veri yapısı blokzinciri teknolojilerinden ayrılmış ve yazılım mühendisleri için farklı amaçlar doğrultusunda kullanılabilir hale getirilmiştir. Bunu yaparken, blokzinciri veri yapısıyla ilgili çok sayıda yazılım mühendisliği sorunu ele alınmış ve çözümler geliştirilmiştir. Önerilen veri yapısı, geleneksel blokzinciri veri yapısı ile karşılaştırılarak doğrulanmış ve değerlendirilmiştir. Önerilen veri yapısının zaman ve uzay karmaşıklığının blokzincirindeki veri seviyesindeki mekanizmalara göre logaritmik bir ölçekte azaltma sağladığı görülmüştür.

ORCID: [1*]0000-0002-6190-288X
[2]0000-0002-7718-9476
[3]0000-0001-8426-2283
[4]0000-0002-5038-8141
[5]0000-0001-8143-0851

1. GİRİŞ

Blokzinciri, Bitcoin ile hayatımıza girmiş olmakla beraber sadece finansal alandaki uygulamalara özgü bir teknoloji değildir. Farklı çalışma alanlarına özgü beklentileri karşılayabilecek bir yazılım mimarisini destekleyecek bileşenlerden meydana gelmektedir [1-8]. Blokzinciri teknolojisi için geliştirilen esnek yazılım bileşenleri, blokzincirinin farklı alanlarda rahatlıkla kullanılabilmesine olanak sağlamaktadır.

Blokzinciri [9-11], blokların işlemlerle ilgili bilgileri sakladığı birbirine zincirlenmiş blok listesidir. Bloklar, kendi bloğunun ve bir önceki bloğun bilgilerini değişmez bir şekilde saklar. Bu nedenle blokzincirine yeni bilgiler eklemek gerekiyorsa yeni bir blok ilave etmek gerekir. Farklı veriler içeren herhangi bir blok, diğerlerinden farklı bir özet (hash) değerine sahiptir. Blokzincirlerinde Merkle ağacı [12], Yönlü Döngüsüz Çizge (Directed Acyclic Graph (DAG)) [13] vb. gibi farklı veri yapıları kullanılabilmektedir.

Blokzincirinin birçok teknolojiden oluştuğunu söylemek hatalı olmaz. Ancak blokzincirinin var olan teknolojileri, her sistem için aynı değeri taşımayabilmektedir [14-15]. Örneğin, dağıtık sistemler, para gibi, bir hafızada az yer kaplayan değerler için uygun bir çözüm olabilir fakat yüksek miktarda hafıza gerektiren yapılarda problemler oluşabilmektedir [16]. Diğer bir deyişle, bu teknolojilerin birbirine girmiş olması kullanım açısından dezavantajlara neden olabilmektedir. Bu nedenle, blokzinciri teknolojisini bir bütün olarak almak yerine, ayrı ayrı değerlendirmek faydalı bir seçenek olabilmektedir. Bu açıdan, blokzinciri türü veri yapıları, blokzincirinin sahip olduğu diğer teknolojilerden arındırılırsa, bu yapı yazılım mühendisliğinde rahat ve esnek bir şekilde kullanılabilir hale gelmiş olur. Sahip olduğu tüm teknolojiler ile birlikte kullanıldığında, blokzinciri ile yapılamayacak pek çok işlem, bu sayede yapılabilir hale gelecektir.

Bir bloktaki verilerle ilgili ele alınması gereken bazı sorunlu durumlar vardır. Bu verilerle ilgili ilk sorun, işlem (transaction) verileri arasındaki ilişkiyi yakalamak için bir uygulamaya duyulan ihtiyaçtır. İşlemle ilgili genel bilgiler bir arada bu zincirlerde saklandığından, geleneksel blokzinciri yaklaşımı, veri odaklı değil işlem odaklıdır. İkinci sorun, uygulamaya özel başka bir çözüme ihtiyaç duyulan blokzincirinin veri bütünlüğü ile ilgilidir.

Veri modeli, bir uygulamada kullanılacak ana kavramları ifade etmek amacıyla oluşturulan veri temelini ifade etmektedir. Veri yapısı ise, herhangi bir verinin bilgisayar ortamında saklanabilmesi için ihtiyaç duyulan daha alt seviyede bir kavramdır. Veri modeli, veri yapısına ihtiyaç duymakla beraber o veri yapısının sahip olduğu özelliğe bağımlı biçimde davranmamalıdır. Bu durumda veri, bir nevi uygulama bağımlı bir şekilde ele alınmaktadır. Mesela, Ontoloji bir veri modelidir. Çizge bir veri yapısıdır. Bir veri modelini ifade etmek için veri yapısına ihtiyaç vardır. Ontoloji oluştururken mutlaka RDF gibi bir dille çizge oluşturmaya gerek yoktur. Örneğin Prolog ile de ontoloji oluşturulabilir [17]. Bu durum ontolojinin çalışma mekanizmasını etkilemez [18]. Ancak blokzincirinde mekanizma olarak özetleme operasyonu örneğin Merkle ağacı üzerinde yürüdüğü için blokzincirini de etkilemekte ve bu durum yazılım mühendisliğine aykırılık oluşturmaktadır. Bu noktada, veri modeli ile veri yapısının tamamıyla bağımsız olduğu bir yaklaşıma gidilmesinin daha uygun olacağı düşünülmektedir.

Çalışmada, yukarıda değinilen problemleri çözümlemek üzere, blokzincirinin yapısında değişiklik sağlanmıştır. Geleneksel blokzinciri yapısı <data, prevHash> şeklindedir. Bu yapıda veri, sadece data içerisinde saklanmaktadır. Yeni veri, prevHash kullanılarak, bir önceki veriye eklemlenir. Çalışmada önerilen yapıda ise veri üç kısma bölünmüştür. Birincisi Meta; ikincisi Data, üçüncüsü ise Tanık Çizgesi (Witnessing Graph) kısmıdır. Çalışmada Tag olarak adlandırılan yapı <meta, data, witnessing graph, stamp> şeklinde formüle edilmektedir. Meta, data ve tanık çizgesi yapıları, Tag yapısı içinde "gösterge" (pointer) olarak tutulmaktadır.

Önerilen modelin meta kısmında değişebilen veriler bulunmaktadır. Böyle bir veri yapısına duyulan ihtiyaç, bir resim tablosunun, kendisi değişmediği halde sahiplerinin değişebilir olmasının sağlanmasına benzetilebilir. Data kısmı ise değişemez veriyi barındırmaktadır. Bu analojiye göre tabloyu çizen ressam, meta kısmında değil, data kısmında bulunmalıdır. Çünkü bir tablonun ressamı sabittir, değişmez. Önerilen veri yapısında esneklik sağlayabilmek için, data ve meta birer gösterge olarak modellenmiştir.

Önerilen modelde Tag'ler veri taşıyabilir veya başka Tag'lere tanıklık yapabilir. Tanık Çizgesi ise tanıklardan oluşan bir yapıdır. Diğer bir deyişle, içerisinde tanık Tag'lerini barındırır. Bu tanıklar sayesinde özet fonksiyonu hesaplanır. Tanık Çizgesi'nin kendi içindeki yapısı çok farklı biçimlerde (array, set, map) mevcut olabilir. Bu sayede, her çeşit veri yapısı rahatlıkla gerçeklenebilmektedir. Örneğin Tag'ler bir harita'ya (map) aktarılabilir veya bir bağlı liste (linked list) ya da bir ağaç içerisinde rahatlıkla kullanılabilmektedir. Çünkü Tanık Çizgesi sayesinde, Tag'lere ulaşmak mümkündür. Yani önerilen veri yapısı, kanıt Tag'lerin göstergelerini kendi içerisinde tutmaktadır. Bu da onlara kolaylıkla erişimi mümkün kılmaktadır. Ayrıca, güncel blokzincirinden farklı olarak, önerilen yaklaşımda Tanık Çizgesi ile veri ağı birbirinden ayrı tutulmuştur. Tanık Çizgesi, tanık Tag'lerinden oluşmaktadır. Veri ağı ise Meta ve Data bölümlerinden oluşmaktadır. Diğer bir husus, örneğin bir Tag'in tanıkları olarak sahip olduğu bazı Tag'ların değiştirildiği anlaşılırsa, bu Tag tanıkların içerisinden çıkarılmaktadır. Bir Tag'in güvenilir olması için en az bir tane tanık Tag'ine ihtiyaç duyulmaktadır. Bu durumu sağlamayan Tag'lere şüpheli Tag adı verilmektedir.

Önerilen veri yapı türü, blokzincir teknolojisinin en önemli özelliklerinden biri olan "eklenebilir ama güncellenemez ve kanıtlanabilir" olma özelliğine bazı eklentiler ve değişiklikler getirilerek geliştirilmiştir. Önerilen veri yapısı, yaygın kullanılan bir dil olan Javascript kullanılarak örneklendirilmiştir. Bunun dışında, önerilen model örnek bir senaryo ile sınanmıştır. Böylece, "eklenebilir ama güncellenemez ve kanıtlanabilir" olma özelliğinin yanı sıra diğer değişikliklerle birlikte bu çalışmada önerilen veri yapısının; sağlık, hukuk, eğitim vb.

gibi pek çok farklı sektörde yazılım süreçlerinde rahatlıkla kullanılabilecek nitelikler sunduğunu söylemek mümkündür.

Çalışmanın en önemli katkıları olarak;

- Blokzincirinin güncel olarak kullandığı veri yapısının yazılım mühendisliği prensiplerine uygun olmayan kısımları ele alınarak bir veri yapısı önerilmektedir.
- Önerilen yenilikçi veri yapısı ile beraber çalışan bir güven mekanizması detaylandırılmaktadır. Bahsedilen güven mekanizması yardımı ile sistemdeki veri takibinin gerçekleştirilebildiğine dair örnekler verilmiştir.
- Veri esnekliği ve farklı tiplerdeki verinin blokzincirinde kullanılabilmesi için bölünebilen ve bölünemeyen verinin, önerilen veri yapısı yardımı ile nasıl oluşturulabildiği açıklanmaktadır.
- Önerilen veri yapısı yardımı ile zaman-uzay karmaşıklığı açısından ciddi bir iyileştirme yapılmaktadır.

2. LİTERATÜR ÖZETİ

Blokzincirinin finansal alandaki başarısı, farklı endüstriyel alanlarda blokzincirine olan ilgiyi arttırmıştır [19]. Ayrıca, dağıtık sistemi ve verinin değiştirilmesine karşı mekanizmaları sayesinde, blokzinciri teknolojisi yaygın bir kullanıma ulaşmıştır.

Blokzinciri, blokların aktarılması için farklı uçtan uca ağ yapıları kullanmakta ve bloklarda bulunan verileri işlemler (transaction) yardımı ile ağ üzerinde kullanıcılar arasında paylaştırmaktadır. Farklı blokzinciri ağları (Bitcoin, Ethereum, Ripple etc.) kendi mimarilerine uygun bir veri seviyesi inşa etmekte ve işlemlerin güvenli olarak gerçekleştirilmesi için bu seviyede kendi veri yapılarından yararlanmaktadır [20]. Yönlü Döngüsüz Çizge (DAG) veri yapısı tabanlı kullanılan çeşitli ağlar da mevcuttur [21].

Blokzincirinin mimari beklentilerini sağlayan farklı veri yapıları vardır. Bağlı bloklar, özet değerleri yardımıyla blokzincirini oluşturur. Bu durumda iki ihtimal bulunur. Bloklar, her durumda blok bağlantılarına göre ayrı ayrı çok sayıda özet değerine sahip olabilir veya bu blokları içeren her işlem sadece bir özet ile temsil edilebilir. İkincisi mantıklı çözümdür. Bu nedenle, Merkle ağacı (veya karma ağacı) [9], kökte sadece bir özet kalana kadar yaprak düğümlerinin özet değerlerini yapraklardan kök düzeyine kadar değerlendiren ikili bir ağaçtır. Her yaprak düğümü kendi özet değerini içerirken, yapraklar dışındaki diğer düğümler, ağacın alt seviyelerindeki özetlerin özetlerini içerir. Blokzincirindeki her bloğun kendi başlığı vardır ve önceki bloğun özet değerini gösterir. Tüm bu önceki özet değerleri bir Merkle ağacında birleştirilir ve bu zincirin doğrulanması için benzersiz bir özet elde edilir. Yönlü Döngüsüz Çizge (DAG), ağaçtaki çeşitli dalların tek bir ileri yönde başka bir dalı gösterebileceği belirli bir Merkle ağacı türüdür. Her veri yapısında, veriler sabit veriler olarak kabul edilir ve bloklardaki verilerin güncellenebilirliği ana odak değildir. Önerimiz, veri yapısından bağımsız bir çözüm ile bloklardaki verilerin güncellenebilirliğinin sağlanmasıdır.

Przytarski [22], blokzincirinin veri modelinin problemlerini ele almak için bir çözüm önermektedir. Anahtar/değer çiftini, bir varlığın niteliklerinin daha esnek bir şekilde tutulduğu *<entity, feature, value>* modeli ile değiştirir. Varlıklar ve özellikler arasındaki ilişki daha genel bir şekilde oluşturulmakta ve bu şekilde verilerin sorgulanması kolaylaştırılmaktadır. Bu yaklaşımda, varlık/nitelik ilişkisi iki farklı Merkle Ağacı ile tanımlanır ve bu yaklaşım için veri yapısı yüksek depolama kapasitesi gerektirirken, yaklaşımımız belirli bir veri yapısından bağımsızdır ve ayrıca işlem verilerinin güncellenebilir özelliği vardır.

Przytarski ve arkadaşları [23] blokzincirine girilen veri türlerini iki farklı şekilde tanımlamaktadır. Birincisi, “sabit nesne”dir. "Sabit nesne", belirli bir zamanda geçerli olan bir öğe türünü tanımlar. Bu nesneler, o olayın zaman damgasını bir özellik olarak tutarak zaman içindeki olaylara işaret eder. İkincisi “genişletilebilir nesne”dir. "Genişletilebilir nesne", zaman içinde değiştirilebilecek bir nesne türüdür. Bu tür nesneler bir dizi anahtar/değer çiftinden oluşur ve bu anahtar/değer çiftleri bir zincir olarak bir zaman damgasıyla bağlantılıdır. “Çıktı Formatı” Bölümünde gösterildiği gibi zaman içinde “genişletilebilir nesne” üzerindeki alanlar için yaratma, okuma, güncelleme ve silme (CRUD) işlemleri uygulanabilir. Ek olarak amacımız, yalnızca anahtar/değer çiftlerini değil, aynı zamanda gerekirse zincir yönelimli nesneleri güncellemek için genişletilmiş bir şekilde Taglerle esnek bir nesne oluşturmaktır. Bu nesneler, blokzinciri uygulamalarında iletişim sağlamak için JSON, RDF/N3, RDF/XML vb. sözdizimi gibi farklı formatlarda temsil edilebilir ve iletilebilir. Bir nesnenin modifikasyonu, değişmez bir zincirin oluşturulmasından bağımsız olduğu için, blokların (meta veri) önceki ve mevcut durumu, gerçek verileriyle birlikte saklanmalıdır. Ancak, bir zaman çizelgesindeki güncellenebilirlik de izlenmelidir. Bu nedenle, bir zincirdeki bir nesne hakkında kilitleme işlemi, nesnelerin güncellenebilirlik durumunu izlemeye devam etmemize yardımcı olur.

Verinin bir parçası olarak blokzincirindeki bloklar, bazı veri odaklı özellikleri kapsar. Veriler bir veri deposunda saklanmalıdır ve saklanan veriler, izlenmesi gereken verilerle ilgili bazı meta verilere sahip olabilir. Gerçek veriler ve meta veriler, geleneksel bir veritabanı sisteminde veya blokzincirinin kendisinde saklanmalıdır. Eberhardt ve Tai [24] bu veri depolama işlemini, verilerin blokzincirinde depolandığı "zincir içi" olarak sınıflandırırken, "zincir dışı", verilerin geleneksel veritabanında depolanmasını ve meta verilerin blokzincirinde tutulmasını önerir.

Burada dikkate alınması gereken ilk sorun, blokzincirinin doğrulanmasıdır. İkinci sorun ise yeni işlemlerin meta verileri sık sık güncellemesi ve bu durumun blokzincirini uzatmasıdır. Makalemizin katkısı, buradaki blokzinciri mekanizmasının farklı bölümlerinin “güncellenebilirliğini” sağlayan ve onları zarif bir şekilde doğrulayan bir veri yapısı önermesidir [22], [23], [24]. Eberhardt ve Tai [24], verilerin ve meta verilerin güncellenebilirliğini

blokzincirinin güveni için bir iletişim yükü ile belirleyerek "zincir içi" bir yaklaşım önermiştir. Bundan farklı olarak makalemizde geliştirilen veri yapısında esnek bir tanık mekanizması önerilmiştir.
Blokzinciri teknolojisinin ortaya çıkması ile beraber farklı özelliklere sahip blokzinciri yapıları geliştirildi. Geliştirilen teknolojiler birbirine çoğunlukla benzemekle beraber Tablo 1'de görüldüğü üzere bazı farklılıklar görülmektedir. Tablo 1'de yedi adet güncel blokzinciri teknolojisi ile beraber önerdiğimiz yöntemin özellikleri karşılaştırılmıştır. Karşılaştırma kriteri olarak blokzincirdeki verinin değiştirilip değiştirilemediği, farklı veri yapılarını desteklemesi, özet yapıları, veri ekleme/silme yetenekleri, blokzincirindeki blokların önceki bloğa bağımlılığı, bir kilitleme fonksiyonu içerip içermemesi ve ayrı bir veri yapısı olarak kullanılabilme yetisi göz önüne alınmıştır. Tablo 1'de görüldüğü üzere önerdiğimiz veri yapısı var olan blokzinciri veri yapılarından ciddi anlamda farklılıklar içermektedir. Önerdiğimiz veri yapısı, diğer çalışmaların bağımsız olarak bir veri yapısına dönüştürülememesi, çalışmamızın güncellenebilir veriye izin vermesi ve güvenilirlik açısından önerdiği Tanık Çizgesi yapısı ile özellikle esneklik açısından önde görülmektedir.

Tablo 1. Farklı blokzinciri veri yapılarının önerilen veri yapısıyla karşılaştırılması.

	Değişebilen Veri	*Veri Yapısı Desteği*	*Hash Yapısı*	*Ekleme ve Çıkarma Özellikleri*	*Önceki blok içinde tutuluyor mu?*	*Kilitleme /Açma Fonksiyonu*	*Çoklu İlişki Desteği*	*Yazılım alanında veri yapısı olarak kullanılabilir mi?*
Önerilen Model	Evet	Evet	Hepsi	Evet	Evet	Evet	Evet	Evet
Ethereum [25]	Hayır	Hayır	MerklePatricia Tree	Hayır	Hayır	Hayır	Hayır	Hayır
Ripple [26]	Hayır	Hayır	Merkle Tree	Hayır	Hayır	Hayır	Hayır	Hayır
Tezos [27]	Hayır	Hayır	Merkle Tree	Hayır	Hayır	Hayır	Hayır	Hayır
Hyperledger [28]	Hayır	Hayır	Merkle Tree	Hayır	Hayır	Hayır	Hayır	Hayır
Stellar [29]	Hayır	Hayır	Merkle Tree	Hayır	Hayır	Hayır	Hayır	Hayır
Hedera [30]	Hayır	Hayır	Hash graph	Hayır	Hayır	Hayır	Hayır	Hayır
IOTA [31]	Hayır	Hayır	Directed Acyclic Graph (DAG)	Hayır	Hayır	Hayır	Hayır	Hayır

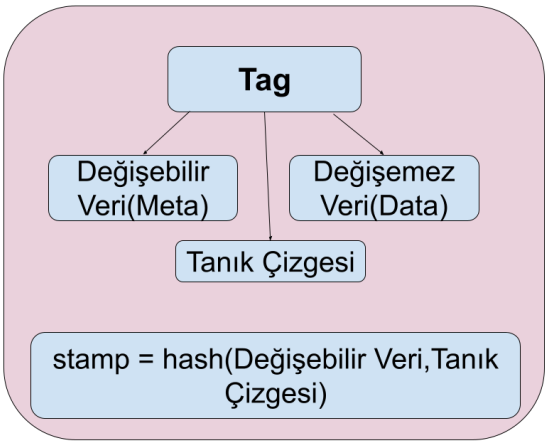
Bildiğimiz kadarıyla ve bu bölümde anlatıldığı üzere belirtilen hiçbir referansta blokzincirinde ihtiyaç dahilinde güncellenmesi gereken veriye dair bir çözüm üretildiği veya blokzincirinin veri seviyesindeki veri yapısının güven mekanizmasından ayrılarak yazılım mühendisliği çerçevesinde ele alındığı bir çalışma görülmemiştir. Çalışma ile bu noktalarda blokzincirine yenilikçi bir bakış açısı sunulmaktadır

3. ÖNERİLEN VERİ YAPISI

Tag adı verilen yapı, üç kısımdan oluşmaktadır. Birinci kısım, Data'nın göstergesini ifade eder. Bu gösterge, herhangi bir veri yapısına referans olabilir. Göstergenin maliyeti, düşüktür. Bu nedenle, değişmeyen veriyi yapı içerisinde tutmamak, bunun yerine göstergeleri tutmak, avantaj sağlamaktadır. İkinci kısım ise ilişkilerin ve güncellenebilir özelliklerin tutulduğu Meta göstergesidir. Bu kısım, genel olarak opsiyoneldir. Çünkü güncellenebilir kısımlar, yapının dışında da tutulabilmektedir. Önerilen veri yapısı, buna izin vermektedir. Üçüncü kısım ise Tanık Çizgesidir. Tanık Çizgesi, verinin değiştirilmediğinin kanıtıdır. Geliştirilen yapıda, Tanık Çizgesinin ve Data'nın özetlenmesiyle, o veri koruma altına alınmaktadır. Birçok blokzinciri önerisinden farklı olarak, oluşturulan yapıya fazladan tanık göstergeleri eklenebilme esnekliği sağlanmaktadır. Tanık Çizgesi, örnek çalışmalarda ifade edileceği üzere, birden fazla Tag'e sahip olabileceği gibi sadece tek bir Tag'e de sahip olabilmektedir. Bu da koruma seviyesinin güçlendirilmesine veya azaltılmasına imkan tanımaktadır. Şekil 1'de, Tag yapısı genel özellikleriyle gösterilmektedir.
Değişmezlik kontrolü, stamp yapısı ile sağlanmaktadır. Stamp, verinin tanık çizgesi ile özetlenmesi yolu ile hesaplanmaktadır. Örnek stamp hesaplama denklemi Denklem (1)'de görülmektedir. İstenildiği zaman stamp hesaplanmakta ve ilgili değişkene atanmaktadır. Daha sonra, değişiklik olup olmadığını kontrol etmek için yeni bir stamp değeri hesaplanmakta ve eski stamp değeri ile karşılaştırılmaktadır. Böylece, yapının değişip değişmediği, karşılaştırmanın sonucuna göre belirlenmiş olmaktadır. Çıkan sonuçlar eşitse yapının değiştirilmediği; sonuçlar eşit değilse yapının değiştirildiği anlamına gelmektedir. Bir diğer nokta ise eğer Tanık Çizgesi hesaplanırken bazı Tag'ler şüpheli pozisyona düşmüş ise o Tag'lerin hesaplamaya dahil edilmemesidir.

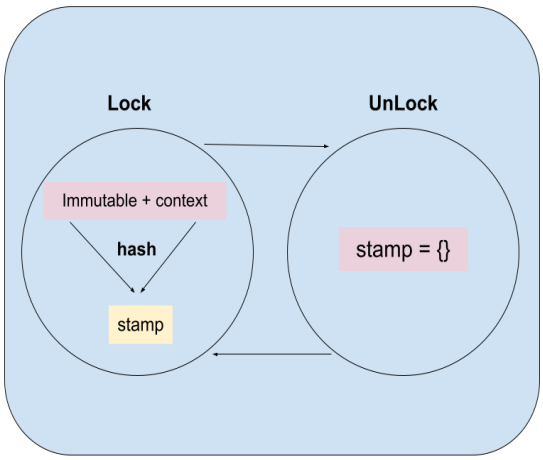
$$stamp = hash(data + \sum_{j=0}^{n} Tag_j) \quad (1)$$

Denklem 1, Tag'in verisi ile o Tag'in sahip olduğu Tag'lerin özetlenmesini ifade etmektedir. Denklem 1'de toplama işareti topolojik yapıdaki ilişkileri ayırt etmemektedir.



Şekil 1. Tag veri yapısının genel görünümü.

Fakat, gerçekte stamp değeri hesaplanırken derin öncelikli arama veya genişlik öncelikli arama yöntemiyle hesaplanmaktadır.
Önerilen yapıda, bunların dışında iki adet fonksiyon bulunmaktadır. Bunlardan birisi Lock() diğeri ise isLock()'tur. Lock() fonksiyonu sistemi kilitlemeye yaramaktadır. Lock() fonksiyonu, stamp'i hesaplayarak sisteme kaydetmektedir. Bu andan sonra, sistemdeki Tag'lerde ve veride bir değişiklik olduğunda farklı özet çıkacağından, verinin değişip değişmediği otomatik olarak ortaya çıkmaktadır. Bunu sorgulayan fonksiyon ise isLock'tur. isLock fonksiyonu veride bir değişiklik olup olmadığını kontrol etmektedir. Değinilen tüm noktalar, Tag veri yapısının temel özelliklerini oluşturmaktadır. Tag'in Lock ve Unlock durumunun genel yapısı Şekil 2'de gösterilmektedir.



Şekil 2. Lock() ve Unlock() fonksiyonu.

Aşağıda, Tag yapısının psödo kodu verilmektedir:

```
class Tag(meta, data, witnessing graph)
        doStamp()
          returnhash(witnessing graph + data)
        lock()
          stamp = doStamp()
        isLock()
          returnstamp == doStamp()
```

Tanık Çizgesi yapısı birçok şekilde olabilir. Aşağıda buna yönelik örnekler verilmektedir:

```
Tag1 = newTag("meta","data",Tag1)
Tag2 = newTag("meta","data",[Tag1, Tag2])
Tag3 = newTag("meta","data",{Tag1:Tag2, Tag4: Tag7})
subject = newTag("meta","data", {verb, object})
```

Data veya Meta ile birçok farklı model oluşturulabileceği aşağıda gösterilmektedir.

```
Tag1 = newTag({"has", Tag2},"data", null)
Tag2 = newTag({"is_a", Tag3},"data", Tag1)
Tag3 = newTag({"inherit_of",Tag1},"data", Tag2)
```

Çalışmada önerilen veri yapısına genel hatlarıyla değindikten sonra, önerilen yapı ile üretilebilecek örnek çıktılara Bölüm 4'te yer verilmektedir.

4. DEĞERLENDİRME

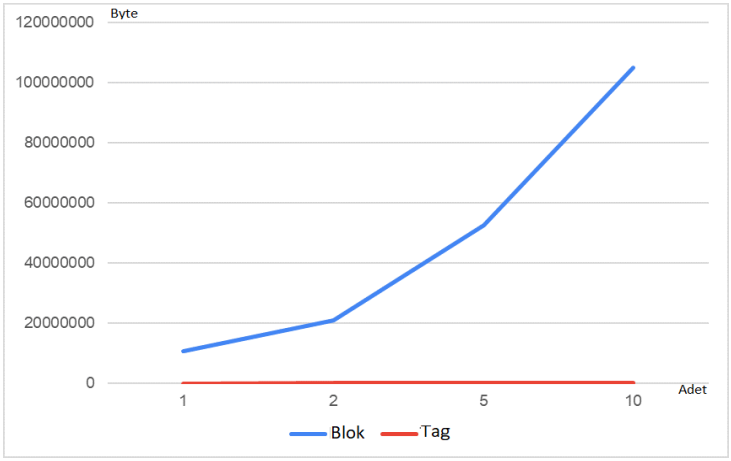
Geliştirilen model, özünde, blokzincirine bir alternatif olarak önerilmektedir. Çünkü senaryo bölümünde görülebileceği üzere, bazı yapıların blokzinciri altında gerçeklenebilmesi oldukça zordur. Aynı zamanda blokzinciri pek çok katmandan oluşan bir yapıya sahiptir. Şüphesiz her katman, bu proje ile uyumlu olmayacaktır. Bu nedenle projede, yalnızca gerekli katmanın kullanılmasına yönelinmiştir. Bu doğrultuda, blokzinciri veri yapısı ile bu çalışma kapsamında geliştirilen veri yapısı karşılaştırılarak, avantaj ve dezavantajlar ortaya koyulmaktadır. Şekil 3, önerilen veri yapısı için Tag mekanizmasının sağladığı avantajları göstermektedir. Her bloğun veya etiketin boyutu 10 MB olarak ve her adresin boyutu 256 bayt olarak ayarlanmıştır. Bu deney düzeneğinden görüldüğü gibi, işlenecek veriler blokların toplam miktarı ile doğrusal olarak artmaktadır. Ayrıca, Tag mekanizması tarafından işlenecek veriler, geleneksel blok zincirine kıyasla neredeyse yok denecek kadar azdır. Blokzincirinde her bir bloğun kapladığı alan aşağıdaki gibidir:

$$\textit{Blok boyutu} = \textit{blok sayısı} * (\textit{bir bloğun veri boyutu} + \textit{adres boyutu}) \quad (2)$$

Buna karşın, geliştirilen Tag mekanizmasında, her bir Tag'in kapladığı alan aşağıdaki gibidir:

$$\textit{Tag boyutu} = \textit{Tag sayısı} * (2*\textit{adres boyutu} + \textit{stamp boyutu}*\textit{adres boyutu} + \textit{Tag boyutu}*\textit{adres boyutu}) \quad (3)$$

Kolaylıkla görülebileceği gibi adres boyutları her zaman bir bloğun veri boyutundan çok daha küçüktür. Blokzincirinde büyük miktardaki verinin zaman ve alan karmaşıklığı O(n)'dir. Buna karşın, geliştirilen Tag mekanizmasının zaman ve alan karmaşıklığı O(1)'dir. Şekil 3'te bu durum gösterilmektedir.



Şekil 3. Blok ve Tag veri depolama karşılaştırması (Byte).

Tag mekanizmasının avantajları Tablo 2'de gösterilmiştir. Burada veri ile ilgili Meta bilgiler ayrı tutularak, bu Meta bilgilere rahatlıkla erişebilmek mümkün hale gelmiştir. Ayrıca Tag yapısı, blokzincirinden farklı olarak, değiştirilebilir veri yapısını da içerisinde barındırmaktadır. Geliştirilen modelde üç önemli unsur bulunmaktadır. Bunlardan ilki, verinin güncellenebilirliğidir. İkincisi, verilerin modelden ayrı tutulmasıdır, üçüncüsü ise güven mekanizmasıdır. Tüm bu katkılar sayesinde, blokzincirinin boyutu arttığında kolaylıkla performans artışı gözlenebilmektedir.

Tablo 2. Blok ve Tag mekanizması (Veri odaklı karşılaştırma).

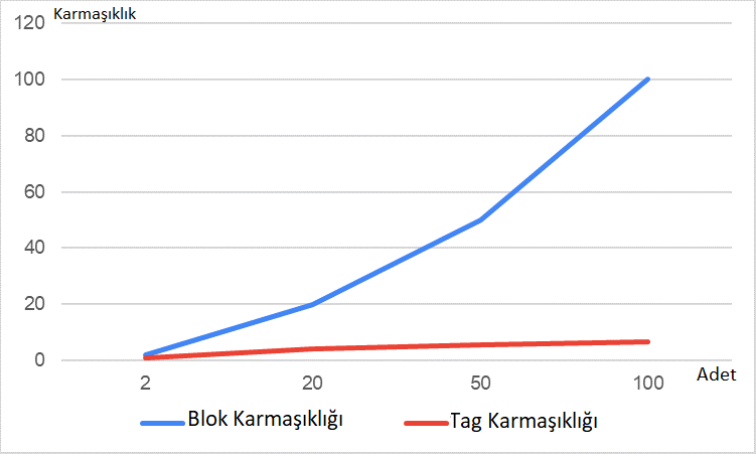
Tür	Data	Yedekleme	Değişme	Silme	Ekleme	Değişme
Blok	İçeride	Model ve veri	Yok	Veri + adres boyutu	Veri + adres boyutu	Veri+ adres boyutu
Tag	Dışarıda	Model veya veri	Var	0	4*adres boyutu	4*adres boyutu

Tablo 2'de, blokzinciri veri yapısı ile geliştirilen Tag yapısı karşılaştırılmıştır. Blokzincirinde her bir blok, bağlı liste yapısında saklanır. Blokzinciri veri yapısı, bunun dışında farklı bir saklama türüne izin vermemektedir. Bağlı listedeki operasyonlar esnek değildir ve güncellenemezdir. Buna karşın önerdiğimiz model, çeşitli yapıları barındırabildiği için blokzincirindeki bloklara göre esneklik ve güncellenebilirlik avantajlarına sahiptir. Bloklar arasındaki bağlantılar, blokzincirinde blokların farklı konumlara taşınmasını engellerken; önerdiğimiz Tag yapısı, çizge tabanlı olması ve esnekliği dolayısıyla, bir Tag'in farklı bir konuma taşınmasında herhangi bir probleme yol açmaz. Veri taşınabilirliği aynı zamanda indeksleme ile ilintilidir. Blokzincirindeki standart bloklarda kullanılan bağlı liste yapısında bu indekslemeyi oluşturmak oldukça zor iken; önerdiğimiz Tag yapısı, her türlü indeksleme mekanizmasını kolaylıkla içinde barındırabilecek özelliklere sahiptir.

Tablo 3. Blok ve Tag mekanizması (Veri yapısı odaklı karşılaştırma).

Tür	Veri Yapısı	Esneklik	Taşınabilirlik	İndeks	Arama
Blok	Bağlı Liste	Yok	Yok	Yok	O(n)
Tag	Herhangi	Var	Var	Var	O(log n)

Şekil 4'te, blokzincirindeki bloklar ile önerilen Tag yapısının arama performansları karşılaştırılmıştır. Blokzincirinde, bağlı liste yapısı nedeniyle blok sayıları arttıkça, arama karmaşıklığı doğrusal şekilde artar. Buna karşın önerilen Tag yapısı, her türlü indeksleme mekanizmasını barındırabildiğinden, örneğin indeksleme için B+Tree kullanılırsa, arama karmaşıklığı önemli oranda azalmaktadır.



Şekil 4. Blok ve Tag arama süresi karmaşıklığı.

Tablo 4, iki yaklaşımı güven odaklı bir perspektiften ele almaktadır. Buna göre her bloğun güveni, bir önceki ekli bloğa bağlıdır ve blokların güveni, derece veya hiyerarşik bir şekilde gerçekleştirilemeyebilir. Öte yandan, Tag'ler, birbirleri arasında farklı türde ilişkiler kurabilir ve farklı Tag'ler için farklı güven dereceleri bulunabilmektedir.

Tablo 4. Blok ve Tag mekanizması (Güven odaklı karşılaştırma).

	Güven Bağımlılığı (blok sayısı)	Güven Bağımlılığı (Yöntem)	Bakım onarım
Blok	1	Sabit	Blok düzeyinde bütünlük yok. Önceki bloğa bağlı.
Tag	n	Derece tabanlı	Gerekirse farklı Tag'ler işlenebilmektedir.

Geliştirilen modelin fiziksel olarak test edilmesi için, dört adet STM32 ve Lora Modülü kullanılmış ve testler gerçekleştirilmiştir. Konsensus algoritmaları bu aşamada entegre edilmemiş ancak bilginin saklanması, aranması ve transferi ile ilgili istemci-sunucu yapısında testler uygulanmıştır. Sonucunda, modelin fiziksel işlerliği tespit edilmiştir. Kodlar, GitHub linkinde[1] mevcuttur.

Görülebileceği gibi, geliştirilen Tag yapısının esnekliği ve güvenilirliği sayesinde, kullanışlı bir veri modeli oluşturabilmek mümkündür. Geliştirilen veri yapısı, aşağıda örnek senaryolarla test edilmiş ve işlevsellikleri ispatlanmıştır. Böylece, geliştirilen modelin çok çeşitli yazılım süreçlerinde kullanılabilir olduğu gösterilmiştir.

[1] https://github.com/kursuApp/tag/

4.1. Senaryo: Güncellenebilirlik

Değişmezliği göstermek amacıyla, aşağıda iki farklı Tag oluşturulmuştur. Burada Tag2'nin bağlamı Tag1'dir. Tag1'in ise bu aşamada herhangi bir bağlamı yoktur. Bu iki Tag'i oluşturmak için kullanılan psödo kodu aşağıdaki gibidir.

```
Tag1 = new Tag(meta, data,null)
Tag2 = new Tag(meta, data,Tag1)
```

Yukarıda gösterilen aşamada, lock fonksiyonu çalıştırılmamıştır. Dolayısıyla bunun sonucunda herhangi bir güncellenebilirlik mekanizması aktive edilmemiştir. Güncellenebilirlik mekanizması aktif hale getirilmediğinde, Stamp değerleri sıfır olacaktır. Bu yapının JSON format görünümü aşağıdaki gibidir:

```
Tag1: {meta, data, context: null, stamp:0}
Tag2: {meta, data, context: Tag1, stamp:0}
```

Aşağıdaki aşamada ise Tag2'nin lock fonksiyonu çağrılarak, sistem kilitlenmiş ve böylece değiştirilemez yapılmıştır. Tag2'nin lock fonksiyonunu çağırma komutu aşağıdaki gibidir:

```
Tag2.lock()
```

Bunun sonucunda Tag1'in Tag2'yi desteklediği yapı, sadece Tag2'nin Lock fonksiyonu çalıştırılsa dahi Tag1'i değiştirilmez yapmaktadır. Tag2'nin Lock fonksiyonu uygulandıktan sonraki yapının JSON format görünümü aşağıdaki gibidir. Burada dikkat edilmesi gereken nokta, stamp değerlerinin girilmiş olmasıdır. Bu stamp değerleri Tag1'in ve Tag2'nin özetidir.

```
Tag1: {meta, data, context: null, stamp:{ 40ka7}}
Tag2: {meta, data, context: Tag1, stamp:{ 50ba8}}
```

İstendiğinde, Meta değişkenine yeni bir Tag eklenebilir ve bu, Stamp değerini değiştirmez. Bu durum bağlamı oluştururken bize büyük esneklik sağlayacaktır. Çünkü bazı Tag'ler bağlam için önemliyken bazıları önemsizdir. Aşağıda, değiştirilebilir değerler değişse de Stamp değerlerinin değişmediği JSON formatında gösterilmiştir:

```
Tag1: {meta: 'first', data, context: null, stamp: 40ka7 }
Tag2: {meta: 'second',data, context: Tag1, stamp: 50ba8 }
```

İstendiğinde data değişkenine yeni Tag eklenirse, Stamp değerleri yeniden oluşturulur ve bu yeni değerler eski değerlerle karşılaştırıldığında, yeni oluşturulan Stamp değerlerinin eski Stamp değerlerinden farklı olduğu görülecektir. Burada dikkat çekmek istediğimiz nokta, Tag1'in data değeri değiştirildiğinde, Tag2'nin de değişeceğidir. Bu durum, aşağıda JSON formatında gösterilmiştir:

```
Tag1: {meta: 'first', data, context: null, stamp: 27b45 }
Tag2: {meta: 'second', data, context: Tag1, stamp: 13ca6}
```

Son olarak geliştirdiğimiz modelde, değişen Stamp değerlerine bakarak, hangi noktadaki Tag'in değiştiği tespit edilebilir.

4.2. Güvenilirlik

Geliştirilen veri yapısında, bir Tag'in güvenilirliği, onun bağlamı ile ilgilidir. Bağlamındaki güvenilir Tag sayısı ne kadar fazla ise o Tag o kadar güvenlidir denilebilir. Eğer bağlamında hiçbir Tag yoksa o Tag'e şüpheli Tag denmektedir. Aşağıdaki örnekte bir Tag'in hiçbir bağlamı olmadan oluşturulmasına örnek gösterilmiştir.

```
Tag1= new Tag(meta,data, null)
```

Aşağıdaki örnek, birden fazla bağlamın dizi yardımı ile nasıl ifade edileceğini göstermektedir. Burada Tag'in, iki tane tanık Tag'i bulunmaktadır.

```
Tag3 = new Tag(meta, data,[Tag1, Tag2])
```

Aşağıdaki durumda Tag'in bağlamı, map yapısıyla oluşmaktadır. Bağlamın içinde dört tane Tag bulunmaktadır.

```
Tag5 = new Tag(meta, data,{Tag1:Tag2, Tag3: Tag4})
```

Aşağıdaki Tag modeli, <subject, verb, object> paternine uygun bir şekilde gösterilmiştir. Burada Tag'ler, birbirlerini zincirleme olarak desteklemektedir.

```
object= new Tag(meta, data)
verb = new Tag(meta, data, object)
subject = new Tag(meta, data", verb)
```

Subject'e, verb ve object'in direk destek verdiği durumu ifade etmek istersek aşağıdaki gibi bir yapı oluşturabiliriz:

```
subject = new Tag(meta, data, {verb, object})
```

4.3. Senaryo: Tutarlılık

Bu bölümde, modele yönelik temel bir senaryoya yer verilmektedir. Burada öncelikle beş adet Tag yaratılmıştır. Bu Tag'ler yaratıldığı anda, henüz Tag'lerin bağlamı bulunmamaktadır. Tag'lerin yaratımı ile ilgili sözde kodlar:

```
Tag1 = new Tag('Femicide is on the rise.')
Tag2 = new Tag('A 36-year-old man stabbed his ex-fiancee to death.')
Tag3 = new Tag('The man who fatally stabbed his ex-fiancee received a 23-year prison sentence.');
Tag4 = new Tag('the man who stabbed her to death was released on good behavior during the first trial.')
Tag5 = new Tag('Women's rights activists demonstrated against the release in front of the court.')
```

Tag'ler yaratıldıktan sonra, neden sonuç ilişkisine göre aralarında bağlam oluşur. Eğer bir Tag'in bağlamında herhangi bir Tag yoksa, bu Tag güvenilir değildir. Örneğin aşağıdaki kodda, birinci Tag'i destekleyen ikinci, üçüncü ve dördüncü Tag'lerdir. Bu Tag'lerin hepsinin yok olması, birinci Tag'in tanık çizgesini ortadan kaldıracak, yani Tag bağlamsız kalacak ve güvenilir olmaktan çıkacaktır. Aşağıdaki kodda, Tag'lerin neden sonuç ilişkileri girildikten sonra, Tag'ler Lock fonksiyonu ile kilitlenmiştir. Bu durumda, Tag'lerin kendisinde ya da onu destekleyen Tag'lerden herhangi birinde bir hata ya da değişiklik olduğunda, model değişikliğin kaynağını bularak, o Tag'i bağlamdan çıkarır.

```
Tag1.addSupporter(Tag2)
Tag1.addSupporter(Tag3)
Tag1.addSupporter(Tag4)
Tag3.addSupporter(Tag2)
Tag4.addSupporter(Tag2)
Tag4.addSupporter(Tag3)
Tag5.addSupporter(Tag2)
Tag5.addSupporter(Tag3)
Tag5.addSupporter(Tag4)
Tag5.lock();
```

Oluşturulan modelde daha sonra, Tag2'nin hatalı olduğu ortaya çıkmış ve bunun sonucunda, hatalı olduğu tespit edilen Tag, güven ağından çıkarılmıştır. Sonrasında, Tag2 ile desteklenen Tag3 güvenilir olmaktan çıkmıştır. Tag3'ün de güven ağından çıkmasıyla, Tag4 desteksiz kalmış ve güvenilir olmaktan çıkmıştır. Bunun sonucunda Tag4 de güven ağından çıkarılmıştır. Böylece Tag1 desteksiz kalmış ve güven ağından çıkmıştır. Bu durum, Tag5'i desteksiz kılmış ve güvenilir olmaktan çıkarmıştır. Sonuçta, örnekteki tüm Tag'ler güvenilir olmaktan, yani tanık çizgesinden çıkmıştır. Bu durum tutarlılığın çalışmasına bir örnektir.

```
Zanlı eski nişanlısını öldürmedi. (Tag2)
Tag2'yi destekleyen yok, silinecek.
Tag3'ü destekleyen yok, silinecek. (Tag2)
Tag4'ü destekleyen yok, silinecek. (Tag2, Tag3)
Tag1'i destekleyen yok, silinecek. (Tag2, Tag3, Tag4)
Tag5'i destekleyen yok, silinecek. (Tag2, Tag3, Tag4)
```

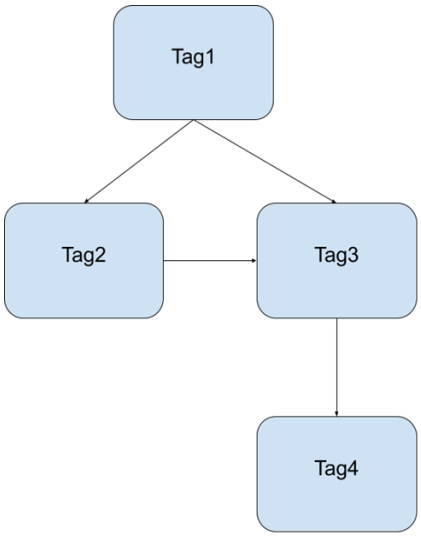
Eğer bir Tag'in bağlamı içerisinde hiçbir Tag bulunmuyorsa, o Tag'in varlığı kabullere bağlıdır. Diğer pek çok Tag, bu ilk etapta hiç desteği bulunmayan Tag'lerden türeyebilir. Bu, sistemde bir problem yaratmaz. Modelde böyle bir Tag bulunabilir.

4.4. Senaryo: Bağlam

Aşağıda bir Tag'in bağlamını açıklayabilmek için dört adet Tag oluşturulmuştur.

```
Tag1 = new Tag(meta,data,null)
Tag2 = new Tag(meta, data, Tag1)
Tag3 = new Tag(meta, data, ,[Tag1, Tag2])
Tag4= new Tag(meta, data ,Tag3)
```

Yukarıdaki Tag'lerin aralarında oluşturdukları bağlam Şekil 5'te ifade edilmektedir. Burada Tag1'in bağlamı yoktur fakat modelde varlığını sürdürmektedir. Bu, Tag1'i doğrulayacak herhangi bir tanık Tag'inin bulunmadığı anlamı taşımaktadır.



Şekil 5. Tag'in bağlam yapısı.

Şekil 6'da görüleceği gibi, Tag4'ün bağlamı içerisinde Tag1, Tag2, Tag3 bulunmaktadır. Bağlamı hesaplarken özet değer, ağa özgü olacağından, yukarıdaki şekilde Tag4'ün bağlamı Tag1, Tag2, Tag3'e ve aralarındaki ilişkiye özgü olacaktır. Ayrıca, bir Tag'in birçok bağlamı olabileceğinden dolayı, bu şekilde istenilen özetler bir diziye atarak bağlamlar kümesi oluşturulması mümkün olmaktadır. Daha sonra, başka bir Tag'in bağlamı ile uyumlu olup olmadığı, bu özet değerlerine bakılarak anlaşılmaktadır.

4.5. Senaryo: Seçim

Örnek bir seçim senaryosunda, Tag yapısının nasıl uygulandığı görülebilir. Öncelikle iki aday Tag belirlenir. Oylayan Tag'ler, oylanan Tag'lere bağlanır. Oylamaya katılan her Tag, oylama sonrası sistemi kilitler. Böylece oylamanın değiştirilemezliği ve dolayısıyla, güvenirliği sağlanmış olmaktadır. Bu kilitlemeler sonucunda oluşan değerler, oylamanın sonucunda oluşan değerlerle karşılaştırıldığında, eğer değerler birbirine eşitse, seçim geçerli kabul edilmektedir. Eğer değerler birbirine eşit değilse, problemin hangi aşamadan kaynaklandığı, bağlam tarafından anında tespit edilebilmektedir. Aşağıda bu yapının nasıl gerçeklendiği gösterilmiştir.

```
Tag0 = new Tag(meta={}, data={kisi:'ADAY_A', secim_no:'sos421'}, context= null,stamp={})
Tag1 = new Tag(meta, data={kisi:'ADAY_B', secim_no:'sos421'}, context= null,stamp={})
Tag1 = new Tag(meta={}, data={oy_no:'85hch', secim_no:'sos421'}, context= Tag1 ,stamp={})
Tag2 = new Tag(meta={}, data={oy_no:'64das', secim_no:'sos421'}, context= Tag1,stamp={})
Tag3 = new Tag(meta={oy='ADAY_A'}, data={oy_no:'28shs', secim_no:'sos421'}, context=Tag0, stamp={})
```

Yukarıda iki kişi 'ADAY_B'ye, bir kişi de 'ADAY_A'ya oy vermiştir. Seçim numarası 'sos421' olarak belirlenmiştir. Oy numarası adayların sayısal kimliğini ifade eder, bu yapı özet fonksiyonu olduğu için, gizlilik korunmaktadır. Sistem şu anda lock() fonksiyonunu çalıştırmadığı için, korumada değildir ve özet değeri yoktur. Eğer lock() fonksiyonu çalıştırılırsa, oylama değiştirilemez hale gelecektir.
Şekil 6'da, Tag yapısının bir seçim senaryosuna nasıl uygulandığı görülebilir. Burada önemli nokta, Tag'lerin çok farklı yapıda tasarlanabilmesidir. Örneğin Şekil 6'da, belli bir zincir oluşturularak Tag yapısı, oyların problemsiz şekilde sayılmasını mümkün kılmaktadır.

Şekil 6. Seçimlerin Tag yapısıyla gerçekleştirilmesi.

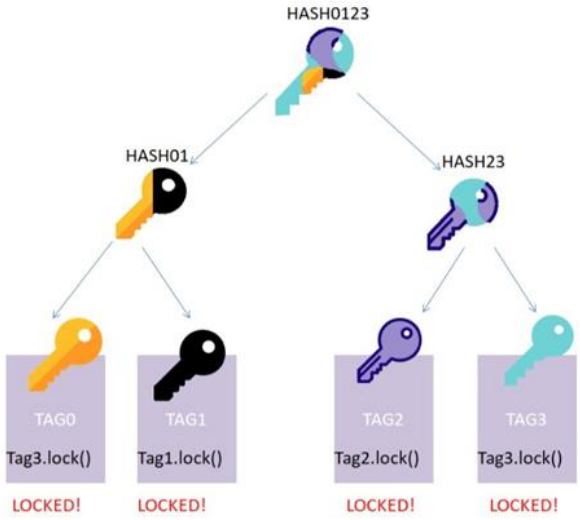
Her oylama gerçekleştirildikten sonra ya da belli farklı periyotlarda, Lock fonksiyonu çalıştırılmakta ve fonksiyonun döndürdüğü stamp değeri ilgili paydaşlarla paylaşılmaktadır. Seçim sonlandıktan sonra sistem, son özet değerinden geri dönerek diğer tüm özet değerlerini vermektedir. Bu değerler, muhataplar tarafından kontrol edildiğinde, eğer sistemin verdiği özet değerleri ile muhatapların daha önceden elde ettiği değerler birbirine eşit ise seçim geçerli kabul edilmektedir. Değil ise problemin hangi aşamadan kaynaklandığı, tag yapısı tarafından anında tespit edilebilmektedir. Blokzinciri pek çok teknoloji barındırdığı için böyle bir yapıyı, düşük veri miktarına rağmen efektif şekilde gerçekleştiremeyebilir. Çünkü muhatap sayısının sonsuza göre belirlenmesi ve aynı zamanda pek çok teknolojiyle birleşik gelmesi dolayısıyla güncel blokzinciri, çok basit eylemlerde çok ciddi tasarım süreçlerine ihtiyaç duyabilmektedir. Önerilen Tag yapısı, aynı işlemi önemli derecede daha az karmaşıklık ile gerçekleştirebilmektedir.

4.6. Senaryo: Merkle Tree

Blokzincirinde Merkle Tree çok kullanılan bir veri yapısıdır. Aşağıda bu yapının Tag veri yapısında gerçekleme yöntemi gösterilmiştir.

```
const tag0 = newTag('meta', 'data',null)
const tag1 = newTag('meta','data','',tag0)
const hash01 =hash( tag0.lock() , tag1.lock())
const tag2 = newTag('meta','data',tag0)
const tag3 = newTag(('meta','data',tag3)
const hash23 =hash( tag2.lock() , tag3.lock())
constroot =hash( hash01, hash23)
```

Yukarıdaki kodun gösterimi, Şekil 7'de ifade edilmiştir. Görselde kullanılan yapı, Tag yapısıdır. Burada, tanık çizgesinin yapısı gösterilmektedir. Bu durum, yukarıdaki kodun veri yapısından farklıdır.

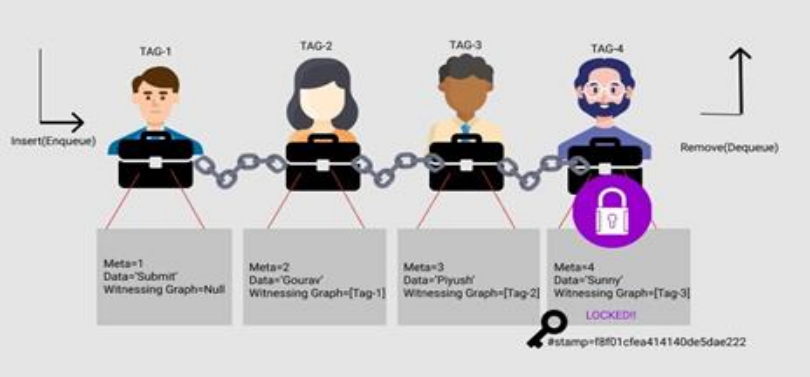


Şekil 7. Tag yapısıyla MerkleTree gösterimi.

4.7. Senaryo: Priority Queue

Aşağıda, bir Öncelikli Kuyruk (Priority Queue) veri yapısı örneği gösterilmektedir. Bu örneğin gösterilmesindeki amaç, rahatlıkla farklı veri yapılarının bu Tag sistemi içerisinde var edilebilmesidir. Burada önceliklendirmeler değişken olduğu için Meta'nın içerisinde yer almaktadır.

```
priorityQueue = newPriorityQueue()
tag1 = newTag(2,"Sumit", null)
tag2 = newTag(1,"Gourav", [tag1])
priorityQueue.enqueue(tag1)
priorityQueue.enqueue(tag2)
priorityQueue.dequeue()
tag3 = newTag(2,"Sunil", [block5])
priorityQueue.enqueue (tag3)
block3.isLock()
```



Şekil 8. Priority Queue ile Tag yapısı.

Priority Queue ile ilgili açıklama, Şekil 8'de görselleştirilmiştir. Burada dikkat çeken nokta, kullanım kolaylığı açısından işlemlerin normal veri yapılarına benzerlik göstermesidir. Fakat önerilen veri yapısı, istendiği anda sistemi kilitleyebilecek, istendiği anda sistemi eski haline döndürebilecek özelliklere sahiptir. Diğer bir nokta, önerilen veri yapısının kanıtlanabilirlik özelliği taşımasıdır.

4.8. Senaryo: Kan basıncı

Şekil 9'da kan basıncı, IoT teknolojileri ile ölçüldükten sonra bu bilginin blokzincirine atılması gösterilmektedir. Bu senaryo, Şekil 6'daki seçim senaryosundan farklı olsa da bu yapı da aynı mantık ile karmaşıklığı düşük bir şekilde kolaylıkla gerçeklenebilir.

Şekil 9. Kan basıncı ölçümlerinin Tag yapısı ile gösterimi.

Yukarıda değinilen tüm örneklerden yola çıkarak, önerilen veri yapısının güncel blokzinciri teknolojisi ile karşılaştırıldığında hantal olmayan, karmaşıklığı az, algılanması ve uygulanması yazılım mühendisliği açısından elverişli, güvenilir, basit ve esnek olduğunu söylemek mümkündür.

5. SONUÇ

Güncel blokzinciri teknolojileri, her geçen gün genişleyen bir kullanım olanağı sunsa da, temelde bazı problemler barındırmaktadır. İşlemler ve bilgiler bir arada blokzincirlerde saklandığından, güncel blokzinciri yaklaşımı, veri odaklı yaklaşıma uygun değildir. İkinci sorun, uygulamaya özel başka bir çözüme ihtiyaç duyulan blokzincirinin veri bütünlüğü ile ilgilidir. Ayrıca veri modeli, veri yapısına ihtiyaç duymakla beraber o veri yapısının sahip olduğu özelliğe bağımlı biçimde davranmamalıdır.

Çalışmada, yukarıdaki problemleri çözümleyebilmek amacıyla, yeni bir veri yapı türü önerilmiştir. Önerilen veri yapı türü, blokzincir teknolojisinin bize göre en önemli özelliklerinden biri olan "eklenebilir ama değiştirilemezlik ve kanıtlanabilirlik" özelliklerine bazı eklentiler ve değişiklikler getirilerek geliştirilmiştir. Geliştirilen veri yapısı Tag'lerden oluşmaktadır. Tag yapısının yaratımında Meta, Data ve Tanık Çizgesi referansları alınır. Meta'da, güncellenebilenler; Data'da güncellenemeyenler, Tanık Çizgesi'nde ise Tag'ler bulunmaktadır. Önerilen veri yapısında, Tag'lere esneklik kazandırıldığı için istenilen veri yapısının içerisinde var olabilmeleri mümkün hale gelmektedir. Çalışmada önerilen veri yapısı, Tag'lerin göstergelerini kendi içinde tutar. Bu da onlara kolaylıkla erişimi mümkün kılmaktadır.

Önerilen model ile diğer pek çok yazılım çözümü için yeni bir veri yapısı oluşturmak hedeflenmiştir. Buradan hareketle önerilen veri yapısı, yaygın olarak kullanılan bir programlama dili olan Javascript kullanılarak modellenmiştir. Oluşturulan model, geleneksel blokzinciri modeli ile karşılaştırılmış ve yazılım mühendisliği açısından eksiklikleri giderilmiştir. Daha sonra bu model, bilgisayar bilimi için önemli olan veri yapılarına uyarlanmıştır. Bunun dışında veri yapısı olarak nasıl kullanılacağı, örnek senaryolarla sınanmıştır.

Sonucunda, günümüzde yazılım mimarisi açısından eksikliği hissedilen, güncellenebilirlik ve güncellenemezlik özelliklerine esneklik katacak, istenildiği ölçüde Tag eklenmesine olanak sunan ve sağlık, hukuk, eğitim, güvenlik gibi pek çok alanda rahatlıkla kullanılabilecek özgün bir veri yapısı geliştirilmiştir. Geliştirilen veri yapısı, blokzincirlerinin yeni tasarımlarında kullanılabileceği gibi, yazılım alanında izlenebilir ve gözlemlenebilir özelliklere sahip bir veri yapısı şablonu sunmaktadır. Son olarak, geliştirilen veri yapısının, esnek kullanım olanaklarının yanı sıra bundan sonraki çalışmalar için perspektif ve kaynak oluşturması beklenmektedir

Yazar Katkıları

Yazarlar makale çalışmasına eşit katkı vermiştir.

Çıkar Çatışması

Makale yazarları aralarında herhangi bir çıkar çatışması olmadığını beyan ederler

KAYNAKÇA

[1] N. El Madhoun, J. Hatin, and E. Bertin "Going beyond the blockchain hype: In which cases are blockchains useful for its applications", 3rd Cyber Security in Networking Conference (CSNet), IEEE, pp. 21-27, 2019.

[2] M. Belotti, N. Božić, G. Pujolle, and S. Secci "A vademecum on blockchain technologies: When, which, and how", IEEE Communications Surveys and Tutorials, vol. 21 no. 4, pp. 3796-3838, 2019.

[3] P. P. Ray, D. Dash, K. Salah, and N. Kumar "Blockchain for IoT-based healthcare: background, consensus, platforms, and use cases", IEEE Systems Journal, vol. 15, no. 1, pp. 85-94, 2020.

[4] O. Ali, A. Jaradat, A. Kulakli, and A. Abuhalimeh "A comparative study: Blockchain technology utilization benefits, challenges and functionalities", IEEE Access, vol. 9, pp. 12730-12749, 2021.

[5] Y. Liu, X. Ma, L. Shu, G.P. Hancke, and A.M. Abu-Mahfouz "From Industry 4.0 to Agriculture 4.0: Current status, enabling technologies, and research challenges", IEEE Transactions on Industrial Informatics, vol. 17, no. 6, pp. 4322-4334, 2020.

[6] P. Bhattacharya, S. Tanwar, U. Bodkhe, S. Tyagi, and N. Kumar, "Bindaas: Blockchain-based deep-learning as-a-service in healthcare 4.0 applications", IEEE transactions on network science and engineering, vol. 8, no. 2, pp. 1242-1255, 2019.

[7] M.B. Mollah, J. Zhao, D. Niyato, Y.L. Guan, C. Yuen, S. Sun, ... and L.H. Koh "Blockchain for the internet of vehicles towards intelligent transportation systems: A survey", IEEE Internet of Things Journal, vol. 8, no. 6, pp. 4157-4185, 2020.

[8] J. Bao, D. He, M. Luo, and K. K. R. Choo "A survey of blockchain applications in the energy sector", IEEE Systems Journal, vol. 15, no. 3, pp. 3370-3381, 2020.

[9] M. Di Pierro "What is the blockchain?", Computing in Science and Engineering, vol. 19, no. 5, pp. 92-95, 2017.

[10] T. Ahram, A. Sargolzaei, S. Sargolzaei, J. Daniels, and B. Amaba "Blockchain technology innovations", IEEE Technology and Engineering Management Conference (TEMSCON), IEEE, pp. 137-141, 2017.

[11] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda, and V. Santamaria "To blockchain or not to blockchain: That is the question", IT Professional, vol. 20, no. 2, pp. 62-74, 2018.

[12] R.C. Merkle "Secrecy, authentication, and public key systems". Stanford University, 1979.

[13] N. Christofides "Graph theory: An algorithmic approach (Computer science and applied mathematics)", Academic Press, Inc, 1975.

[14] J. Golosova, and A. Romanovs "The advantages and disadvantages of the blockchain technology". IEEE 6th workshop on advances in information, electronic and electrical engineering (AIEEE), IEEE, pp. 1-6, 2018.

[15] A. Chauhan, O.P. Malviya, M. Verma, and T.S. Mor "Blockchain and scalability", IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), IEEE, pp. 122-128, 2018.

[16] Q. Zhou, H. Huang, Z. Zheng, and J. Bian "Solutions to scalability of blockchain: A survey", IEEE Access, vol. 8, pp. 16440-16455, 2020.

[17] J. Wielemaker, Z. Huang, and L. Van Der Meij "SWI-Prolog and the web", Theory and practice of logic programming, vol. 8, no. 3, pp. 363-392, 2008.

[18] R. Mizoguchi, and K. Kozaki "Ontology engineering environments", Handbook on Ontologies, Springer, Berlin, Heidelberg, pp. 315-336, 2009.

[19] I. Konstantinidis, G. Siaminos, C. Timplalexis, P. Zervas, V. Peristeras, and S. Decker "Blockchain for business applications: A systematic literature review", International Conference on Business Information Systems, Springer, Cham., pp. 384-399, 2018

[20] C.G. Akcora, Y. R. Gel, and M. Kantarcioglu "Blockchain networks: Data structures of Bitcoin, Monero, Zcash, Ethereum, Ripple, and Iota", Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 12, no. 1, e1436, 2022.

[21] H. Pervez, M. Muneeb, M.U. Irfan, and I. U. Haq "A comparative analysis of DAG-based blockchain architectures", 12th International conference on open source systems and technologies (ICOSST), IEEE, pp. 27-34, 2018.

[22] D. Przytarski "Using Triples as the Data Model for Blockchain Systems", BlockSW/CKG@ ISWC, 2019.

[23] D. Przytarski, C. Stach, C. Gritti, and B. Mitschang "Query Processing in Blockchain Systems: Current State and Future Challenges", Future Internet, vol. 14, no. 1, 2021.

[24] J. Eberhardt, and S. Tai "On or Off the Blockchain", Insights on off-chaining computation and data, European Conference on Service-Oriented and Cloud Computing, Springer, Cham, pp. 3-15, 2017.

[25] V. Buterin "A next-generation smart contract and decentralized application platform", White Paper, vol. 3, no. 37, pp. 2-1, 2014.

[26] D. Schwartz, N. Youngs, and A. Britto "The ripple protocol consensus algorithm", Ripple Labs Inc White Paper, vol. 5, no. 8, p. 151, 2014.

[27] L.M. Goodman "Tezos—a self-amending crypto-ledger", White paper. URL: https://www.tezos.com/static/papers/white paper. 2014.

[28] D. Voell, F.L.N., Gaski, R. Jagadeesan, R. Khasanshyn, H. Montgomery, S. Teis, ... and M. Bowman "Hyperledger whitepaper", Published:https://wiki.hyperledger. org/groups/whitepaper/whitepaper-wg, 2016.

[29] D. Mazieres "The stellar consensus protocol: A federated model for internet-level consensus", Stellar Development Foundation, vol. 32, pp. 1-45, 2015.

[30] L. Baird, M. Harmon, and P. Madsen "Hedera: A governing council and public hashgraph network", The trust layer of the internet, whitepaper, vol. 1, pp. 1-97, 2018.

[31] S. Popov "The tangle", White paper, vol. 1, no. 3, 2018.