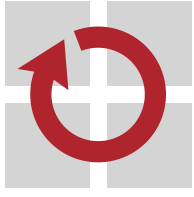
Web-basierte Systeme

11: Caching

Wintersemester 2024

Rüdiger Kapitza



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

Friedrich-Alexander-Universität
Technische Fakultät

Vorlesungsplan

18. Oktober	Einführung und Darstellung von Webseiten (HTML und CSS)
8. November	Hypertext Transfer Protocol
15. November	Browser Schnittstellen
22. November	Kommunikationsschnittstellen im Browser
6. Dezember	WebAssembly
20. Dezember	**Architektur moderner Browser** und Vorbereitung Papieranalyse
10. Januar	Clientseitige Architekturmuster und serverseitige Implementierung von Web-basierten Systemen
17. Januar	Papieranalyse
24. Januar	Caching bzw. Lastverteilung durch Zwischenspeicher
31. Januar	Aspekte von Web Sicherheit
7. Februar	Zusammenfassung und Ausblick

Caching

Caching

Zielsetzung der Lerneinheit

- Verständnis warum die Zwischenlagerung von Daten in webbasierten Systemen sinnvoll und nötig ist.
- Kennenlernen der verschiedenen Möglichkeiten der Zwischenlagerung auf Client- und Serverseite.
- Basiskenntnis von Ansätzen und Strategien zur Lagerung

Caching

Vorüberlegungen

- Daten einer Webanwendung sollen zwar durch einen Dienst bereitgestellt und durch Clients aktualisiert werden können – aber vor allem auch
  - eine möglichst kurze Zugriffszeit besitzen
  - und skalierbar verfügbar sein.

Caching im Browser – Vorüberlegungen

Vorüberlegungen

- Antwortzeiten und die menschliche Wahrnehmung

Delay	User Reaction
0-100 ms	Instant
100-300 ms	Small perceptible delay
300-1000 ms	Machine is working
1000+ ms	Likely mental context switch
10,000+ ms	Task is abandoned

Caching im Browser – Vorüberlegungen

Vorüberlegungen

- Antwortzeiten und die menschliche Wahrnehmung

Delay	User Reaction
0-100 ms	Instant
100-300 ms	Small perceptible delay
300-1000 ms	Machine is working
1000+ ms	Likely mental context switch
10,000+ ms	Task is abandoned

- Minimale Antwortzeiten unter Einbezug der Datenübertragung

Route	Distance	Time, light in vacuum	Time, light in fiber
NYC to SF	4,148 km	14 ms	21 ms
NYC to London	5,585 km	19 ms	28 ms
NYC to Sydney	15,993 km	53 ms	80 ms
Equatorial circumference	40,075 km	133.7 ms	200 ms

- Natürlich ist die Latenz in der Regel höher auf Grund der Vermittlung und anderen Faktoren.

Caching

Warum sollte man einen Cache verwenden?

- Ein Cache ist eine Komponente, die Daten transparent speichert, so dass wiederholte Anfragen nach den gleichen Daten schneller beantwortet werden können.

Caching

Warum sollte man einen Cache verwenden?

- Ein Cache ist eine Komponente, die Daten transparent speichert, so dass wiederholte Anfragen nach den gleichen Daten schneller beantwortet werden können.

Wo können im Prinzip Daten zwischengelagert werden?

- Beim Endnutzer im Browser
- Vor den Diensten im ’Netzwerkpfad’
  - Verwendung eines Content Delivery Networks (CDN)
  - Verwendung eines Proxies (abnehmend in der Bedeutung))
- Key/Value Dienste
- Im Anwendungsserver
- In der Datenbank (Anfrage-Cache)

Wie passt das mit der Edge Cloud zusammen?

Caching im Browser

Wann und warum *cached* der Browser Daten?

- Es gibt eine Reihe von HTTP-Header, die festlegen, wie und ob eine Zwischenspeicherung erfolgt.
- Zwischenlagerung kann vom Server zum Client aber auch andersherum festgelegt werden.
- Wir konzentrieren uns auf die Direktiven des Servers.

Caching im Browser

Wann und warum *cached* der Browser Daten?

- Es gibt eine Reihe von HTTP-Header, die festlegen, wie und ob eine Zwischenspeicherung erfolgt.
- Zwischenlagerung kann vom Server zum Client aber auch andersherum festgelegt werden.
- Wir konzentrieren uns auf die Direktiven des Servers.

HTTP Header (Auswahl)

- Cachebarkeit: `cache-control`
  - Policy: `no-store, no-cache, max-age, public | private`
- `etag`
- `last-modified`
- `if-none-match`
- `if-modified-since`

Caching im Browser

`cache-control: no-store`

- Indiziert dem Browser und evtl. vermittelnden Proxies
  - die Daten nicht persistent zu speichern
  - und jegliche Kopien aus ihrem Speicher unmittelbar zu löschen.
- *Anmerkung:* Browser halten evtl. dennoch eine Kopie im Arbeitsspeicher vor um eine Vorwärts-/Rückwärtsnavigation zu ermöglichen.
- Generell wird dieser Header genutzt um ’sensitive’ Daten zu annotieren.

Caching im Browser

`cache-control: no-cache`

- Indiziert dem Browser und evtl. vermittelnden Proxies
  - die gelagerten Daten sollten auf ihre Aktualität überprüft werden, bevor sie erneut ausgeliefert werden.
- Nützlich um den Browser und Proxies zu zwingen die Aktualität der Daten zu überprüfen.

`cache-control: private`

- Es bleibt dem Browser überlassen die Daten zwischenzulagern – Proxies hingegen sollen die Daten nicht speichern.

`cache-control: public`

- Kennzeichen, dass die Response von jedem Cache zwischengelagert werden darf.

Caching im Browser

`cache-control: max-age=<seconds>`

- Spezifiziert die maximale Zeitdauer, die eine Ressource als aktuell betrachtet wird.
- Im Gegensatz zu *Expires*, ist diese Direktive relativ zum Zeitpunkt der Anfrage.

`cache-control: must-revalidate`

- Der Cache muss den Status der abgelaufenen Ressource überprüfen bevor sie verwendet wird.
- Abgelaufene Ressourcen sollten nicht verwendet werden.

Caching im Browser

Beispiel: Caching komplett verhindern

- `Cache-Control: no-cache, no-store, must-revalidate`

Beispiel: Statische Assets cachen

- Für die Dateien einer Anwendung, die sich nicht ändern werden kann man langfristig zwischenspeichern.
  - Dies schließt statische Dateien ein, die von der Anwendung bereitgestellt werden, wie z.B. Bilder, CSS- und JavaScript-Dateien.
- `Cache-Control: public, max-age=31536000`

Caching im Browser

`ETag: "<etag_value>"`

- Ein ETag stellt einen eindeutigen Identifikator für die Version einer Ressource dar und wird mit der Ressource ausgeliefert.
- Mit jeder neuen Version der Ressource wird ein neuer ETag erzeugt (und im Cache gespeichert).

`if-none-match:"<etag_value>"`

- Wird in einer HTTP-Anfrage `if-none-match:` mitgegeben so kann der Server überprüfen ob die Ressource noch aktuell ist.
- Im Post-Fall wird falls der ETag nicht mit der ETag Version des Server übereinstimmt eine Fehlermeldung geliefert: `412 Precondition Failed`

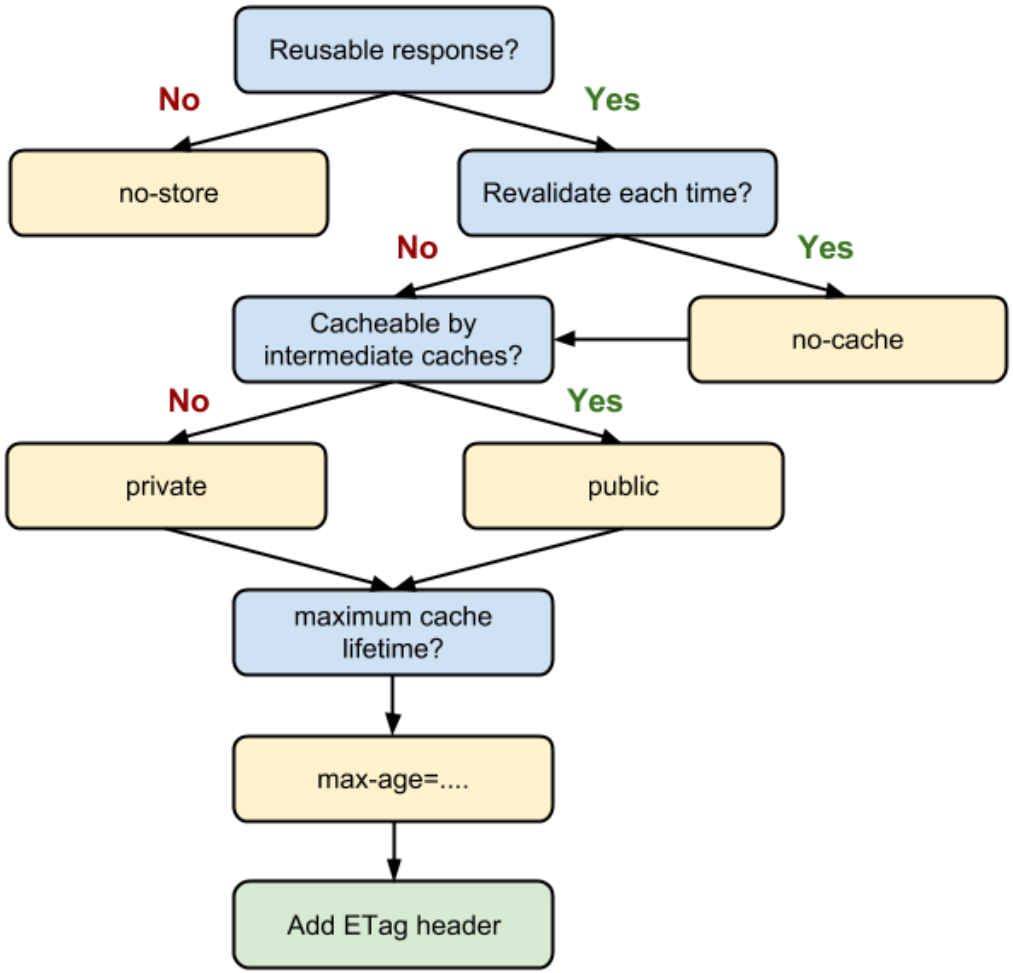
Caching im Browser

`if-none-match:"<etag_value>"`

- Für lokal veraltete Versionen kann auch in einer GET Operation `if-none-match` angegeben werden.
- In diesem Fall wird falls die Ressource sich auf Serverseite nicht verändert hat `304 Not Modified` zurückgeliefert ohne die Ressource erneut auszuliefern.
- Sollte der ETag und damit auch die Ressource sich verändert haben wird sie normal an den Browser übermittelt.

Eine ähnliche Vorgehensweise kann mit `last-modified` und `if-modified-since` durchgeführt werden.

Caching im Browser



Caching auf der Serverseite

Vorüberlegungen

- Webserver beantworten Anfragen einer Vielzahl von verschiedenen Clients
- Jede Antwort erfordert in der Regel Rechenzeit und I/O-Operationen.
- Oft sind Antworten *recht* ähnlich.
- Für das Caching auf Client-Seite sind wir von identischen Antworten ausgegangen – nun schwächen wir das ab und betrachten auch ähnliche Antworten.

Caching auf der Serverseite

Welche HTTP-Antworten bezogen auf einen einzelnen Webserver könnten ähnlich/gleich sein?

- Einzelne View-Elemente der Webseite
  - Zum Beispiel: Kopf- oder Fußelement, Sidebar oder auch Listen die öfters eingeblendet werden.
- Selten modifizierte Datenobjekte
  - Zum Beispiel: Zugriffsrechte, Konfigurationsparameter, Produktdaten – alles was selten aktualisiert wird.
- Aufwendig zu berechnende Daten
  - Dinge die man evtl. sowieso auslagert
  - Beispiel: Diff zwischen zwei Commits bei *GitHub* oder die Kontaktliste von *LinkedIn*.

Caching auf der Serverseite

Wo könnte man in Prinzip Zwischenergebnisse speichern?

1. Direkt im Hauptspeicher des Anwendungs- bzw. Webservers
2. Im lokalen Dateisystem
3. Auf einem anderen System

Caching auf der Serverseite

Numbers Every Programmer Should Know (2018)

- Zahlen die ursprünglich von Jeff Dean erarbeitet wurden in aktualisierter Form:

Ressource	Time
L1 cache reference	1ns
Branch mispredict	3ns
L2 cache reference	4ns
Mutex lock/unlock	17ns
Main memory reference	100ns
Compress 1KB wth Zippy	2,000ns $\approx 2\mu$s
Send 2,000 bytes over commodity network	88ns
SSD random read	16,000ns $\approx 16\mu$s
Read 1,000,000 bytes sequentially from memory	5,000ns $\approx 5\mu$s
Round trip in same datacenter	500,000ns $\approx 500\mu$s
Read 1,000,000 bytes sequentially from SSD	78,000ns $\approx 78\mu$s
Disk seek	3,000,000ns $\approx$ 3ms
Read 1,000,000 bytes sequentially from disk	1,000,000ns $\approx$ 1ms
Packet roundtrip CA to Netherlands	150,000,000ns $\approx$ 150ms

Caching auf der Serverseite

Abbildung dieser Zahlen auf serverseitiges Caching

- Ablegen der Daten im Arbeitsspeicher um später auf sie zurückzugreifen ist schnell!
  - Lesen einer zufälligen Stelle im Arbeitsspeicher dauert < 0.1 μs
- Ablegen von Daten auf einer Festplatte (wenn es sich nicht um eine SSD handelt) ist langsam
  - Positionieren eines Lesekopfes dauert 4000 μs
  - Anschließendes Lesen 1 MB dauert weitere 2000 μs
- SSD zu nutzen macht Sinn
  - Lesen einer zufälligen Speicherstelle dauert 16 μs
  - Lesen von einem 1 MB dauert 156 μs
- Speichern von Daten auf einem Rechner innerhalb des gleichen Datenzentrums ist eine Alternative:
  - Round-trip im Datenzentrum dauert 500 μs

Caching auf der Serverseite

Zusammenfassung

- Im Speicher unter hundert μs
- Auf der SSD speichern hunderte von μs
- Auf einer klassischen Festplatte speichern tausende von von μs
- Auf einer entfernten Maschine: zusätzlich hunderte von μs

Fazit

- Arbeitsspeicher > SSD > entfernter Rechner

Reicht das schon aus als Daumenregel?

Caching auf der Serverseite

Hit rate bzw. Trefferquote in Abhängigkeit des Speicherortes

- Werden die Daten im Arbeitsspeicher abgelegt kann in der Regel nur der lokale Prozess profitieren
- Werden die Daten auf der lokalen SSD gespeichert können alle Prozesse der Maschine zugreifen
- Im Fall einer entfernten Maschine können alle Rechner des Clusters zugreifen.

Fazit

- Es gibt eine Abwägung zwischen Zugriffszeit und Trefferquote!

Caching auf der Serverseite

Beipiel: Memcached[1]

- Wird häufig zum zwischenlagern von Daten verwendet.
- Speichert alle verwalteten Daten im Arbeitsspeicher
- Hat eine einfache über TCP ansprechbare Schnittstelle
- Kann verteilt betrieben werden …
  - Schlüssel können eine Länge von 250 byte haben
  - Werte eine maximale Größe von 1 MB
- Es wird LRU (least recently used) verwendet um bei Speicherknappheit dafür zu sorgen das Platz für neue Daten existiert.
- Alle wichtigen Methoden haben eine konstante Zugriffszeit!

[1] `https://memcached.org`

Content Delivery Networks (CDN)

Motivation

- Einzelner Ort zur Bereitstellung von sehr populären Inhalten hat inhärente Probleme:
  - Skalierbarkeit, Zuverlässigkeit und Performanz
- *Flash crowd!*
  - Spontane Popularität
- Zwischenablage von Ressourcen an den *Rändern* des Netzes
  - Zugriffe auf den primären Server reduzieren
  - Schnellerer Zugriff für Nutzer da näher

Content Delivery Networks (CDN)

Motivation

- Einzelner Ort zur Bereitstellung von sehr populären Inhalten hat inhärente Probleme:
  - Skalierbarkeit, Zuverlässigkeit und Performanz
- *Flash crowd!*
  - Spontane Popularität
- Zwischenablage von Ressourcen an den *Rändern* des Netzes
  - Zugriffe auf den primären Server reduzieren
  - Schnellerer Zugriff für Nutzer da näher

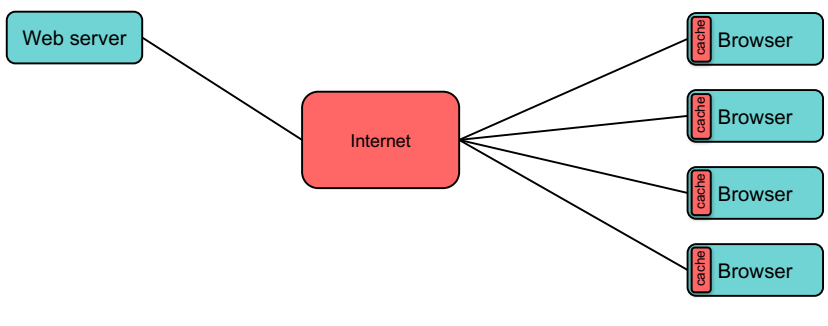
Was sollte ausgelagert werden?

- Vor allem statische Inhalte – da sie den Hauptanteil am Datenverkehr ausmachen
  - z.B. Bilder, Video, CSS und statische Seitenanteile

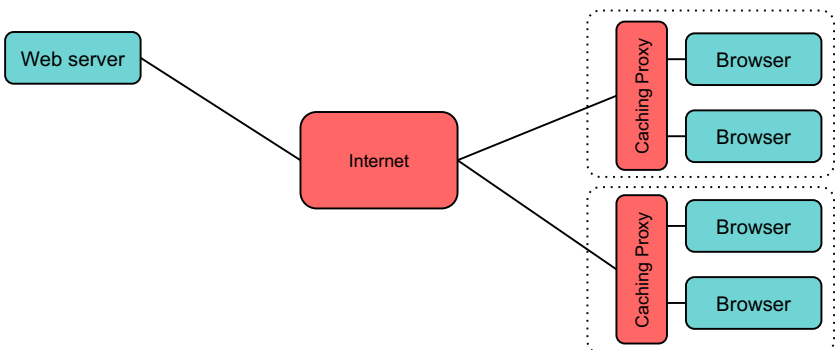
Content Delivery Networks (CDN)

Bisherige Möglichkeiten im Überblick

- Jede *initiale* Anfrage muss vom Server beantwortet

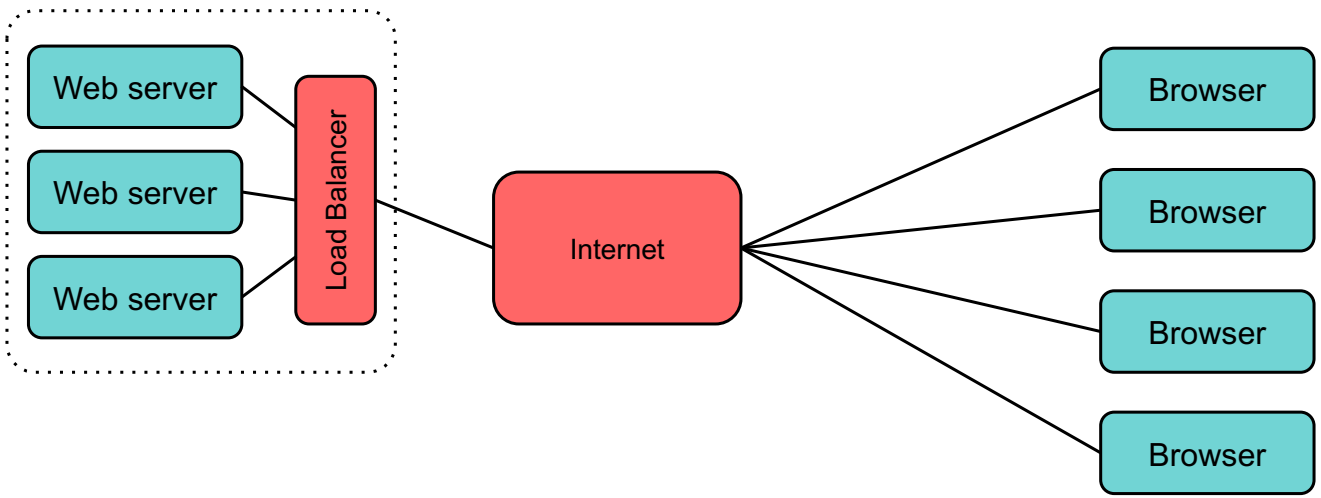


- Caching innerhalb einer Institution



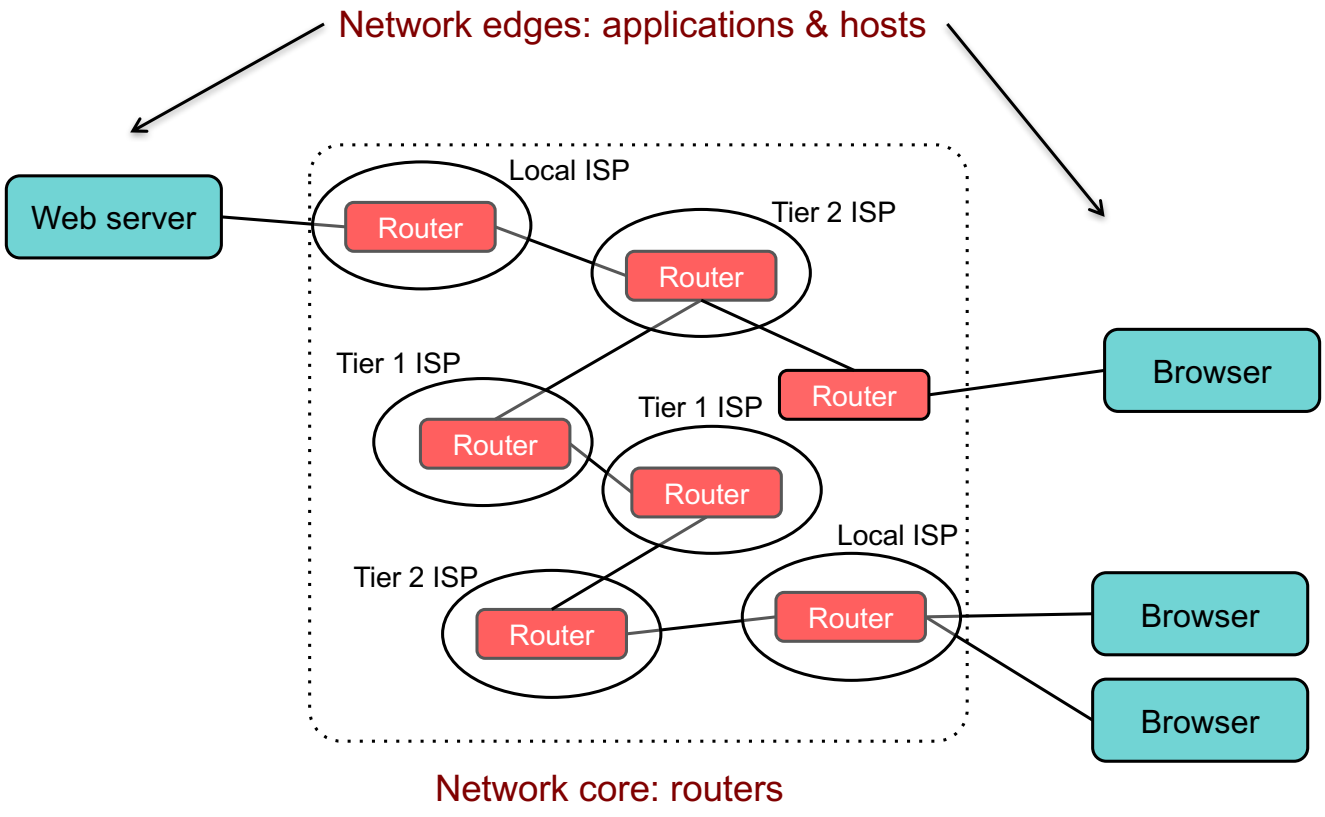
Bisherige Möglichkeiten im Überblick

- Weitere Möglichkeit wäre die Serverseite skalierbarer zu machen unter Verwendung eines *Load Balancer*
- Aber die Anbindung an das Netzwerk kann einen Engpass darstellen
- Latenz zwischen Client und Server ist immer noch hoch



Content Delivery Networks (CDN)

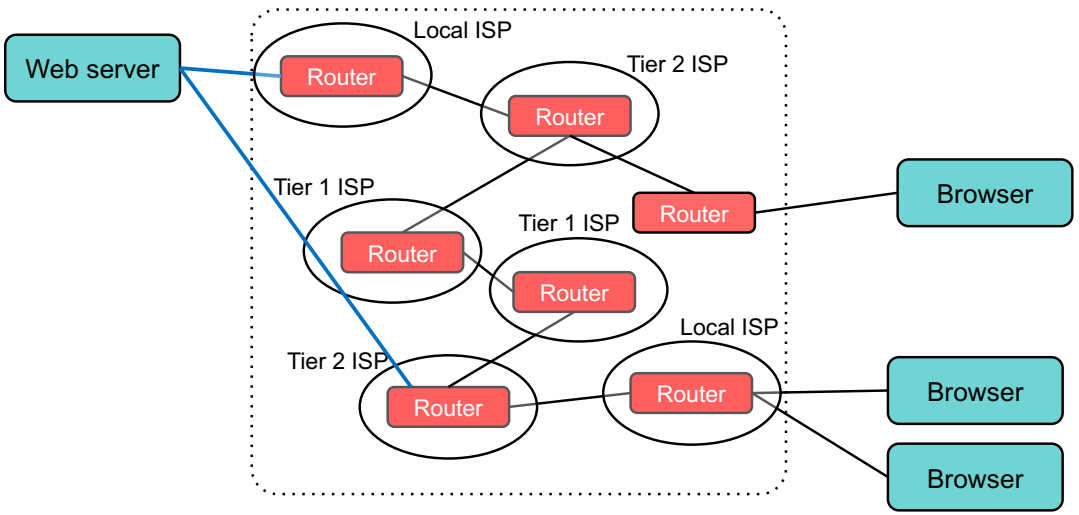
Weitere Möglichkeiten



Content Delivery Networks (CDN)

Multihoming

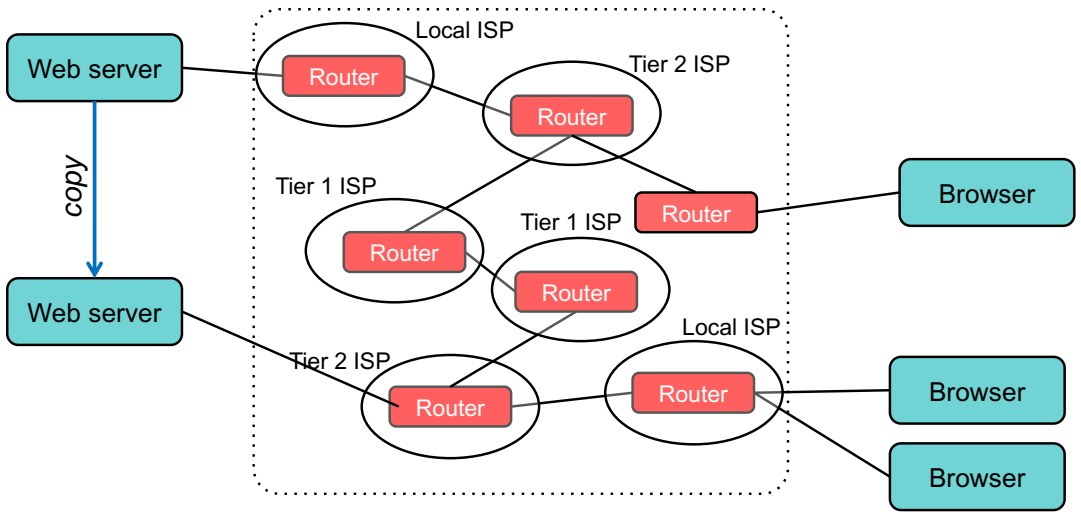
- Etablierung von Netzwerkverbindungen zu verschiedenen ISPs
- Nur eine IP aber mehrere Links über die diese erreicht werden kann
- Adressen werden über Border Gateway Protocol (BGP) vermittelt
- Clients können nun verschiedene Routen nutzen und Ausfall eines ISPs kann toleriert werden



Content Delivery Networks (CDN)

Mirroring bzw. Replikation

- Synchronisation mehrerer Server
- Einbindung von verschiedenen ISPs ermöglicht Load Balancing in Abhängigkeit Clients
- Fehlertoleranz gegenüber Ausfällen von ISPs und Servern



Content Delivery Networks (CDN)

Vorzüge der Ansätze

- Skalierbarkeit
  - Replikation über mehrere Server (Load Balancing)
  - Nutzung verschiedener ISPs, falls das Netzwerk einen Engpass darstellt
- Verfügbarkeit
  - Replikation der Server
  - Nutzung verschiedener Datenzentren und ISPs
- Performanz
  - Caching der Inhalte in der Nähe der Clients

Content Delivery Networks (CDN)

Probleme der bisher diskutierten Ansätze

- Lokales Load Balancing
  - Datenzentren können natürlich ausfallen
- Multihoming
  - Es dauert einige Zeit bis neue Routen gefunden sind
- Mirroring
  - Synchronisation ist nicht immer einfach
- Proxy Server
  - Lösung außerhalb der Kontrolle des Dienstbetreibers mit oft geringem Nutzen (Trefferquote)

Akamai als Beispiel für ein CDN

Akamai

- Entstand basierend auf Forschung am MIT
- Ziel ist es eine Antwort auf das *flash crowd*-Problem zu geben
- Aktuelle Daten der Firma [2]:
  - Mehr als >345,000 Server in >1,300 Netzwerken über >135 Länder verteilt
  - Liefert zwischen 15-30% der Web-Daten aus
  - Akkumulierte Bandbreite beträgt 30 Terabit pro Sekunde (alte Zahl von 2020)
- Ziele: Daten über Server bereitzustellen
  - die bzgl. der Latenz am nächsten sind,
  - nicht überlastet sind
  - und am wahrscheinlichsten die Daten auch vorhalten.

[2] `https://www.akamai.com/uk/en/about/facts-figures.jsp`

Akamai als Beispiel für ein CDN

Das Internet besteht aus vielen autonomen Systemen

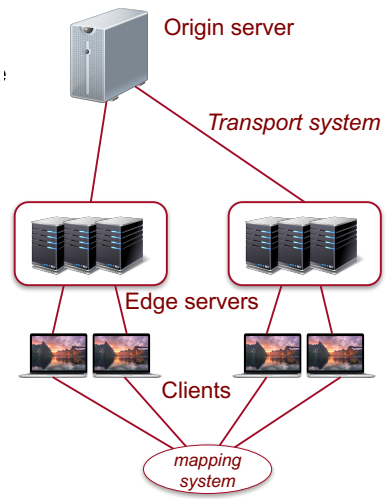
- Der Verbindungen der jeweiligen Netzwerke basiert zumeist auf geschäftlichen Entscheidungen
- ISPs sind vor allem an
  - einer schnellen Verbindung zwischen Nutzer und ihrem Netzwerk (sogenannte *last mile*), sowie
  - einer schnellen Anbindung von Servern in ihrem Netz interessiert.

Basis von Akamai: *Overlay* Netzwerk

- Verband von Caching Servern welche über sehr viele ISPs verteilt sind
- Alle Server sind untereinander verbunden

Akamai als Beispiel für ein CDN

1. Namensauflösung
   - Mittels eines *Mapping Systems* Abbildung auf einen Server
   - Verwendung von eigenen DNS Servern
     - Zielsetzung Weiterleitung auf den nächstgelegenen *Edge* Server
2. Vermittlung der Anfrage vom Browser zum ausgewählten Server
   - Edge Server hat evtl. die benötigten Daten schon lokal vorliegen
   - Falls nicht wird der Ursprungsdienst angefragt



Akamai als Beispiel für ein CDN

Abbildung von Anfragen auf Server mittels dynamischer Zuweisung durch DNS Server

- Ermittlung des zuständigen Servers unter Einbezug:
  - des Ursprungs der Anfrage
  - Verfügbarkeit und Last von Servern
  - Netzwerkbedingungen
- Grundsätzlich ist ein Server im ISP Netzwerk des anfragenden Clients zu bevorzugen

Akamai als Beispiel für ein CDN

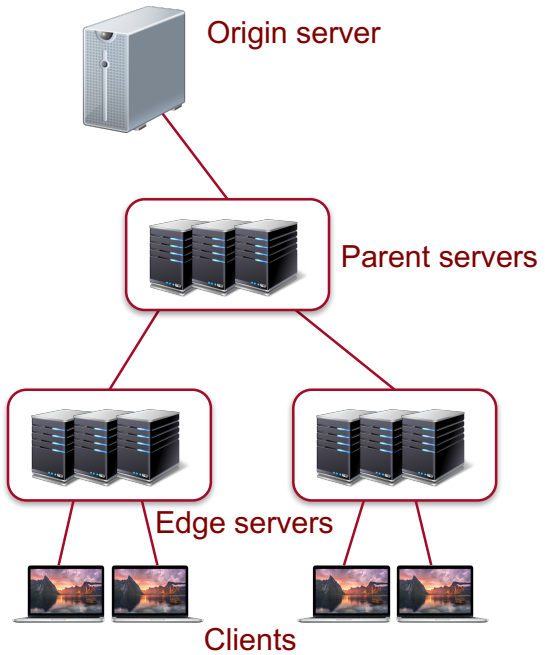
Sammeln von wichtigen Monitoringinformationen

- Abbildung der Netzwerktopologie
  - Nutzung von BGP und *traceroute* Informationen
  - Ermöglicht es Anzahl der Hops und Übertragungszeit abzuschätzen
- Server melden ihre aktuelle Belastung an eine zentrale Monitoringanwendung
- Diese vermittelt die Informationen weiter an Akamai DNS Server
- Informationen werden genutzt um den richtigen Server für eine Anfrage auszuwählen
- Sollten Server zu überlastet sein, werden diese erstmal nicht weiter in Betracht gezogen

Akamai als Beispiel für ein CDN

Vorzüge eines CDNs

- Daten werden bereits lokal vor Ort vorgehalten
- Dynamische Inhalte liegen immer noch beim Ursprungsserver
- Man kann auch noch eine Hierarchie von Caches etablieren um die Trefferquote zu erhöhen...



Akamai als Beispiel für ein CDN

Vorzüge eines CDNs

Welche Arten von Inhalten können/sollten zwischengelagert werden?

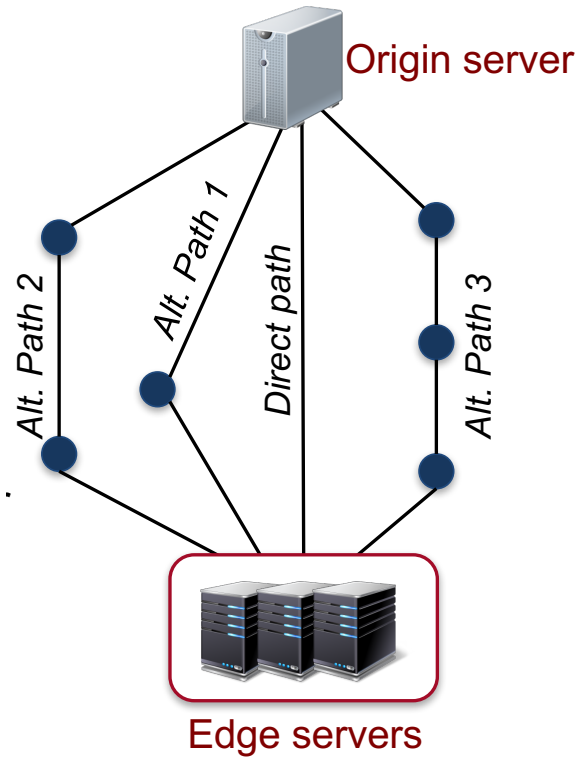
- Statische Inhalte
  - Sehr variable Lagerungszeiten
- Dynamische Inhalte sind möglich
  - Proxies können das nicht tun
  - Akamai setzt auf Edge Side Includes[3]
  - Zusammenfügen von Inhalten kann in Edge Servern vorgenommen werden
  - Konzept ähnlich zu Server Side Includes (SSI)
- Streaming
  - Daten werden intern über das Akamai Netzwerk verteilt und an mehrere Edge Server ausgeliefert

[3] https://www.w3.org/TR/esi-lang/

Akamai als Beispiel für ein CDN

Vorzüge eines CDNs: Routing

- Routen von Edge zu Ursprungsservern werden über eigenes Overlay etabliert
- Entscheidung der Route bezieht eine Vielzahl von Faktoren ein
- Zwischenknoten schicken die Daten weiter



Akamai als Beispiel für ein CDN

Vorzüge eines CDNs: **Sicherheit**

- Hohe Kapazität
  - Widerstandsfähig gegenüber DDoS
- Expertise
- Gehärtete Infrastruktur
  - Spezieller Netzwerkstack
  - Monitoring zum Erkennen von Angriffen
- Schutz der Ursprungsdienste
  - Der Angreifer sollte die IP des eigentlichen Dienstes garnicht kennen

Zusammenfassung

- Rigorose Nutzung von Caching reduziert Last- und Bandbreitenbedarf eines Websites
- Vielzahl von Möglichkeiten mit unterschiedlicher Trefferquote und Auswirkung auf die Anfragelatenz
- Systemeigenschaften bei Auswahl der Verfahren im Blick haben!
- Auch wenn Webseiten evtl. aktuell schwach besucht sind kann sich dies spontan ändern …

Literatur

[1] John Dilley et al. “Globally Distributed Content Delivery”. In: *IEEE Internet Computing* 6.5 (Sep. 2002), S. 50–58. ISSN: 1089-7801.

[2] Ilya Grigorik. *High Performance Browser Networking: What every web developer should know about networking and web performance*. O’Reilly Media, Inc., 2013.