Willkommen bei Kommunikations- und Netztechnik!

–

Von Kupferkabel, Glasfaser und Mikrowelle
über Telefon, Ethernet und TCP
zu E-Mail, Webserver und REST.

–

–

Heute: **DNS und Kampf den RTTs!**

Organisatorisches

- Klausuraufsicht: Ich habe vor, da zu sein.

Inhalt heute

- IPSec
- DNS
- Server –> Client Kommunikation mit HTTP
- Web Dev, HTTP 2/3
- Misc Stuff
- letztes Übungsblatt

IPSec

Wie können wir unkooperative[1] *Programme sichern?*

Ausreichende Sicherheit auf der Vermittlungsschicht.

[1] Zum Beispiel alte, ungewartete, unfreie, selbstzusammengehackte, nicht selbst gewartete. Eigentlich geht es darum, nicht jedes Programm anpacken zu müssen, aber unkooperativ macht das deutlicher.

Struktur

- Sicherheitsassoziation: Schlüsselaustausch (IKE)
- Authentifizierungs-Header (AH)
- Encapsulating Security Payload (ESP): Verschlüsselung *und* Authentifizierung[2]
- Transport- oder Tunnelmodus

[2] Weil sie es konnten.

Sicherheitsassoziation

Internet Key Exchange.

- Preshared-Key: Vorab einkonfiguriert
- IKEv1:
  - Algorithmen aushandeln
  - Diffie-Hellmann → Gemeinsamer Schlüssel
- IKEv2: Komplexer
- Pre-Shared-Keyring (PSK) oder Öffentlicher Schlüssel

⇒ ISAKMP: Internet Security Association and Key Management Protocol → `https://www.kame.net/`

Authentication Header

... IP Header ...		
Next Header	Data-Lengh	Reserviert
Sicherheitsparameterindex		
Sequenznummer		
Authentifizierungsdaten (HMAC)		
... TCP Header ...		

→ Integritätsprüfung, Nutzdaten und **unveränderliche IP Header Daten** signiert.

HMAC

```
HMAC(Message, Key):
  Hash(
      (Key xor opad)
    + (Hash(Key xor ipad) + Message)
  )
```

- ipad, opad: Konstanten

Encapsulating Security Payload (ESP)

Transportmodus

IP-Header	ESP-Header	TCP-Header	Nutzdaten	HMAC
		verschlüsselt		

/Pakete bleiben einzeln erkennbar, geringerer Overhead.

Tunnelmodus

Neu IP	ESP-Header	Alter IP-Header	TCP	Nutzdaten	HMAC
		verschlüsselt			

Tunnel zwischen Gateways, kann wieder ausgepackt werden und TCP Verbindungen bündeln. Verhindert Analyse der Header durch andere. Aber Doppelte Header.
HMAC am Ende: Leichter für Netzgeräte. Ab ESP Header ⇒ Keine der IP Header.

ESP-Header

... IP Header ...
Sicherheitsparameterindex
Sequenznummer
Initialisierungsvektor (Optional)
... TCP Header ...

Wieso AH?

- ESP sollte ursprünglich nur Verschlüsselung machen
- AH prüft einen Tel des IP Headers.
- Kein anderer guter Grund

Zusammenfassung

- Verschlüsselung auf IP-Ebene.
- Header nach IP Header
  - Tunnel: Kapselt und Verschlüsselt in ganz neuem IP Paket
  - Transport: IP-Header bleibt erhalten.
- AH von ESP großteils ersetzt.

Ziele

- Sie kennen Sinn und Funktion von DNS
- Sie kennen DNS records
- Sie kennen DNS Namerserver Klassen
- Sie kennen den Ablauf einer Namensauflösung
- Sie kennen DNSSEC

DNS allgemein

- 216.58.214.36
- IPs sind schlecht zu merken
- Umzug auf anderen Host (neue IP): Nutzer müssen informiert werden
- –> Mechanismus um Namen in IP zu übersetzen, und zu entkoppeln

DNS als Lösung

- DNS implementiert ein hierarchisches Namenssystem
- mittels einer verteilten Datenbank
- verwendet UDP

nslookup

```
nslookup www.google.com
Server:  192.168.0.2

Non-authoritative answer:
Name: www.google.com
Address: 216.58.214.36
Name: www.google.com
Address: 2a00:1450:4001:819::2004
```


Namespace

- Vergleichbar mit Post-Adressen
  - Land, Plz Stadt, Adresse
- top-level Domains von ICANN (Internet Corporation for Assigned Names and Numbers) verwaltet
- 2 Arten von top-level Domains:
  - generic (`com`, `org`)
  - countries (`de`, `fr`)
- second-level Domains werden vom jeweiligen Registrar vergeben
  - Bsp: DENIC für `de`

Hierarchien

Für die verschiedenen Hierarchieebenen sind unterschiedliche Organisationen verantwortlich.

- subdomains werden jeweils vom Inhaber der nächsthöheren Domain vergeben
- Bsp: `dhbw-karlsruhe.de` von DENIC vergeben
  - `else.dhbw-karlsruhe.de` von der DHBW vergeben

Resource Records

- sind ein 5er Tupel aus:
- `Domain_name`: für welche Domain gilt der Record?
- `Time_to_live`: wie lange darf ein Record gecached werden (in Sekunden)?
- `Class`: `IN` für Internet, andere Werte sind selten
- Type: `A` (Address), `AAAA` (IPv6), `MX` (Mail), `NS` (Nameserver), `CNAME` (alias)
- `Value`: abhängig von Type

```
mail.google.com.         1732    IN      CNAME   googlemail
googlemail.l.google.com. 181     IN      A       108.177.12(
```


Root Nameserver

- 13 Root Server
- zuständig für Auflösung von Top Level Domains (TLDs)
- betrieben von ICANN

dig the toplevel NS

```
dig NS .

;; ANSWER SECTION:
.   9342 IN NS i.root-servers.net.
.   9342 IN NS j.root-servers.net.
.   9342 IN NS k.root-servers.net.
.   9342 IN NS l.root-servers.net.
.   9342 IN NS m.root-servers.net.
.   9342 IN NS a.root-servers.net.
.   9342 IN NS b.root-servers.net.
.   9342 IN NS c.root-servers.net.
.   9342 IN NS d.root-servers.net.
.   9342 IN NS e.root-servers.net.
.   9342 IN NS f.root-servers.net.
.   9342 IN NS g.root-servers.net.
```


TLD Nameserver

- zuständig für TLDs (org, de)
- betrieben von z.B.: DENIC

dig the de NS

```
dig NS de

;; ANSWER SECTION:
de.   4032 IN NS n.de.net.
de.   4032 IN NS s.de.net.
de.   4032 IN NS z.nic.de.
de.   4032 IN NS a.nic.de.
de.   4032 IN NS f.nic.de.
de.   4032 IN NS l.de.net.
```


Authoritative Nameserver

- sind offiziell für eine Zone zuständig
- werden bei Registrar angegeben

dig the DHBW

```
dig NS dhbw-karlsruhe.de

;; ANSWER SECTION:
dhbw-karlsruhe.de. 3600 IN NS dns3.belwue.de.
dhbw-karlsruhe.de. 3600 IN NS dns1.belwue.de.
```


Non-authoritative Nameserver

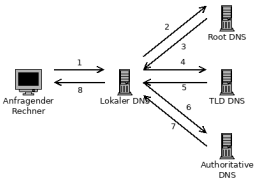
- entlasten die authoritative Nameserver
- werden z.B.: von ISPs betrieben
- beziehen ihre Daten von authoritative Nameservern
- und Cachen die Daten (time to live)

DNSSEC

- Authoritativer Server hat public key
- Schickt DNS-Record + Signatur
- DS-Record (Delegation Signer) beim Domain Registrar enthält hash (digest) des public key
- Aber: Plaintext ⇒ DNS-over-HTTPS.
- Details: *DNSSEC What it is and what it isn't* https://www.youtube.com/watch?v=WrHrtXvO1qM Spaß . . .
- DNS-over-TLS und DNS-over-HTTPS: https://blog.circuitsofimagination.com/2018/11/08/dns-o-t-dnssec-dns-o-h.html

Anatomie einer Namensauflösung

1. dhbw-karlsruhe.de
2. dhbw-karlsruhe.de
3. a.nic.de
4. dhbw-karlsruhe.de
5. dns1.belwue.de
6. dhbw-karlsruhe.de
7. 185.30.157.2
8. 185.30.157.2



dig to trace I

```
dig +trace dhbw-karlsruhe.de
.                       8771    IN      NS      d.root-servers.net.
.                       8771    IN      NS      e.root-servers.net.
.                       8771    IN      NS      f.root-servers.net.
.                       8771    IN      NS      g.root-servers.net.
.                       8771    IN      NS      h.root-servers.net.
.                       8771    IN      NS      i.root-servers.net.
.                       8771    IN      NS      j.root-servers.net.
.                       8771    IN      NS      k.root-servers.net.
.                       8771    IN      NS      l.root-servers.net.
.                       8771    IN      NS      m.root-servers.net.
.                       8771    IN      NS      a.root-servers.net.
.                       8771    IN      NS      b.root-servers.net.
.                       8771    IN      NS      c.root-servers.net.
;; Received 717 bytes from 192.168.0.2#53(192.168.0.2) in 11 ms

de.                     172800  IN      NS      l.de.net.
de.                     172800  IN      NS      f.nic.de.
```


dig to trace II

```
de.                     172800  IN      NS      a.nic.de.
de.                     172800  IN      NS      z.nic.de.
de.                     172800  IN      NS      s.de.net.
de.                     172800  IN      NS      n.de.net.
;; Received 751 bytes from 199.9.14.201#53(b.root-servers.net) in 168 ms

dhbw-karlsruhe.de.      86400   IN      NS      dns1.belwue.de.
dhbw-karlsruhe.de.      86400   IN      NS      dns3.belwue.de.
;; Received 698 bytes from 194.0.0.53#53(a.nic.de) in 24 ms

dhbw-karlsruhe.de.      3600    IN      A       185.30.157.2
dhbw-karlsruhe.de.      3600    IN      NS      dns3.belwue.de.
dhbw-karlsruhe.de.      3600    IN      NS      dns1.belwue.de.
;; Received 135 bytes from 131.246.119.18#53(dns3.belwue.de) in 21 ms
```


Zusammenfassung

- DNS übersetzt Domains in Adressen
- DNS ist hierarchisch gegliedert (`else.dhbw-karlsruhe.de.`)
- es existieren verschiedene DNS records (NS, A, CNAME)
- unterschiedliche DNS Server für unterschiedliche Hierarchiestufen zuständig
- non-authoritative Server entlasten authoritative Server
- Namensauflösung verwendet mehrere Server

Server –> Client

- Die Rückrichtung effizienter machen.

Lernziele

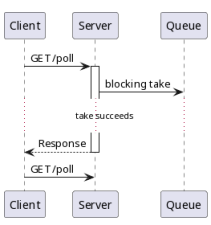
- kennen der 3 Verfahren Long Polling, SSE, Websockets (WS)
- je 1 Unterschied nennen können

Das Problem

- HTTP ist ein Request-Response Protokoll
- initiiert vom Client
- Client möchte etwas vom Server wissen
- Server antwortet

Problem: Server kann keine Kommunikation initiieren (Bsp: Chat vs. Forum)

Long Polling



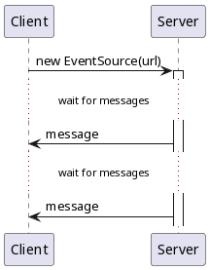
Long Polling

- Client baut Verbindung zu Server auf
- Server antwortet nicht sofort, sondern blockt
- sobald Server unblocked wird, antwortet er Client
- Client baut erneut Verbindung auf
- falls Verbindung geschlossen wird (timeout), baut Client neue Verbindung auf

Long Polling Fazit

- hacky
- erfordert keine Browserunterstützung
- besser als normales polling (Server antwortet sofort)

Server Sent Events



Server Sent Events

- Javascript-API
- Client baut Verbindung zu Server auf
- Server blockt bis Message verfügbar
- Server sendet Message an Client
- Verbindung bleibt offen, Server blockt wieder bis Message verfügbar

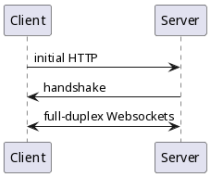
Beispiel Client Implementierung

```
const eventSource = new EventSource("/sse");

// handler für events ohne Typ
eventSource.onmessage = (e) => {
  console.log(e)
};

// handler für events vom Typen eventType
eventSource.addEventListener('eventType', (e) => {
  console.log('eventType', e)
});
```


Server Sent Events Fazit

- erfordert Browserunterstützung
  - hello darkness (IE, Edge legacy) my old friend
  - –> Polyfill
- Vorteil ggü. Long Polling: Verbindung bleibt auch über mehrere Messages hinweg offen
- aber: simplex

Websockets



- verwendet spezielles URL Schema (`ws://` und `wss://`)
- Client initialisiert Websocket Connection mit spezieller HTTP Request
- Server antwortet mit HTTP Response

initiale HTTP Request des Clients

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```


initiale HTTP Request des Clients

- Request-URI: identifiziert die Websocket Connection
  - erlaubt mehrere Websocket Connections pro Server
- Sec-WebSocket-Protocol: Liste von unterstützten Subprotokollen
- Origin: Schutz vor cross-origin Verwendung
- Sec-WebSocket-Key: verwendet für Handshake

Handshake HTTP Response des Servers

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```


handshake HTTP Response des Servers

- Status 101: erfolgreiche Websocket Verbindung
- Sec-WebSocket-Accept: vervollständigt den Handshake
  - Nonce des Client "dGhlIHNhbXBsZSBub25jZQ=="
  - Server: base64(sha1(concat(nonce, "258EAFA5-E914-47DA-95CA-C5AB0DC85B11")))
  - –> 83pPLMBiTxaQ9kYGzzhZRbK+xOo="
- Sec-WebSocket-Protocol: **eins** der Client-Protokolle wird gewählt
  - Protokolle können standardisiert sein (mqtt)
  - oder auch nicht (chat)

Websockets Praktisch

Dryads wake zeigen.

Websocket Fazit

- ermöglicht full-duplex über persistente TCP Verbindung
- benötigt Browserunterstützung (ab IE 10)
- Subprotokolle müssen implementiert werden
- Vorteile Websocket-Libraries:
  - Fallback auf Long Polling
  - Channels (multiplexing über WS)

Zusammenfassung

- Problem: Server initiierte Kommunikation
- Long Polling:
  - 1 Connection pro Message Austausch
  - benötigt keine Browserunterstützung
- SSE:
  - mehrere Server-Messages pro Connection
  - benötigt Browserunterstützung
  - simplex
- Websockets:
  - eine Connection für mehrere Messages
  - benötigt Browserunterstützung

HTTP

Wenn nichts mehr hilft (und du alle Entwicklungsteams finanzierst), änder' den Standard.

Lernziele

- verstehen der Problematik in Hinsicht auf mehrere Requests
- kennen der HTTP 1 Erweiterungen
- verstehen der 'Userspace' Lösung
- kennen von HTTP 2
- verfluchen von IE

Das Problem

```
<html lang="">
    <head>
        <meta charset="utf-8">
        <link href="/a.css" rel="stylesheet" type="text/cs
        <!-- ... -->
        <link href="/g.css" rel="stylesheet" type="text/cs
    </head>
    <body>
        <script src="/a.js"></script>
        <!-- ... -->
        <script src="/g.js"></script>
    </body>
</html>

touch {a, b, c, d, e, f, g}.{css, js}
```


Wireshark Capture

GET / HTTP/1.1	HTTP: GET / HTTP/1.1
HTTP/1.0 304 Not Mod	HTTP: HTTP/1.0 304 Not Modified
GET /a.js HTTP/1.1	HTTP: GET /a.js HTTP/1.1
GET /b.js HTTP/1.1	HTTP: GET /b.js HTTP/1.1
GET /c.js HTTP/1.1	HTTP: GET /c.js HTTP/1.1
GET /a.css HTTP/1.1	HTTP: GET /a.css HTTP/1.1

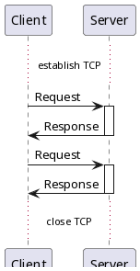
usw.

- 15 HTTP-Requests (index + 7 CSS + 7 JS)
- –> 15 TCP Connections

Persistent Connections

- ab HTTP 1.1 default
- unterliegende TCP Connection wird nicht nach jeder Response geschlossen

Persistent Connections



Probleme mit Pipelining:

- Server bearbeitet Anfragen immer noch sequentiell
- Antworten müssen in gleicher Reihenfolge wie Requests gesendet werden
- Implementierungen waren buggy und in Browsern nicht der default

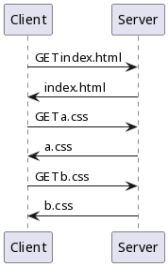
caniuse HTTP2?

Danke IE!

HTTP 2

- aus SPDY (sprich speedy) Protokoll hervorgegangen
  - SPDY von Google entwickelt
- größtenteils mit HTTP 1.1 kompatibel
- HTTP 2.0 erfordert keine encryption
- header compression
- HTTP/2 Server Push
- Multiplexing

Bisher



HTTP 2 Multiplexing - Nachteile

- Annahme: wir multiplexen 2 Streams über 1 TCP Verbindung
- TCP: buffert Frames bis alle vorherigen Frames erhalten wurden
- Packte Loss blockiert beide Streams!

Problem: Head of line Blocking

Browser muss auf Erhalt der Response warten, bevor neue Request über selbe TCP-Connection abgesetzt werden kann.

Userspace: Domain Sharding

- Browser erlaubt z.B.: 6 parallele Connections zu gleichem Hostname
- –> unterschiedliche Hostnames = mehr parallele Connections
- wir hosten unserer assets auf verschiedenen subdomains wie `www1`, `www2`

```
<link href="www1.example.com/a.css" rel="stylesheet" ty
<link href="www2.example.com/b.css" rel="stylesheet" ty
<!-- ... -->
```


Userspace: Webpack

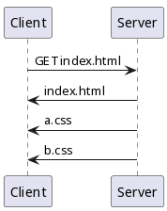
- JS Build Tool
- dank IE momentan weitverbreitet
- unterstützt Bundles
  - mehrere JS, CSS, etc. Dateien werden zu einem Bundle (einzelne Datei) zusammengefasst
  - mehrer Bundles pro Projekt verwendbar (chart.html, table.html)
  - Code der in verschiedenen Bundles verwendet wird kann in separates Bundle ausgelagert werden
- unterstützt Übersetzung von ES6 zu JS, das von IE verstanden wird (Babel)
- Minification uvm. wird auch unterstützt

HTTP 2 Encryption

- sollte eigentlich in den Standard
- war von einigen Standardisierungsteilnehmern aber unerwünscht
- Firefox, Chrome, Safari, Opera, IE und Edge:
  - unterstützen HTTP2 nur über TLS
- Encryption dadurch de facto im Standard
- kann problematisch sein bei Endkunden

Server-Push

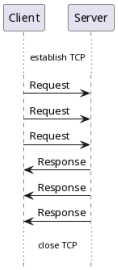
Server Push erlaubt dem Server Stylesheets etc. bereits vor der Anfrage zu senden.



HTTP 2 Multiplexing - HTTP/1?

At 2% packet loss (which is a terrible network quality, mind you), tests have proven that HTTP/1 users are usually better off - because they typically have up to six TCP connections to distribute lost packets over — https://http3-explained.haxx.se/en/why-quic/why-tcphol

Request Pipelining



Mozilla zu Domain Sharding

Unless you have a very specific immediate need, don't use this deprecated technique; switch to HTTP/2 instead. In HTTP/2, domain sharding is no longer useful: the HTTP/2 connection is able to handle parallel unprioritized requests very well. Domain sharding is even detrimental to performance. Most HTTP/2 implementations use a technique called connection coalescing to revert eventual domain sharding.

-

Webpack Nachteile (subjektiv):

- Webpack ist komplex
- schlechte Performance
- Bundleoptimierung benötigt viel Arbeit
- automatisierte Codeoptimierung durch JS dynamische Aspekte schwer
  - Google Closure Compiler? –> viel Aufwand!

HTTP 2 Server Push

Ein Server kann weitere Daten senden.

Bisher: Client ruft `index.html` auf, danach werden Stylesheets etc. aus `index.html` angefragt.

HTTP 2 Multiplexing

- ähnlich Pipelining
- aber: Responses müssen nicht in selber Reihenfolge eingehen
- Congestion Control
  - Browser verwendet mehrere TCP Connections
  - Congestion Control pro TCP Connection
  - Multiplexing erreicht bessere Congestion Control durch Verwendung einer Verbindung
- löst viele der hier genannten Probleme

Zusammenfassung

- moderne Webseiten fragen viele Ressourcen an
  - –> benötigt viele Connections
- Persistent Connections verringern die benötigten TCP Connections
  - aber immer noch relativ wenig Requests gleichzeitig
- Domain Sharding, Webpack und ähnliches als Userspace Lösungen
- HTTP 2: Server Push und Multiplexing

HTTP 3 (HTTP über QUIC)

> *To address this, I'd like to suggest that – after coordination with the HTTP WG – we rename our the HTTP document to "HTTP/3", and using the final ALPN token "h3". — Mark Nottingham, IETF chair (https://mailarchive.ietf.org/arch/msg/quic/RLRs4nB1lwFCZ_7kOiuz0ZBa35s/)*

- encryption
- verwendet UDP (Latenz, Overhead)
- UDP erlaubt `recvmmsg()` call
  - ruft mehrere UDP Pakete auf einmal aus Kernel ab
  - –> weniger Syscalls –> bessere Performance?
- Benchmarks:
  - 2020: [illegible]

HTTP 3 - UDP

- TCP verhindert effizientes Multiplexing wie in HTTP 2 gesehen
- Lösung neues Transportprotokoll?
  - Viele Router, Firewalls, NATs kennen nur UDP und TCP
  - –> blocken alles andere
  - Transportprotokolle werden im Kernelspace implementiert –> nur langsame Entwicklung möglich
  - Viele TCP Verbesserungen können nicht deswegen nicht flächendeckend verwendet werden

HTTP 3 - Verbreitung

- lt. Google ca. 7% des Netzwerktraffics (Googles Version, nicht IETF)
- NGINX geplant für Version 1.17
- Mozilla: in Entwicklung (Neqo), Teil von Necko (verwendet in Firefox)
- curl 11.9.2019 (HTTP 3 draft-22)

HTTP 3 - Probleme

- verwendet TLS 1.3 mit TLS messages statt records
  - viele TLS Libs stellen keine API dafür zur Verfügung (OpenSSL) — Google hat einen Fork.
- Google + FB: HTTP 3 erzeugt doppelte CPU Last im Vergleich zu HTTP 2
  - UDP in Linux nicht so gut optimiert wie TCP
  - TCP + TLS oft hardwarebeschleunigt, UDP nicht
- Viele Firewalls blocken UDP bis auf Port 53 (DNS)

HTTP 3 - Connection

- eigene Definition einer Connection (über Ids)
  - TCP (IP+Port)
  - Problem: mobile –> ändernde IPs
  - wird durch HTTP 3 gefixt
- Connection unterstützt mehrere Streams
  - einzelne Streams sind in-order
  - unterschiedliche Streams können out-of-order verarbeitet werden
- Flow Control (Flusskontrolle) für Connections und Streams

Zusammenfassung

- HTTP 3: über UDP

https://http3-explained.haxx.se

Weiteres:
https://archive.fosdem.org/2020/schedule/event/http3/

Misc

nicht klausurrelevant

VPN

- wird gerne für erweiterte Privacy empfohlen
  - Usertracking über IP ist kleinstes Problem
- ermöglicht auch Umgehung von gesperrten Ports
  - Steam in der DHBW
- Tip: selber hosten
  - VPN Provider kann gesamten Traffic mitschneiden
  - im kostenlosen VPN Bereich ist dies teilweise der einzige Grund für die Existenz

SSH

- sichere Verbindung auf Server für Remoteterminals
  - praktisch für Server, die sich ohne GUI administrieren lassen
- Port Forwarding:
  - local: `ssh -L 9000:imgur.com:80 user@example.com`
    - lokaler Port 9000 wird durch SSH Verbindung auf `imgur.com:80` weitergeleitet
  - remote: `ssh -R 9000:localhost:22 user@example.com`
    - Port 9000 auf Server wird auf lokalen Port 22 weitergeleitet
    - Bsp: Box A mit dynamischer IP `ssh -R 9000:localhost:22 user@exmple.com`
    - Auf `example.com`: `ssh user@localhost` stellt SSH Verbindung zu Box A her

Klausurthemen

☐ Slides durchgehen

Erfahrung

Sie haben *Erfahrung* mit Netztechnik gesammelt und können sich in jede Schicht einarbeiten.

Überblick

Sie *erkennen* die wichtigsten Dienste aller Schichten und können ihre Eigenschaften und Aufgaben *nennen*.

Sie haben eine *konkrete Vorstellung* des Weges von REST Client oder IMAP über DNS und IP, durch TCP, Fehlerkorrektur und Netzverbindung, bis hinunter auf die analoge Bitübertragung.

Verständnis

Sie können Optimal-Leistung von Netz-Anwendungen *abschätzen*.

Fragen für die Prüfung?

Zusammenfassung

Viel Erfolg beim Lernen!

Bilder: