NPM Ekosisteminde Yönlü Karmaşık Ağ Analizi

Yusuf Talha ARABACI

22 Ekim 2025

Özet

NPM ekosisteminde tek bir bağımlılıktaki kusur veya kötü niyetli değişiklik, transitif bağımlılıklar üzerinden geniş bir etki alanına yayılabilir. Bu rapor, paket içeriklerinden ziyade paketler arası ilişkilerin *topolojik* yapısına odaklanır. Bağımlı → bağımlılık yönünde kurulan yönlü ağ üzerinde in-degree, out-degree ve betweenness merkeziyetleri hesaplanır; bu ölçüler min–max normalizasyonu ile bir *bileşik risk skoru*na dönüştürülür. Ayrıca, en kritik düğümlerin çıkarılmasıyla ağın bağlanırlığı üzerinden bir *sağlamlık* değerlendirmesi yapılır. Tüm görseller ve tablolar results/ dizinindeki çıktılara dayanır ve ilgili başlıklar altında sunulur.

1 Giriş ve Amaç

Modern yazılım tedarik zincirinde tek bir bağımlılıktaki hata veya kasıtlı bir saldırı, transitif bağımlılıklar üzerinden yüzlerce hatta binlerce projeye yayılabilir. NPM ekosistemi; büyük ölçek, hızlı sürüm döngüleri ve yoğun bağımlılık grafiği nedeniyle bu tür zincirleme risklere özellikle açıktır. Bu rapor, paket içeriğinden ziyade paketler arası ilişkinin topolojik yapısına odaklanır: Bir paketin ağ içindeki konumu ve bu konumun sistemik etkileri nicel olarak değerlendirilir.

Bu çalışmanın hedefi, NPM bağımlılıklarını yönlü bir ağ olarak modelleyip yapısal riski görünür kılmaktır. Bunun için:

- Bağımlı → bağımlılık yönünde kurulan ağ üzerinde, *omurga* ve *köprü* niteliğindeki paketler merkeziyet ölçütleriyle (in-degree, out-degree, betweenness) belirlenir.
- Bu ölçütlerden türetilen *bileşik risk skoru* ile paketler karşılaştırılabilir biçimde sıralanır.
- Kritik düğümlerin çıkarılmasına dayalı *sağlamlık* göstergeleriyle (zayıf bileşen sayısı, en büyük bileşen boyutu vb.) ağın kırılganlığı nicel olarak değerlendirilir.

Bu yaklaşım, güvenlik değerlendirmesini yalnızca paket içi zafiyetlere indirgemek yerine, bağımlılık topolojisinden kaynaklanan yapısal riskin de hesaba katılmasını sağlar. Böylece bakımı yapılacak, yakından izlenecek veya kısıtlanacak paketler veri temelli biçimde önceliklendirilebilir.

2 Kuramsal Çerçeve ve Tanımlar

Yönlü Ağ (DiGraph). Düğümler paketleri, yönlü kenarlar ise "bağımlı → bağımlılık" ilişkisini temsil eder. Bir paketin başka bir paketi *kullanması*, bağımlı paketten bağımlılığa giden bir ok (out-edge) ile modellenir.

In-degree. Bir düğüme *gelen* kenar sayısıdır; başka kaç paket o pakete bağımlıdır. Yüksek in-degree, *yayılım potansiyelini* (tekil bir sorun çok projeyi etkileyebilir) ve *omurga* konumunu işaret eder.

Out-degree. Bir düğümden *çıkan* kenar sayısıdır; paketin kaç bağımlılığı vardır. Yüksek out-degree, *bağımlılık yüzeyinin geniş* olduğunu ve tedarik riskine maruziyetin arttığını gösterir.

Betweenness Merkeziyeti. En kısa yollar üzerinde bir düğümün *köprü* (aracılık) rolünü ölçer. Topolojik *boğaz noktaları*nı belirler; tek hata noktası riski taşıyan düğümleri ortaya çıkarır.

Bileşik Risk Skoru. Normalize edilmiş (min–max) ölçülerin ağırlıklı toplamıdır:

$$\text{risk}(n) = w_{in}\,\tilde{d}_{in}(n) + w_{out}\,\tilde{d}_{out}(n) + w_b\,\tilde{b}(n)$$

Burada $\tilde{x} = (x - x_{\min})/(x_{\max} - x_{\min})$ min–max normalizasyonudur (payda 0 ise $\tilde{x} = 0$). Ağırlıklar w_{in}, w_{out}, w_b, omurga duyarlılığı (in-degree), bağımlılık yüzeyi (out-degree) ve köprü rolü (betweenness) arasında vurguyu dengeler.

Sağlamlık (Robustluk) ve Kaskad Etkisi. Risk sıralamasına göre en kritik $k \in \{1, 3, 5\}$ düğüm çıkarılır; zayıf bağlanırlık bileşen sayısı, en büyük bileşen boyutu ve (mümkünse) ağ çapı ölçülür. *Kaskad etkisi*, bir düğümdeki sorunun transitif olarak etkileyebileceği paket sayısını ifade eder.

3 Veri ve Yöntem

Veri Toplama. Top N paket listeleri ve bağımlılıklar öncelikle ecosyste.ms API'lerinden; yedek olarak npm registry ve npms.io üzerinden alınır. Varsayılan olarak dependencies alanı kullanılır; isteğe bağlı olarak peerDependencies dahil edilebilir.

Ön İşleme. Paket adları normalize edilir; yinelenen kayıtlar ayıklanır; yönlü kenarlar bağımlı → bağımlılık yönünde oluşturulur. Sürüm tercihinde en güncel kararlı sürümler esas alınır.

Ağın Kurulumu. NetworkX ile DiGraph kurulup düğümler/kenarlar eklenir. Tüm düğümler için in/out-degree hesaplanır; büyük graflarda betweenness, örnekleme (k) ile hızlandırılır (rastgele kaynak düğüm seçimi).

Risk Skoru Hesabı. Ölçüler gözlenen aralıkları üzerinden min–max normalize edilir. Ağırlıklar (ör. $w_{in} = 0.5$, $w_{out} = 0.2$, $w_b = 0.3$) omurga duyarlılığı, bağımlılık yüzeyi ve köprü rolü arasında bir denge sağlar; eşitlik durumunda bağıl sıralama korunur.

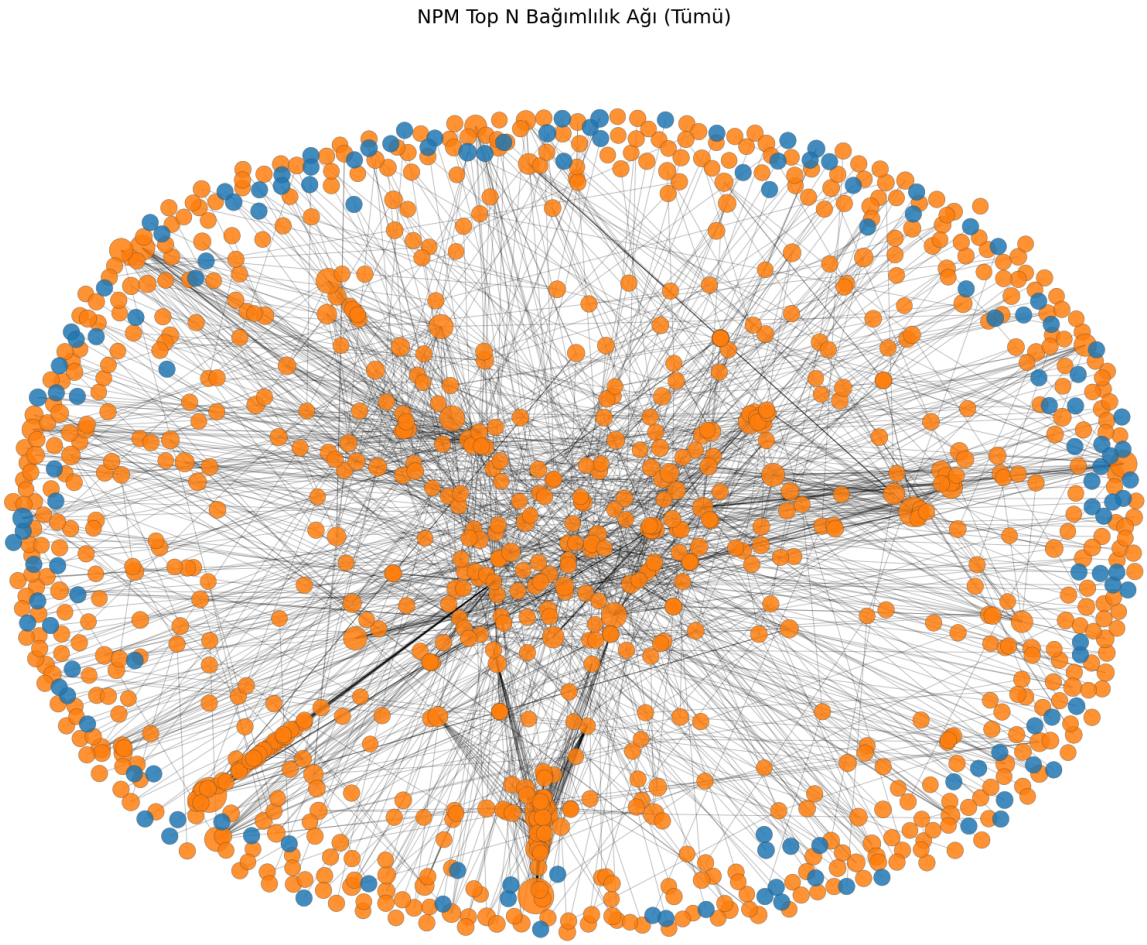
Sağlamlık Deneyi. Risk sıralamasındaki ilk k düğüm grafikten kaldırılır; zayıf bileşen sayısı ve en büyük bileşen boyutu raporlanır (robustness_risk.json). Bu göstergeler, *kritik düğüm kaybına* karşı ağın kırılganlığını nicel olarak özetler.

Üretilen Çıktılar. Görseller ve tablolar results/ altında saklanır: ağ görselleri, derece histogramları ve saçılım grafikleri, merkeziyet liderleri, risk sıralaması, kaskad etkisi ve özet metrik tabloları.

4 Bulgular ve Yorum

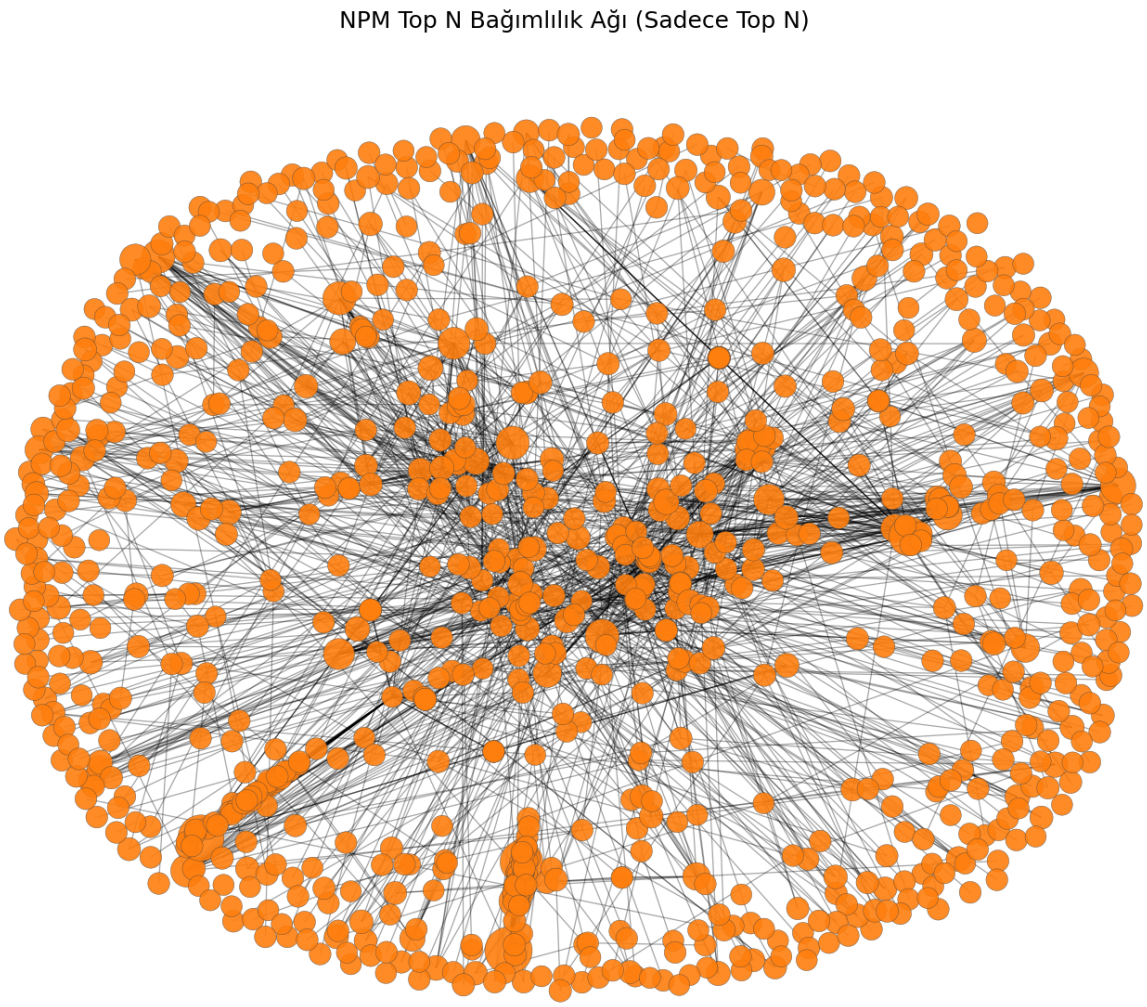
Bu bölümde, results/ dizininden seçilen görseller/tablolar sunulur ve her biri doğrudan altında yorumlanır.

4.1 Ağın Genel Görünümü



Şekil 1: Top N ve bağımlılıklarının oluşturduğu yönlü ağ. Düğüm boyutu in-degree ile, renkler Top N (turuncu) ve diğerleri (mavi) olarak kodlanmıştır. Yüksek in-degree'li düğümler ekosistemin omurgasını oluşturur; bu düğümlerdeki sorunlar geniş yayılıma neden olabilir.

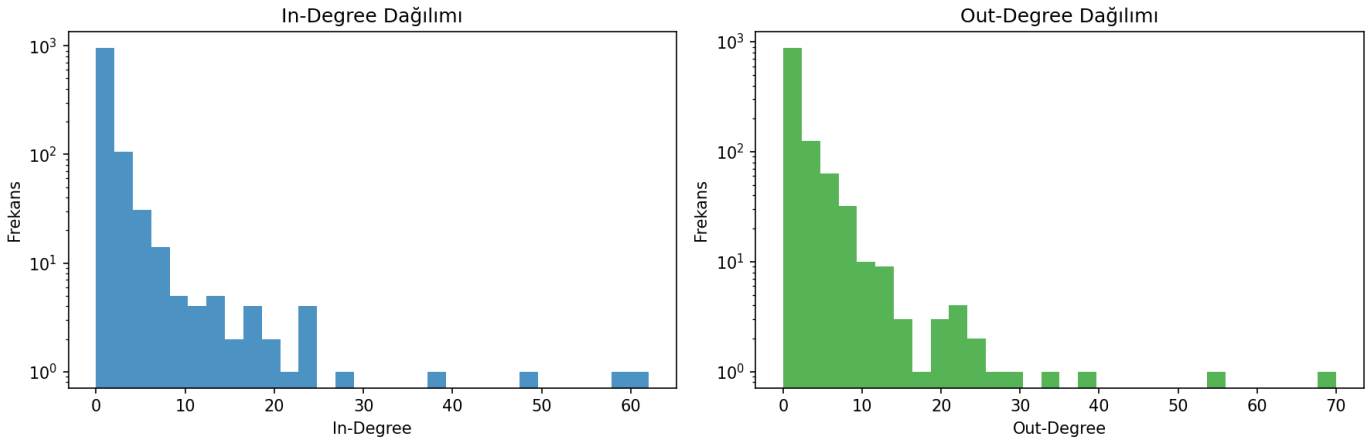
Görsel, merkezî kümelenmeleri ve omurga düğümleri ayırt etmeyi kolaylaştırır. Düğümler arası yoğun bölgeler, paket ekosistemindeki birlikte-kullanım kalıplarına işaret eder.



Şekil 2: Sadece Top N düğümlerin indüklenmiş alt-ağı. Paketin ağ içi konumuna göre merkezî (yüksek in-degree/betweenness) düğümler görselde öne çıkar.

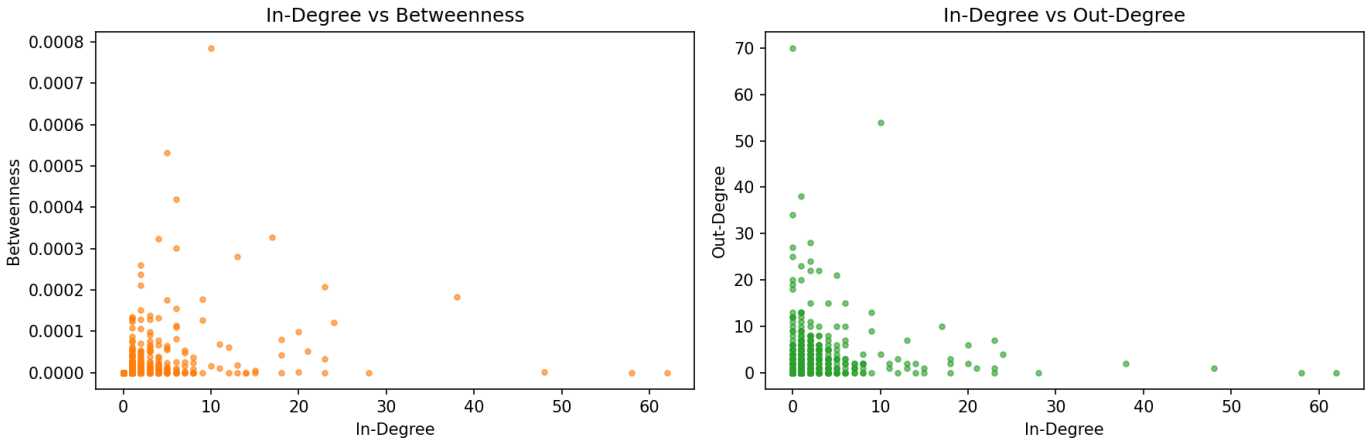
Top N alt-ağında bağlantı motifleri daha belirgindir. Zincirleme etkinin Top N içinde nasıl yayılabileceği niteliksel olarak gözlemlenebilir.

4.2 Derece Dağılımları ve İlişkiler



Şekil 3: In-degree ve out-degree histogramları (log ölçek). Ağın ağır kuyruklu yapısı, az sayıda çok etkili paket ile çok sayıda düşük dereceye sahip paketin birlikte varlığını gösterir.

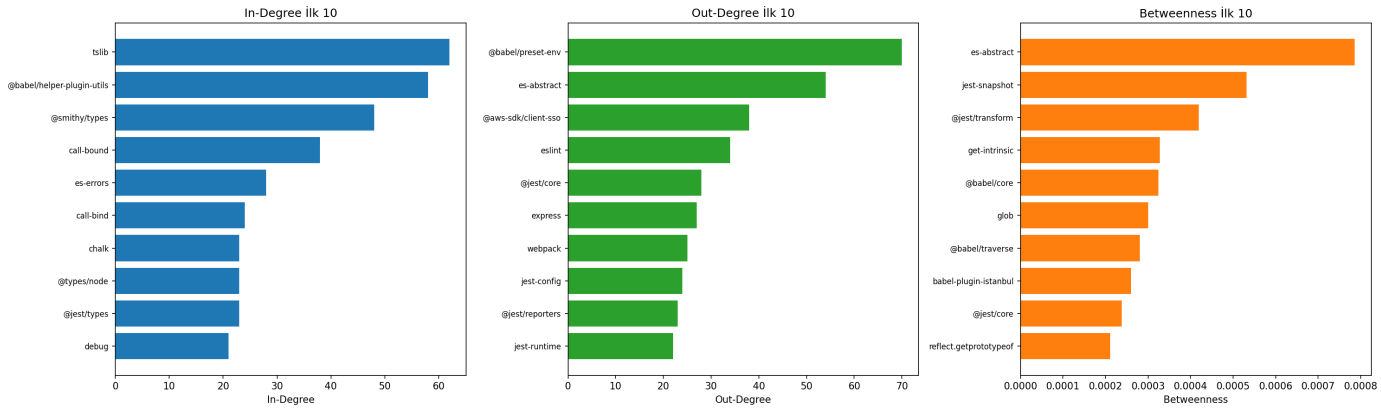
In-degree dağılımı, sistemik etkisi yüksek omurga paketlerini az sayıda düğümde toplar. Out-degree dağılımı, bazı paketlerin geniş bağımlılık yüzeyi nedeniyle tedarik riskine daha açık olduğunu gösterir.



Şekil 4: Korelasyonlar: (sol) In-degree vs Betweenness; (sağ) In-degree vs Out-degree. In-degree ile betweenness arasındaki pozitif ilişki, omurga düğümlerin çoğu zaman köprü rolünde olduğunu gösterir.

In-degree ile out-degree ilişkisi, bir paketin hem çok kullanılıyor hem de çok sayıda bağımlılığa sahip olabildiğini; bu durumda hem yayılım hem maruziyet riskinin birlikte arttığını gösterir.

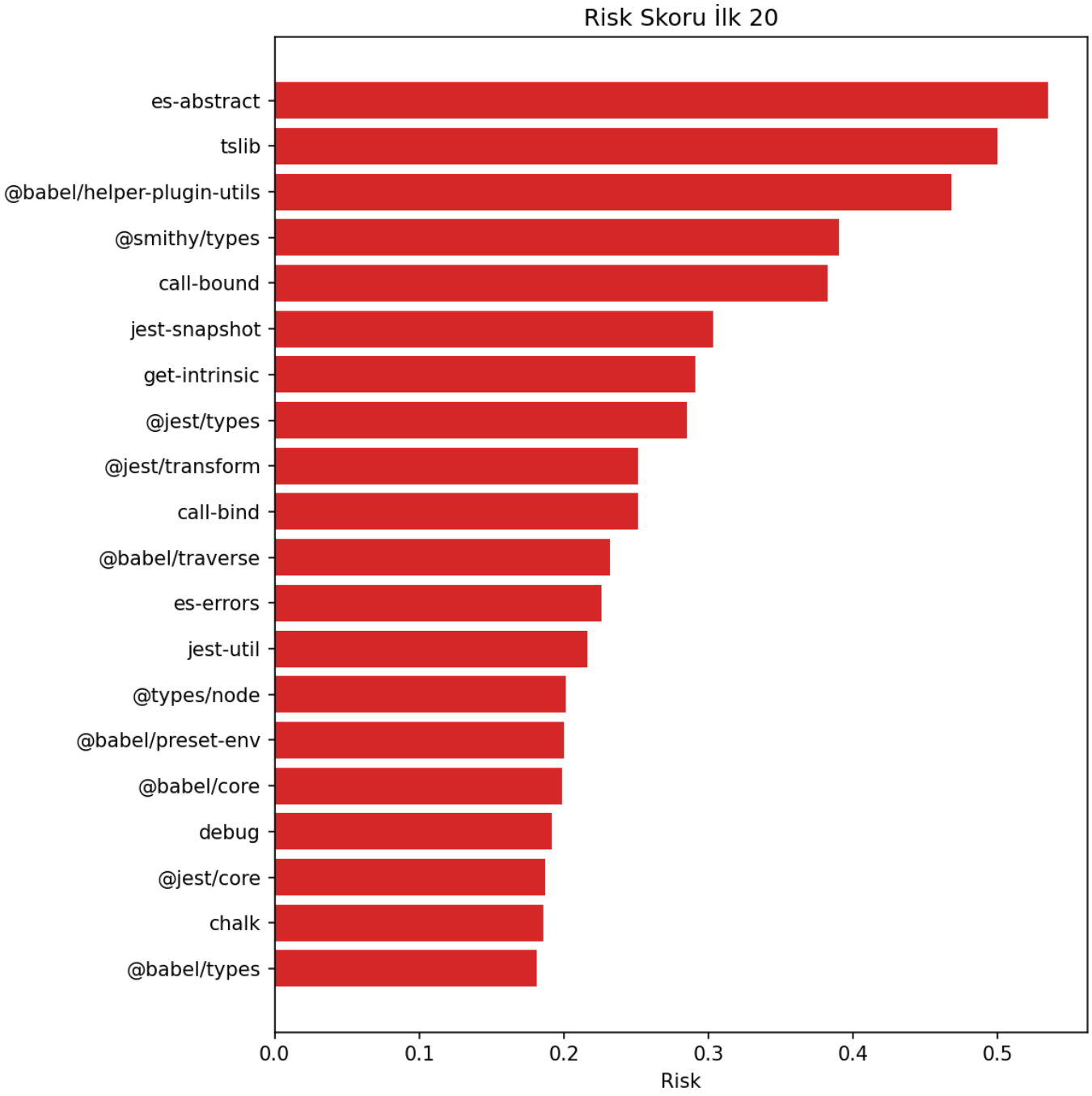
4.3 Merkeziyet Liderleri



Şekil 5: Merkeziyet liderleri (ilk 10): In-degree, Out-degree ve Betweenness karşılaştırması tek görselde. In-degree liderleri yayılım etkisi yüksek paketleri; Out-degree liderleri geniş bağımlılık yüzeyini; yüksek Betweenness ise köprü/boğaz noktası riskini gösterir.

Bir arada sunulan liderlik çizelgeleri, yayılım (in-degree), maruziyet (out-degree) ve köprü (betweenness) etkilerini aynı anda okumayı kolaylaştırır.

4.4 Bileşik Risk Skoru



Şekil 6: Bileşik risk skoru ile en riskli 20 paket. Ağırlıklar in/out/betweenness arasında denge kuracak biçimde seçilmiştir.

Tablo 1: Top 20 Risk Skoru

Paket	Risk	In-Degree	Out-Degree	Betweenness	TopN?
es-abstract	0.534931	10	54	0.000785	True
tslib	0.500000	62	0	0.000000	True
@babel/helper-plugin-utils	0.467742	58	0	0.000000	True
@smithy/types	0.390249	48	1	0.000001	True
call-bound	0.382129	38	2	0.000183	True
jest-snapshot	0.303511	5	21	0.000532	True
get-intrinsic	0.290993	17	10	0.000328	True
@jest/types	0.284735	23	7	0.000207	True
@jest/transform	0.251456	6	15	0.000419	True

Paket	Risk	In-Degree	Out-Degree	Betweenness	TopN?
call-bind	0.251171	24	4	0.000121	True
@babel/traverse	0.231984	13	7	0.000280	True
es-errors	0.225806	28	0	0.000000	True
jest-util	0.216164	20	6	0.000099	True
@types/node	0.201331	23	1	0.000034	True
@babel/preset-env	0.200000	0	70	0.000000	True
@babel/core	0.199075	4	15	0.000324	True
debug	0.191845	21	1	0.000051	True
@jest/core	0.187008	2	28	0.000238	True
chalk	0.185484	23	0	0.000000	True
@babel/types	0.181088	18	2	0.000079	True

Tablo sütunları: Risk (0–1 aralığında normalize), In-Degree (pakete bağımlı paket sayısı), Out-Degree (paketin bağımlılık sayısı), Betweenness (en kısa yollardaki köprü rolü), TopN? (Top N listesinde yer alıp almadığı). Liste, derleme/test altyapısında yer alan ve geniş yayılıma sahip kütüphanelerin (ör. Babel/Jest/TypeScript ekosistemleri) *sistemik risk* potansiyeline işaret eder.

1. `es-abstract`: ECMAScript (JavaScript) spesifikasyonundaki soyut işlemleri (abstract operations) uygulayan bir kütüphanedir. Örneğin `es-abstract/2020/Call` gibi yollarla ES sürümlerine özgü çağrılar yapılabilir. Altyapı katmanında geniş kullanımda olduğu için, bir olay durumunda zincirleme etki doğurabilir.

2. `tslib`: TypeScript derleyicisinin yardımcı işlevlerini sağlayan çalışma zamanı kitaplığıdır. Derlemede ortak yardımcıların tek paketten içe aktarılmasını sağlar; yaygın kullanımı nedeniyle tedarik zinciri açısından kritik bir düğümdür.

3. `@babel/helper-plugin-utils`: `@babel/core` eklentileri için genel yardımcı işlevler sunar. Birçok eklenti ve preset bu pakete dayanır; ele geçirilmesi, geniş bir eklenti ekosistemini etkileyebilir.

4. `@smithy/types`: AWS JavaScript SDK (v3) içinde tip altyapısı sağlayan iç modüllerden biridir. Büyük ölçekli SDK kullanımlarında transitif etkisi yüksek olabilir.

5. `call-bound`: JavaScript'in yerleşik (intrinsic) fonksiyonlarını güvenli şekilde çağırma/bağlama amacıyla kullanılan yardımcı bir pakettir. Küçük görünmesine karşın çok sayıda modül tarafından dolaylı olarak kullanılır.

6. `jest-snapshot`: Jest için snapshot testlerini sağlar; çıktının gelecekteki çalışmalarda karşılaştırılmasını mümkün kılar. Test altyapısında yer alması, geniş projelerde dolaylı etkileri artırır.

7. `get-intrinsic`: ECMAScript "intrinsics" nesnelerini güvenli biçimde elde etmeye yarayan bir yardımcı kütüphanedir. Alt katmanda sık kullanıldığı için yapısal önem taşır.

8. `@jest/types`: Jest ekosistemi için tip tanımlamaları ve ortak tür altyapısı sunar. Altyapı paketleri, modüller arası etkileşimi artırdığı için ağda kritik konumda olabilir.

9. `@jest/transform`: Jest içinde kaynak dönüştürme (transpile vb.) katmanını yönetir. Test–üretim kodu dönüşüm zinciri açısından önemlidir.

10. `call-bind`: JavaScript fonksiyonlarının bağlamla çağrılması (bind/call) için yardımcı işlevler sağlar. Küçük ama çok yaygın bir yardımcı pakettir.

11. `@babel/traverse`: AST üzerinde gezinmeyi sağlayan Babel modülüdür. Kod analizi ve dönüşümünde temel rol oynar.

12. `es-errors`: ECMAScript hata tipleri ve ilgili soyut işlemler için yardımcı işlevler içerir; altyapı rolünde geniş kullanım görür.

13. `jest-util`: Jest ekosisteminde çeşitli yardımcı işlevler sunar; test altyapısının güvenilir çalışmasına katkıda bulunur.

14. `@types/node`: Node.js için TypeScript tip deklarasyonlarıdır. Ekosistemde standartlaşmış geniş kullanım alanı nedeniyle, çok sayıda projede transitif olarak bulunur.

15. `@babel/preset-env`: Hedef ortama göre derleme (transpile) yapılmasını sağlayan Babel preset'idir. Çok sayıda proje tarafından kullanıldığı için yayılım etkisi yüksektir.

16. `@babel/core`: Babel'in çekirdek modülüdür; AST üretimi, plugin/preset yükleme ve çıktı üretimi gibi temel işlevleri sağlar. Derleme zincirinin en üst katmanında konumlanır.

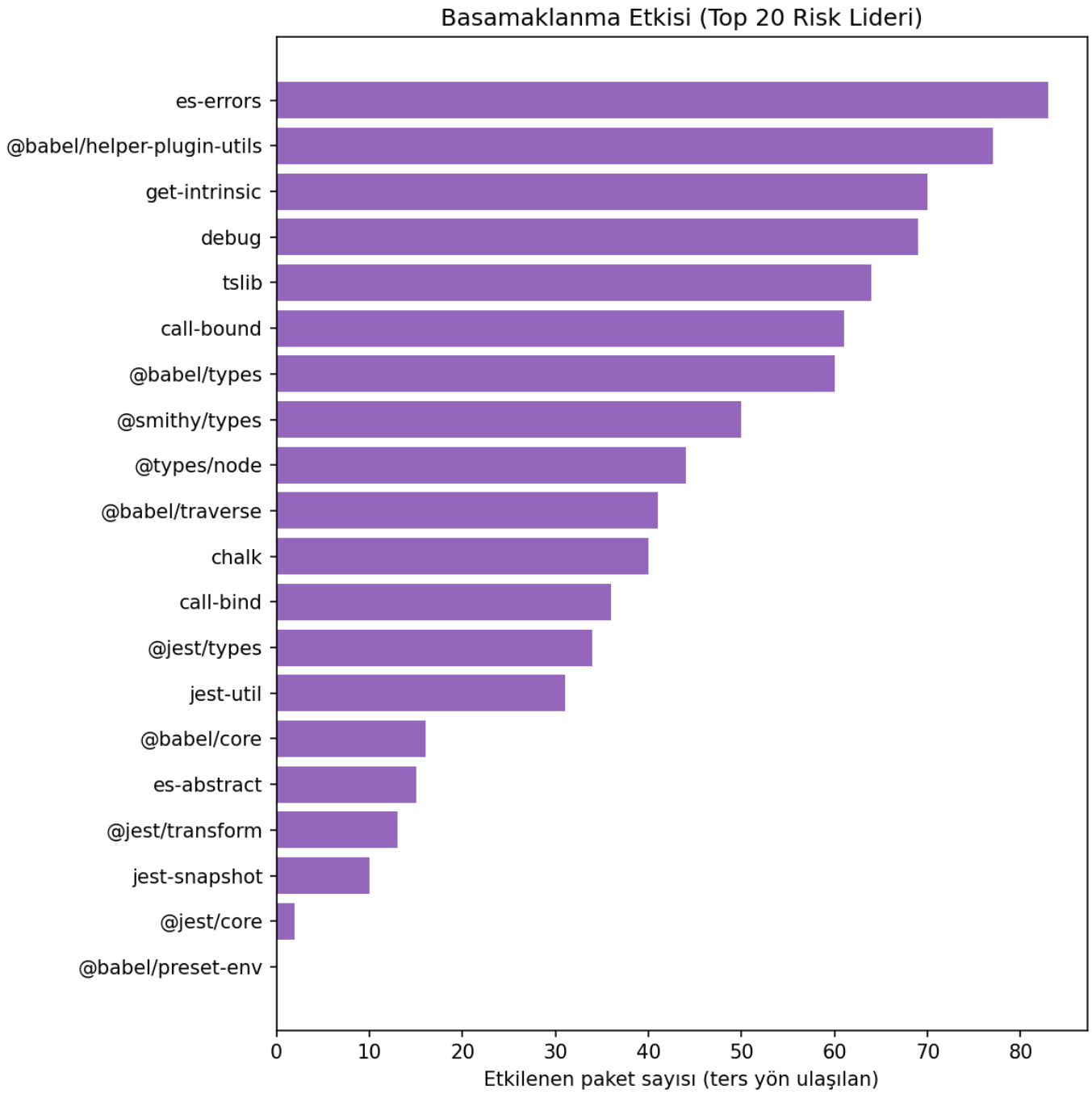
17. `debug`: Ad alanı (namespace) tabanlı loglama sunan çok popüler bir kütüphanedir. Popüler küçük modüllerin tedarik zinciri riski açısından hedef olabileceğini gösteren örnekler mevcuttur.

18. `chalk`: Terminal çıktılarını renklendirme ve stil verme için kullanılır. Yüksek popülerliği nedeniyle tedarik zinciri hedefi olabilir.

19. `@jest/core`: Jest'in çekirdek test çalıştırıcısı ve yönetim modülüdür. Geniş ölçekli projelerde test sürecinin merkezinde yer alır.

20. `@babel/types`: AST düğümlerinin oluşturulması ve denetlenmesi için araçlar sağlar; kod dönüşümünde temel katmandır.

4.5 Kaskad Etkisi ve Sağlamlık

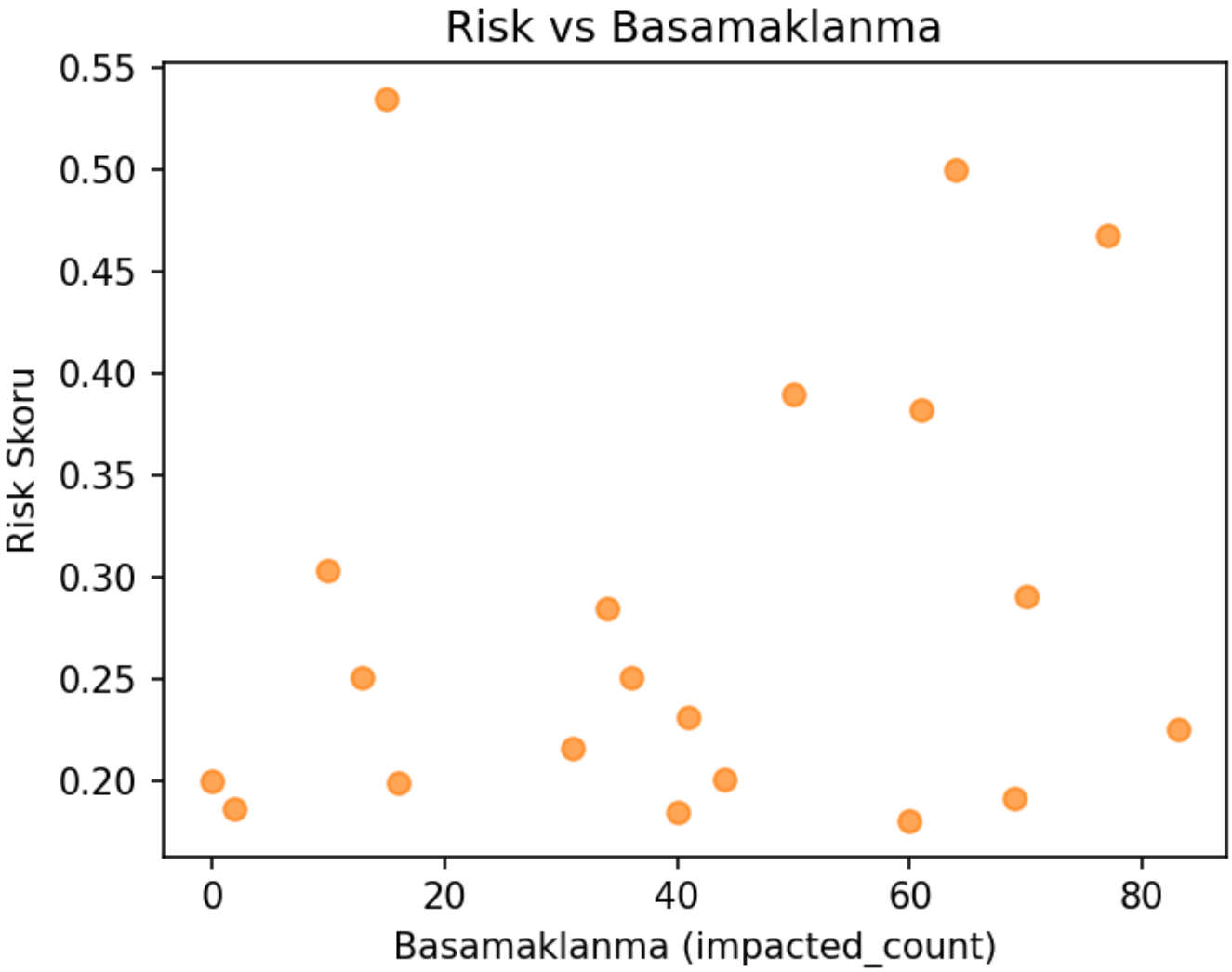


Şekil 7: Risk skoruna göre en riskli 20 paket için kaskad etki büyüklüğü. Yüksek risk genellikle daha büyük kaskada karşılık gelse de, topolojiye bağlı istisnalar görülebilir.

Tablo 2: Basamaklanma Etkisi: Top 20 (Ters yonde etkilenebilecek paket sayisi)

Paket	Etkilenen Paket Sayisi
es-errors	83
@babel/helper-plugin-utils	77
get-intrinsic	70
debug	69
tslib	64
call-bound	61
@babel/types	60
@smithy/types	50

Paket	Etkilenen Paket Sayisi
@types/node	44
@babel/traverse	41
chalk	40
call-bind	36
@jest/types	34
jest-util	31
@babel/core	16
es-abstract	15
@jest/transform	13
jest-snapshot	10
@jest/core	2
@babel/preset-env	0



Şekil 8: Risk skoru ile kaskad etkisi ilişkisi (saçılım grafiği). Doğrusal olmayan yapı, tek başına derece metriklerinin tüm *yayılım dinamiğini* açıklamadığını gösterir.

Kaskad etkisi, ağın bağlantı yapısına duyarlıdır: Aynı risk puanına sahip iki düğüm, komşuluklarının farklılığı nedeniyle farklı kaskad profilleri üretebilir.

4.6 Köprü Kenarlar

En yüksek *edge betweenness* değerine sahip kenarlar, alt-ağlar arasında *kırılgan* bağlantıları işaret eder; bu kenarlar üzerindeki trafik, topolojik ayrışma noktalarını anlamaya yardımcı olur.

Tablo 3: Edge Betweenness Ilk 10 (Yuksek kopru kenarlar)

U	V	Edge Betweenness
@jest/transform	babel-plugin-istanbul	0.000222
@jest/expect	jest-snapshot	0.000212
jest-snapshot	@jest/transform	0.000206
jest	@jest/core	0.000150
call-bound	get-intrinsic	0.000150
glob	jackspeak	0.000147
reflect.getprototypeof	which-builtin-type	0.000146
jackspeak	@isaacs/cliui	0.000140
babel-plugin-istanbul	test-exclude	0.000139
@babel/core	@babel/helper-compilation-targets	0.000138

4.7 Ağın Temel İstatistikleri

Tablo 4: Graf İstatistikleri (özet)

Ölçüt	Değer
Düğüm sayısı	1139
Kenar sayısı	2164
Bileşen sayısı (zayıf)	160
En büyük bileşen boyutu	853
Ortalama in-degree	1.8999
Ortalama out-degree	1.8999

Bu özet tabloda düğüm/kenar sayıları, zayıf bağlanırlık bileşen sayısı, en büyük bileşen boyutu ve ortalama dereceler raporlanır. *Zayıf* (weak) bileşen, kenar yönleri yok sayıldığında bağlı olan düğüm kümeleridir; *güçlü* (strong) bileşen, her düğüm çiftinin birbirine yönlü yollarla erişebildiği alt-kümedir (bu raporda zayıf bileşenler kullanılmıştır).

4.8 Ana Bulgular

- Ağ ağır kuyrukludur: Az sayıda yüksek etkili *omurga* paket ve çok sayıda düşük dereceli paket birlikte bulunur.
- In-degree ve betweenness çoğu zaman birlikte yükselir: Omurga paketler aynı zamanda *köprü* rolü oynar.
- Yüksek out-degree, geniş bağımlılık yüzeyi ve artan maruziyeti işaret eder; bu paketler tedarik riski açısından duyarlıdır.
- Bileşik risk skoru, tekil metriklerin sınırlamalarını dengeleyerek *karşılaştırılabilir* bir sıralama sağlar.
- Kritik düğümlerin çıkarılması, bileşenleşmeyi hızlandırır ve en büyük bileşenin küçülmesine yol açar: Ağ, kritik düğüm kaybına duyarlıdır.

5 Sınırlamalar

Veri. Varsayılan olarak yalnızca dependencies alanı kullanılmıştır; peerDependencies isteğe bağlıdır. Global dependent sayıları doğrudan dahil edilmemiştir. API kaynaklı eksiklikler ve isimlendirme farklılıkları ölçümleri etkileyebilir.

Modelleme. Betweenness büyük graflarda örnekleme ile hesaplanmıştır; yakınsama, ağ büyüklüğü ve k seçimine bağlıdır. Min–max normalizasyonu gözlenen aralığa bağımlıdır; sıra korunsa da mutlak değerlerin yorumu veri setine özeldir.

Sunum. Ağ görselleri büyük dosyalardır; çizim sıralaması ve düzen parametreleri, yerleşim sezgisellerine (layout) bağlı görsel yanlılıklar taşıyabilir.

6 Sonuç

NPM ekosistemi, az sayıda *omurga* ve *köprü* paket etrafında yoğunlaşan bir topoloji sergiler. Bileşik risk skoru ve sağlamlık bulguları, tedarik zinciri güvenliğinde *yapısal* bakışın analitik değerini açıkça ortaya koyar: Kritik paketlerin izlenmesi ve korunması, geniş ölçekli kaskad etkilerini sınırlamada belirleyicidir.

Tekrarlanabilirlik. Tüm kod ve çıktı üretimi depo içindedir. `analysis.ipynb` çalıştırılarak results/ çıktıları yeniden üretilebilir ve bu makale derlenebilir.