

Post Office Protocol

- Protokol pro přístup uživatelů k poštovní schránce
- Aktuální je verze 3, RFC 1939, port 110
- Hlavní nevýhody:
 - Otevřené posílání hesla; existuje rozšiřující nepovinný příkaz pro šifrovanou autentikaci (APOP), ale řada klientů ho nemá implementovaný
 - Dopisy je nutno stahovat ze serveru celé; i zde existuje rozšiřující nepovinný příkaz TOP pro stažení začátku dopisu, který je ale opět jen řídce implementovaný
 - Není možné pracovat se strukturou dokumentů
- Dnes podporován spíše kvůli zpětné kompatibilitě a nahrazován protokolem IMAP
- Plaintextová komunikace byla v RFC 8314 prohlášena za Obsoleted

Pro vzdálený přístup k dopisům ve svém mailboxu může uživatel použít buďto POP nebo IMAP protokol.

POP je starší z obou protokolů a vykazuje všechny známé bezpečnostní problémy. Postupem času se objevila rozšíření, která se je pokoušela eliminovat, ale mezitím získal větší popularitu protokol IMAP, takže implementace těchto rozšíření není úplně obvyklá. Dnes se proto spíše používá zabezpečení pomocí TLS a v roce 2018 bylo dokonce plaintextové použití protokolu prohlášeno za Obsoleted.

Zajímavostí je, že první dvě verze pracovaly na principu *push*, tedy že server inicioval přenos dopisů na klienta, zatímco poslední už pracuje na principu *pull*. Nicméně hlavní nevýhodou protokolu zůstávají omezené možnosti práce s mailboxem.

Internet Message Access Protocol

- Modernější, ale složitější nástupce POP
- Aktuální verze 4rev1, RFC 3501, port 143
- Hlavní výhody:
 - Zabudována možnost používat šifrované spojení
 - Server uchovává informace o dopisech (stav)
 - Podpora více schránek (složek)
 - Protokol umožňuje vyžádat pouze část dopisu
 - Je možné nechat na serveru v dopisech vyhledávat
 - Možnost zadat paralelní příkazy
- Šifrování:
 - a) navázání spojení na port 993
 - b) vyvoláno příkazem STARTTLS
- IMAP má implementována většina stávajících MUA

Ačkoliv první verze IMAP protokolu se objevila jen dva roky po POP1, jeho autor od počátku kladl větší důraz na práci se vzdáleným mailboxem a už IMAP2 měl třeba příkaz na hledání dopisu v mailboxu podle několika kritérií nebo stažení pouze určitých informací o dopisu. Vývoj verze 4 převzala skupina pod IETF a tato verze už zahrnuje řadu bezpečnostních a funkčních rozšíření.

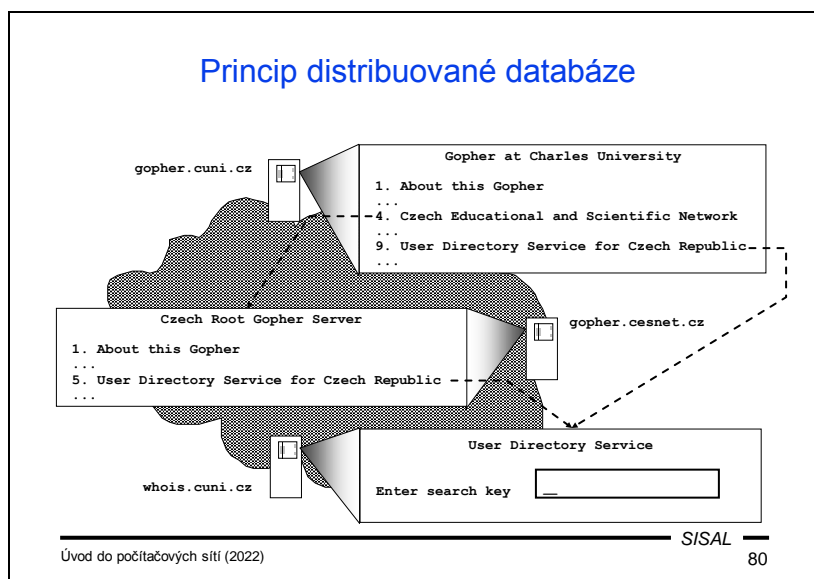
Tato verze je dnes široce podporována poštovními aplikacemi a v souladu se současnými standardy se používá v kombinaci s TLS, a to buď na vyhrazeném portu 993 místo 143, anebo na vyžádání klienta příkazem STARTTLS.

Ukázka IMAP4 protokolu

```
← "*" OK [IMAP4REV1 STARTTLS AUTH=LOGIN] IMAP4 ready
⇒ 1 LOGIN first heslo
← 1 OK User authenticated
⇒ 2 LIST "" ""
← * LIST (HasNoChildren) "/" "INBOX"
← 2 OK LIST completed
⇒ 2 SELECT INBOX
← * 123 EXISTS
← 2 OK [READ-WRITE] SELECT complete
⇒ 3 UID SEARCH NEW FROM "Joe" NOT BODY "Test"
← * SEARCH 1234 1248
← 3 SEARCH complete
⇒ 4 FETCH 1248 (BODY.PEEK[HEADER])
← Received: from ...
← ...
← )
```

Protokol je rovněž textový, ale oproti POP je uživatelsky daleko přívětivější. Klient dokáže pomocí příkazů pracovat s různými mailboxy (foldery a podfoldery), dokáže vyhledávat dopisy podle složitých podmínek, stahovat pouze části dopisů atp.

Zajímavou vlastností je i značkování (tagování) příkazů a odpovědí, což usnadňuje přiřazení odpovědí zadaným příkazům. Klient tedy může paralelně poslat na server více příkazů. Absence tohoto mechanismu např. ve FTP komplikuje implementaci některých operací, jako je třeba přerušení přenosu souboru.



V roce 1991 se objevil systém Gopher, který představoval první celosvětově rozšířenou službu distribuované databáze, tj. databáze informací, které jsou uloženy na obrovském množství serverů a navzájem provázány tak, že uživatel přechází pomocí odkazů z jednoho serveru na jiný, aniž o tom musí nutně vědět. Gopher nabízel ve své době podobnou podporu při vyhledávání informací jako v současnosti web. Už tehdy např. všechny univerzity nabízely pomocí této služby přístup k informacím o studiu, telefonní seznam atd. Při přihlášení na server se uživateli zobrazilo menu a po vybrání požadované položky se zobrazila další stránka obsahující jiné menu, text s požadovanou informací, anebo formulář, po jehož vyplnění a odeslání se uživateli zobrazila nějaká dynamicky vygenerovaná informace. Odkazy mezi jednotlivými stránkami přitom mohly vést na úplně jiný server. Jak vidíte, práce s gopherem se téměř shodovala s tím, jak uživatelé dnes pracují s webem. Podstatný rozdíl spočíval pouze v tom, že gopher poskytoval pouze textové informace. Pokud nějaký odkaz vedl na obrázek, gopher ho sice stáhl, ale uživatel si ho musel prohlédnout v jiném programu.

Hypertext

- Základní myšlenka (1945):
nelineární hierarchický text obsahující vazby, které umožňují pokračovat čtením podrobnější informace nebo příbuzného tématu
- Pozdější rozšíření (1965):
doplnění samotného textu o netextové informace (obrázky, zvuk, video...), někdy se používá pojem *hypermediální text*
- Praktická implementace (1989):
systém World Wide Web vyvinutý v CERNu

To, co gopheru chybělo, byl hypertext.

Myšlenka hypertextu je překvapivě velmi stará. Už v polovině minulého století, tedy v době, kdy jediným zdrojem informace byly tištěné materiály, se objevily úvahy o tom, že by text mohl obsahovat vazby na vysvětlující nebo příbuzné texty. Tedy nějak jako rejstřík, ale dostupný přímo z textu, nikoliv jako souhrn na konci knihy. Nicméně realizace v té době pochopitelně nebyla možná.

O dvacet let později se základní myšlenka rozšířila o možnost, aby text přímo obsahoval i netextové prvky, a objevil se alternativní název *hypermediální text*. Pořád se ale nacházíme v době, kdy na sebe realizace musel nechat čekat.

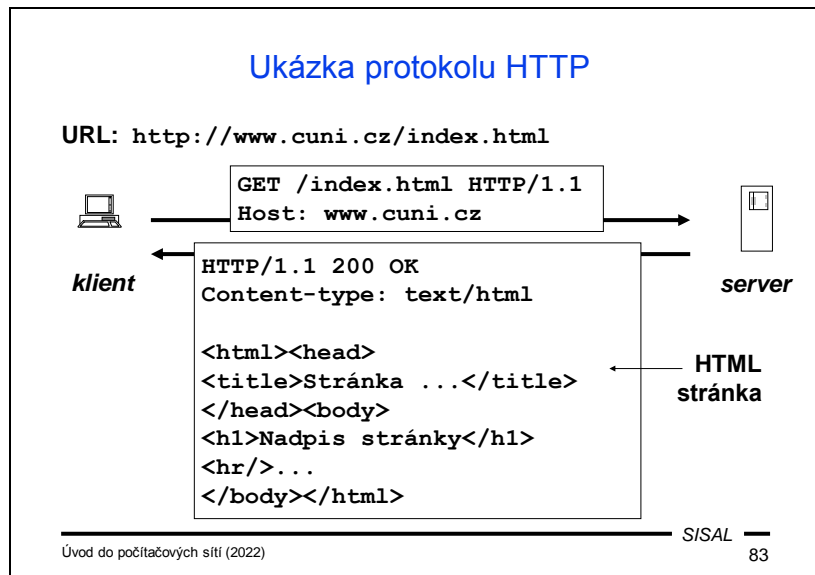
Čekala dalších dvacet let, přesněji do roku 1989, kdy se ve švýcarském středisku CERN zrodila myšlenka systému na šíření (původně interních) informací rozličného charakteru pomocí služby později nazvané World Wide Web.

World Wide Web

- WWW je distribuovaná hypertextová databáze
- Základní jednotkou je hypertextová *stránka* (dokument), kterou server posílá na žádost klientům
- Dokumenty jsou psány v textovém jazyce HTML (Hypertext Markup Language)
 - popisuje obsah i formu
 - konkrétní zobrazení je v režii klienta resp. uživatele
- Dokumenty existují staticky (cesta v URL pak obvykle odpovídá skutečné relativní cestě na disku serveru) nebo se vytvářejí dynamicky dle požadavků klienta
- Přenos stránek se odehrává pomocí protokolu HTTP (Hypertext Transfer Protocol), chybějící zabezpečení se řeší standardně TLS mezivrstvou (HTTP+SSL=HTTPS)

Jednoduše řečeno, web je distribuovaná hypertextová databáze. Podobně jako u gopheru jsou globální informace distribuované na obrovském množství serverů po celém světě a jsou navzájem provázány tak, aby bylo možné mezi nimi procházet bez toho, že by se uživatel musel starat o to, k jakému serveru je zrovna připojený.

Základní jednotkou informace je *stránka*, tento pojem odpovídá tomu, co před webem existovalo v gopheru, ale význam se dost odlišuje. Stránka na webu se někdy označuje jako *dokument*, což lépe odpovídá tomu, že stránka na webu nemá nic společného se stránkou v nějakém tištěném materiálu. Webová stránka může být uložena jako soubor na nějakém serveru (pak jí říkáme *statická*), anebo ji na základě parametrizovaného požadavku uživatele server vygeneruje *dynamicky*. V obou případech procedura probíhá tak, že klient vyšle požadavek na server a server odpoví textem stránky (nebo chybovou zprávou). Stránky se píšou obvykle v jazyce HTML (Hypertext Markup Language), který obsahuje prostředky na vyjádření obsahu (tj. text, vložené obrázky, odkazy apod.) i formy (font, barva, tabulky, zarovnání apod.). Jazyk popisuje představu autora o tom, jak by mělo zobrazení stránky na obrazovce klienta vypadat, ale přesná podoba je dána mnoha dalšími faktory (implementace vlastností jazyka v prohlížeči, uživatelská nastavení apod.).



Pro přenos webových stránek se používá protokol HTTP (Hypertext Transport Protocol). Ve své základní verzi (1.1) se jedná o textový protokol, v němž

- klient naváže spojení na server a pošle textový požadavek obsahující URL (zapsané zčásti na první, příkazové řádce a zčásti v následujících hlavičkách) a některé další parametry
- server odpoví řádkou obsahující kód odpovědi (v našem příkladu 200), hlavičkami obsahujícími mj. MIME typ zaslaného dokumentu a požadovaný dokument (v našem případě stránku v jazyce HTML). V případě chyby bude odpověď obsahovat jiný kód a HTML text chybové stránky.

Hypertext Transfer Protocol v.1

- V současnosti převažuje verze 1.1, RFC 7230, port 80
- Obecný formát zpráv:
 - úvodní řádka (požadavek/odpověď)
 - doplňující hlavičky
 - požadavek: jazyk, kódování, stáří stránky, autentikace,...
 - odpověď: typ dokumentu, kódování, expirace,...
 - (volitelné) tělo dokumentu
- Kódy odpovědí:
 - 1xx **informativní odpověď** (požadavek přijat, zpracovává se)
 - 2xx **kladná odpověď** (definitivní)
 - 3xx **přesměrování** (očekává se další požadavek od klienta)
 - 4xx **chyba na straně klienta** (nesprávný požadavek)
 - 5xx **chyba na straně serveru** (nepodařilo se vyhovět požadavku)

V současnosti se většinou provozuje verze 1.1 protokolu. Tato verze je textová, požadavek i odpověď obsahují úvodní řádku, po níž většinou následuje záhlaví (prakticky stejného formátu, jako jsme viděli u pošty), prázdná řádka a po ní (v některých situacích) může následovat tělo dokumentu.

V požadavku se úvodní řádka skládá z názvu metody (např. GET pro žádost o poslání stránky), cesty (tj. části URL za názvem serveru) a označení verze protokolu. Z hlaviček je ve verzi 1.1 povinná pouze hlavička Host, která specifikuje, na jaký server se klient obrací. Na stejné IP adrese totiž může pracovat více serverů a právě pomocí této hlavičky se rozlišuje, kterému požadavek patří. Mezi další užitečné hlavičky patří specifikace jazyků a kódování, které klient akceptuje, požadované stáří stránky, data pro autentikaci aj. Tělo v běžném požadavku zastoupeno nebude, výjimku tvoří speciální typy požadavků, při nichž se buďto posílají na server nějaké dodatečné parametry, anebo se na server uploaduje celý dokument.

V odpovědi serveru je první řádka stavová, obsahuje opět číslo protokolu, trojmístný kód odpovědi a slovní popis odpovědi. Hlavičky se týkají formálních náležitostí (např. čas poslední změny nebo čas expirace stránky), detailů protokolu (např. způsob přenosu) a vlastností posílaného dokumentu (např. MIME hlavičky). Tělo odpovědi obsahuje požadovaný dokument, anebo text chybové zprávy.

Kódy odpovědí v zásadě odpovídají tomu, co známe z dřívějších protokolů. 2xx je konečná kladná odpověď, 1xx a 3xx znamenají, že požadavek je víceméně v pořádku, přičemž 1xx znamená, že míč je na straně serveru (ještě přijde další odpověď), zatímco 3xx přehrává míč na stranu klienta (klient musí zadat jiné URL, chce-li obsah stránky). Pouze kódy chybových odpovědí mírně změnily sémantiku,

4xx znamená nějakou chybu na straně klienta (špatné URL, autentikace apod.), zatímco 5xx chybu na straně serveru.

Metody HTTP

Metoda	Tělo požadavku	Tělo odpovědi	Bezpečná, Idempotentní	
GET	—	dokument	ano	ano
HEAD	—	—	ano	ano
POST	parametry	dokument	ne	ne
PUT	dokument	—	ne	ano
DELETE	—	výsledek	ne	ano
CONNECT	⇐ tunel ⇒			

Nejběžnějšími metodami HTTP jsou GET, HEAD, POST, PUT, DELETE a CONNECT.

- GET je základní metoda, kterou klient žádá o poskytnutí stránky nebo dokumentu. Tělo požadavku je prázdné, tělo odpovědi obsahuje požadovanou stránku. V případě chyby obsahuje tělo odpovědi (stejně jako u ostatních metod) obvykle text HTML stránky s popisem chyby. Metoda je v RFC deklarovaná jako *bezpečná*, což znamená, že **nemění** obsah serveru. I v případě, že požadavek vznikl vyplněním a odesláním HTML formuláře a obsahuje data formuláře v URL v části **query**, neměl by způsobit změnu dat uložených na serveru. Metoda je rovněž definovaná jako *idempotentní*, což znamená, že **opakované** použití má **stejný** efekt.
- HEAD je zjednodušená forma metody GET, kdy odpovědi jsou pouze hlavičky a nikoliv dokument. Jejím smyslem je zjistit, zda je požadovaná stránka dostupná, jak je velká, zda je dostupná v určitém jazyce apod.
- POST je metoda, jejímž účelem je poslat na server určité informace (obvykle data z formuláře) a dostat jako odpověď obsah určitého dokumentu. Může být používána jako způsob, jak získat dynamickou stránku, ale zároveň je legitimní pomocí ní **měnit** obsah dat na serveru. Proto také metoda není idempotentní, protože každé zavolání může mít jiný efekt (např. se neustále zvětšuje obsah nějakého souboru na serveru).
- PUT je metoda, která má za úkol přepsat obsah dokumentu na serveru dokumentem zaslaným v požadavku. Není tedy rozhodně bezpečná, ale je idempotentní – opakování požadavku by mělo mít stále stejný efekt.
- DELETE je metoda pro smazání dokumentu na serveru. I tato metoda je idempotentní – při opakování sice dostaneme jinou zprávu (že dokument již byl smazán), ale výsledkem je v každém případě neexistence daného dokumentu.

- CONNECT je specifická metoda, jejímž výsledkem je otevření spojení dle požadovaných parametrů a tím vlastně vybudování tunelu, kterým je možné přes HTTP spojení realizovat jiné spojení v úplně jiném protokolu. To je sice užitečná vlastnost, kterou webové servery často využívají, ale zároveň to představuje bezpečnostní riziko, protože to umožňuje obejít omezení daná bezpečnostní politikou sítě.

Vlastnosti HTTP v.1

- Odpověď na jeden požadavek je obvykle jeden dokument (stránka, obrázek,...)
- Po jednom (perzistentním) spojení může jít postupně více požadavků, klienti si obvykle otevírají současně několik spojení
- Požadavky jsou nezávislé, komunikace je bezstavová; stav je nutno přenášet jako dodatečná data, tzv. *cookies*:
 - server vygeneruje cookies na základě dat z daného logického dialogu a pošle je v hlavičkách klientovi
 - prohlížeč si cookies ukládá a při dalších požadavcích na stejný server tato data přidává do hlaviček požadavku
 - data z cookies může server využít pro shromažďování informací o uživateli

Styl, jakým se používá *www* prohlížeč, vede někdy uživatele k pocitu, že okamžikem připojení se na nějaký server vznikl mezi klientem a serverem komunikační kanál, který přenáší data logického dialogu po celou dobu, po níž je k serveru připojen. To je ale zcela v rozporu s realitou. Ve skutečnosti je každý uživatelský požadavek odesílán klientem jako zcela **nezávislý**. Pokud se *www* stránka sestává z textu a tří obrázků, pak to dokonce budou čtyři nezávislé požadavky. Kliknutí na odkaz na stránce vygeneruje dalších několik nezávislých požadavků. Verze 1.1 přinesla novinku v tom, že zavedla **perzistentní** spojení, což znamená, že po skončení jednoho požadavku klient nemusí zavírat TCP spojení a navazovat nové, ale může použít totéž TCP spojení pro více sériových požadavků. Prohlížeče navíc často dělají to, že při připojování na nějaký server rovnou otevírají víc paralelních TCP spojení, neboť očekávají, že většina stránek obsahuje více komponent ze stejného serveru, takže se vyplatí mít patřičná TCP spojení už připravená pro následné požadavky.

Fakt, že jednotlivé požadavky jsou nezávislé, znamená, že celá komunikace je **bezstavová**, neboli že server obecně nemá představu o tom, které požadavky ze stejného klienta „patří k sobě“ a v jakém stavu je vlastně logický dialog, který s uživatelem vede. Pokud uživatel během práce předá serveru nějaké informace potřebné pro další práci (např. nějaká nastavení), server by je musel po klientovi chtít u každého požadavku znova. Problém bezstavovosti komunikace se řeší pomocí tzv. *cookies*. Jsou to data, která vygeneruje server na základě informací od uživatele a pošle je klientovi při odpovědi ve formě hlaviček Set-Cookie (resp. Set-Cookie2). Prohlížeč si tato data uloží a posílá je posléze ve formě hlavičky Cookie při každém dalším požadavku na stejný server. Server podle nich dokáže identifikovat spojení resp. uživatele a zabezpečit to, že uživatel najde na serveru při každém dalším požadavku prostředí, které odpovídá existujícímu (nebo minulému) dialogu.

Tento charakter cookies je důležitý, protože z něj plyne několik bezpečnostních důsledků:

- Cookies sama o sobě nepředstavují žádné přímé nebezpečí, nemohou např. obsahovat viry apod.
- Představují ale nebezpečí sekundární, protože webový server si do nich může ukládat libovolnou informaci, kterou o uživateli zjistí, a tu pak libovolně používat. Např. pomocí reklamních bannerů umístěných na různých stránkách může server sbírat o uživateli komplexnější informace a uskutečňovat cílenou reklamu.
- Nebezpečí rovněž představuje to, pokud se cookies uložená v počítači dostanou do cizích rukou.

Hypertext Transfer Protocol v.2

- Web je dnes úzce svázán s komercí, takže kolem vývoje inovace HTTP bylo trochu rušno
- Momentálně se prosazují implementace dle RFC 7540
 - binární protokol, lze na něj přejít v rámci HTTP/1 spojení
- Hlavní motivace: větší propustnost
- Metody:
 - vlastní multiplexing více *streamů* v rámci jednoho TCP spojení (streamy se neblokují, dají se prioritizovat)
 - server může poslat (*push*) více dat, než požadoval klient, pokud usoudí, že je klient bude potřebovat
 - v současné době narůstá rozsah hlaviček, navíc často mají podobný obsah - lze je efektivně komprimovat
- Neprošlo: povinné šifrování

Nová verze protokolu HTTP se snaží postihnout současné trendy v používání webu, aby pokud možno zvýšila „rychlost“ z pohledu uživatele. Zásadní změnou je změna z textového charakteru na binární. To je samozřejmě nepříjemné z mnoha důvodů: na verzi 1 zkušenější uživatel ocení to, že se může pokusit kontaktovat server přímo, předat mu svůj požadavek a vidět přesně, co server odpověděl, a správce zase ocení možnost monitorovat příp. nekalou aktivitu uživatelů či serverů. Nicméně tahle idylická doba vlastně neskončila nástupem HTTPv2, ale masivním nástupem používání HTTPS. Na transportní vrstvě tedy dneska už vidíme HTTP v textové podobě jen velmi zřídka, takže změna na binární protokol ve skutečnosti není z provozního hlediska tak zásadní.

Verze 2 se nespokojila se službami TCP a vytvořila si koncept vlastních *streamů* provozovaných nad jedním TCP spojením. Tyto vlastní streamy jsou efektivnější, protože se navzájem neblokují, a dají se prioritizovat podle potřeb aplikační vrstvy.

Server může ve verzi 2 poslat klientovi i data, o něž sice zatím nežádal, ale o nichž server usoudí, že je klient bude stejně vzápětí potřebovat. Klasickým příkladem je stránka s obrázky – u verzi 1 jsme se učili, že na každý obrázek musí klient vytvořit nový požadavek, ve verzi 2 to není nutné.

Další vlastností nové verze protokolu je možnost komprimovat hlavičky. Zejména s nárůstem používání autentikace hodně nabobtnaly i hlavičky, které navíc často mají stejný obsah (autentikace, nabízené jazyky, kódování, tohle všechno se prakticky po celou dobu dialogu nebude měnit a je zbytečné vše posílat s každým požadavkem znova).

Jazyk HTML

- Hypertext Markup Language, vývoj v posledních letech poněkud dramatický, 2014 vyšla kompromisní verze 5
- Vlastní textový obsah stránky je doplněn doplňujícími informacemi, značkami: strukturálními (např. odstavec), sémantickými (např. adresa), formátovacími (např. tučně)
- Je aplikací staršího SGML (Standard Generalized ML) a předchůdcem XML (Extensible ML)
- Formát značky: `<znacka [atributy]>`
- Volný formát řádek (bílé znaky nevýznamné)
- Speciální znaky - entity (`<`, `>`, `&`, ` `;...)
- Komentáře (`<!-- ... -->`)

Učivo o HTML a návazná témata se budou probírat v druhé části přednášky.

Telnet

- Protokol pro přihlašování na vzdálené stroje, port 23
 - Zkratka z *Telecommunication Network*
 - Jeden z nejstarších protokolů, poprvé v RFC 97 (1971!)
 - Uživatel má k dispozici síťový virtuální terminál (NVT), protokol přenáší oběma směry znaky a příkazy pro řízení NVT (slabiny: např. nerozlišuje příkaz a odpověď)
 - Hlavní nevýhoda: otevřený přenos dat (řeší až rozšíření podle RFC 2946, které ale přichází pozdě)
 - Dnes:
 - přístup na síťová zařízení v rámci odděleného segmentu LAN
 - ladění jiných protokolů:
- ```
> telnet alfik 25
220 alfik.ms.mff.cuni.cz ESMTP Sendmail ...
HELO betynka
250 alfik Hello betynka, pleased to meet you
```

Úvod do počítačových sítí (2022)

SISAL

99

Telnet je další z velmi starých protokolů a představuje první řešení vzdáleného přihlášení na jiný stroj, a to formou emulace terminálu. Klient a server si předávají povely uživatele a reakce protistrany tak, že uživatel má k dispozici podobné prostředí, jaké by seděl u reálného terminálu.

Poučným prvkem této komunikace je způsob výměny některých povelů. Princip emulace terminálu je založen na tom, že klient a server se musejí domlouvat na tom, jakou část práce kdo z nich udělá. Když uživatel stiskne klávesu, někdo musí způsobit její zobrazení na obrazovce (tzv. echo). Klient to buďto může udělat sám, anebo počkat, až server po zpracování klávesy pošle zpátky pokyn pro zobrazení znaku. Naopak v případě, že uživatel právě píše heslo, nebude echo dělat ani klient ani server. Na tom se ale musejí domluvit. Existují na to čtyři zprávy DO ECHO, DON'T ECHO, WILL ECHO a WON'T ECHO. Problém je v tom, že zpráva **neobsahuje** příznak, zda jde o povel nebo odpověď! Takže pokud dojde omylem k rozsynchronizování významu zpráv, má to fatální důsledky. Řekněme, že klient pošle povel DO ECHO, a protože nepříjde odpověď, zopakuje ho, ale nepoznamená si, že poslal povely dva. Server správně odpoví 2x WILL ECHO, jenomže klient druhou odpověď bude považovat za novou zprávu a odpoví na ni DO ECHO. Tím odstartuje nekonečnou smyčku extrémně rychlé výměny zpráv DO ECHO/WILL ECHO. Poučením tedy je v návrhu protokolu určitě nešetřit za každou cenu a jeden bit na příznak požadavek/odpověď rozhodně obětovat.

Z bezpečnostního hlediska má telnet všechny slabiny starých internetových protokolů. Otevřený přenos hesel sice řeší RFC 2946, ale přichází v době, kdy už existovala lepší náhrada, SSH. Nicméně i dnes se kvůli relativní jednoduchosti protokol používá v místech, kde nehrozí odposlech hesel – např. na odděleném



segmentu lokální sítě určeném pro správu infrastruktury. I tato zařízení ale postupem času přecházejí na SSH.

Aplikace **telnet** se ale dá použít ještě k jinému účelu, a to navázání spojení na nějaký server pracující v jiném (ideálně textovém) protokolu. Můžeme tak např. vyzkoušet, jak reaguje na jednotlivé povely třeba SMTP nebo HTTP server.

## Secure Shell (SSH)

- Bezpečná náhrada starších protokolů pro vzdálené přihlašování resp. přenos souborů
  - klient se pokouší ověřit server
  - komunikace je šifrovaná
- Aktuální verze 2, RFC 4250-4254, port 22
- SSHv2 kromě základní funkce umožňuje:
  - otevírat paralelně více zabezpečených kanálů
  - tunelovat zabezpečeným kanálem jiný provoz
  - zpřístupnit souborový systém (SSHFS)
- Klienti (windows): putty, winscp
- Příkazy (unix):

```
ssh [user@]host [command]
scp [-pr] [user@[host:]]file1 [user@[host:]]file2
```

SSH (Secure shell) je protokol, který slouží ke vzdálenému přihlášení a přenosům souborů. Na rozdíl od starších protokolů je založen na důsledném ověřování serveru, k němuž se připojujeme a šifrování veškeré komunikace.

V současnosti se používá verze 2, která rozšiřuje možnosti zejména o

- otevírání paralelních kanálů; je možné být např. současně přihlášen pomocí virtuálního terminálu a současně přenášet soubory,
- tunelování; komunikace z jedné strany SSH kanálu se přenáší kanálem a na druhé straně se nasměruje na nějaký tamější server (možnost obcházení firewallu),
- SSHFS; server zpřístupní část svého souborového systému prostřednictvím kanálu tak, že se klientovi jeví jako lokální souborový systém.

Na platformě MS Windows jsou k dispozici volně dostupné programy pro vzdálené přihlašování (např. **putty**) a přenos souborů (např. **WinSCP**), na UNIXu existují pro tento účel řádkové příkazy **ssh** a **scp**.

## Bezpečnost SSH

- Klient ověřuje server
  - na základě kontroly klíče (potvrzuje uživatel)
  - certifikátu (ověřeno podepisující autoritou)
- Server ověřuje uživatele
  - pomocí hesla
  - pomocí výzev a odpovědí (OTP)
  - pomocí veřejného klíče (server posílá výzvu zašifrovanou klíčem uživatele, klient odpovídá plain textem)
- Strategie používání klíčů
  - důkladně ověřovat klíč serveru, pozor zvl. při změně (nebezpečí útoku „*man-in-the-middle*“)
  - přihlášení bez hesla vázat na privátní klíč s heslem
  - na méně důležité cíle je možné i bez hesla, ale rozhodně nikoliv recipročně ( $A \rightarrow B$  i  $B \rightarrow A$ ) - ochrana proti *červům*

Před tím, než klient začne se serverem komunikovat o autentikaci uživatele, ověřuje, zda je skutečně připojen na správný server. Používá se buďto možnost serverového certifikátu, anebo klient pouze zobrazí uživateli otisk klíče serveru a uživatel rozhodne, zda se jedná o správný klíč a tedy i server. Pokud uživatel identitu serveru potvrdí, aplikace si toto rozhodnutí uloží a při dalším přihlášení se již neptá.

Pokud se pomocí SSH hlásíme na nějaký server poprvé, teoreticky bychom si vždy měli důkladně ověřit správnost klíče, co posílá server. Ve skutečnosti je riziko v normálním případě velmi malé. Musíme zhodnotit, jak důležitý je server, na který se hlásíme, jak pravděpodobné je, že by mohl být vystaven útoku, jak tajné heslo na daném serveru máme (na kolika jiných serverech máme stejné heslo), a pokud tato rizika vyhodnotíme jako malá, je možné ověřování přeskočit. Rozhodně to ale **neplatí**, pokud zpráva o změně klíče na serveru přijde **neočekávaně**. Pokud server, na který se hlásíme už rok, se najednou představí jiným klíčem, je třeba zpozornět a vše důkladně ověřit! Toto by mohl být symptom útoku. Můžeme např. u správců serveru telefonicky ověřit, že opravdu přeinstalovali systém a mají nový klíč. Pokud takové ověření není možné, měli bychom přihlášení **ukončit**.

Jakmile je ověřen server, začne si s ním klient vyměňovat informace vedoucí k ověření uživatele. Základní možností je standardní použití jména a hesla. V tomto okamžiku je již komunikace šifrovaná, takže nehrozí žádné riziko prozrazení. Druhým běžným způsobem je používání klíčů. Uživatel si na klientovi vytvoří pár klíčů (resp. může sdílet stejný pár klíčů na více klientech), přičemž veřejný klíč uloží na určené místo na serveru. Existuje-li takový klíč a klient se jím prokáže, uživatel nemusí psát heslo ke vzdálenému účtu.

Nastavit si přihlašování pomocí klíče je velmi pohodlné, ale skrývá to v sobě riziko. Pokud dojde k útoku na náš účet, otevíráme tím dveře k dalším serverům. Řada lidí používá dvouúrovňový systém klíčů: na strojích malé důležitosti používá jeden pár klíčů a na důležitých strojích jiný pár, přičemž tajný klíč z tohoto druhého páru je ještě chráněn lokálním zaheslováním. Pokud totiž klíč není chráněn heslem, může ho útočník použít k přihlášení na další stroje. Teoreticky útočník nemá informace o tom, na jaký další stroj by se měl pokusit přihlásit. Má ale informace o tom, **z jakých** jiných počítačů máte povolen přístup na tento svůj účet (stačí projít seznam uložených veřejných klíčů). Nic mu tedy nebrání v tom vyzkoušet přihlášení na každý z tohoto seznamu účtů, zdali u některého z nich nemáte náhodou také nastaveno přihlašování bez hesla recipročně. A pokud ano, dostane se na další počítač. Toto je princip práce internetových **červů**. Obrana je jednoduchá – vyhněte se tomu mít povolené přihlašování mezi dvěma svými účty tam i zpět, a pokud už to opravdu musíte mít takhle nastavené, chraňte tajný klíč heslem.

## Voice over IP

- Obecné označení technologií pro přenos hlasu po IP
- Lze realizovat různými navzájem nekompatibilními způsoby:
  - standard H.323
  - standard SIP
  - proprietárně (Skype)
- Celá řada problémů:
  - digitalizace hlasu
  - dohadování vlastností zařízení
  - nalezení partnera
  - propojení s běžnou telefonní sítí

Termín **Voice over IP** neoznačuje jeden konkrétní protokol, ale obecně jakýkoliv nástroj na přenos (nejen) hlasu pomocí TCP/IP sítě. Těchto možností existuje celá řada, některé z nich jsou realizovány prostřednictvím obecnějších protokolů (třeba Skype přes HTTP), ale nás budou zajímat hlavně ty, které používají vlastní protokoly.

Celé problematika je pochopitelně hodně složitá, musí pokrývat oblast digitalizace hlasu se všemi jejími metodami, variantami, parametry apod., musí pokrývat domluvu protistran o tom, jaké jsou možnosti na té či oné straně a jak tomu přizpůsobit chování protistrany, musí zahrnovat principy řešení otázky nalezení partnera pro komunikaci a ideálně i možnost, jak celý systém napojit na běžnou telefonní síť tak, aby bylo možné komunikovat z běžného telefonu na číslo reprezentované VoIP zařízením a naopak.

## H.323

- Komplexní řešení multimediální komunikace od ITU
- Založeno na ASN.1 (binární, bitové protokoly)
- Zahrnuje celou řadu dílčích protokolů, mj.:
  - H.225/RAS (Registration/Admission/Status) pro vyhledávání partnera pomocí tzv. *gatekeeper* uzlů
  - Q.931 (síťová vrstva ISDN) řeší navazování spojení
  - H.245 řeší řízení hovoru (dohodu o používaných vlastnostech zařízení)
  - RTP kanály (Realtime Transport Protocol, RFC 3550) se používají pro vlastní přenos multimediálních dat
  - RTCP (RTP Control Protocol) zabezpečuje jejich řízení
- Dnes postupně nahrazováno SIP

Telekomunikační společnosti (resp. ITU, International Telegraph Union) nastoupily už dávno na cestu, jak jednotlivé procesy digitalizovat. Pro jednotlivé oblasti existuje řada samostatných protokolů, které se souhrnně označují jako **H.323** a které řeší problematiku vyhledávání partnera, navazování spojení, dohodu o vlastnostech zařízení a nakonec i vlastní přenos audio nebo video dat. Bohužel ale tyto protokoly jednak nejsou všechny volně dostupné a jednak pocházejí z doby, kdy datové toky nedisponovaly kapacitou, kterou máme dnes, a proto se v nich zoufale šetří každým bitem. Ano, tyto protokoly nejen, že nejsou textové, ale jsou **binární** v pravém významu toho slova. Hranicemi hodnot zde nejsou bajty, ale **bity** a navíc každý bit může ovlivnit hranice dalších hodnot. Protokoly jsou tak nejen nečitelné pomocí editoru, ale dokonce i velmi obtížně implementovatelné – představte si, jak naprogramujete čtení desetibitového čísla, které zasahuje do tří bajtů, ale jenom někdy...

Protokoly, které mají na starosti úvodní fáze přípravy hovoru, jsou dnes postupně nahrazovány protokolem SIP. Vlastní audio nebo video data se posílají pomocí protokolu RTP (Realtime Transport Protocol) a k řízení jeho činnosti se používá řídicí protokol RTCP. Oba protokoly jsou sice rovněž binární, ale vzhledem k tomu, že jejich úkolem je přenos binárních multimediálních dat, je to adekvátní jejich účelu a navíc je jejich popis volně dostupný v RFC 3550.

## Abstract Syntax Notation 1

- Formální definice datové struktury, př.:

```
Answer ::= CHOICE {
 word PrintableString,
 flag BOOLEAN }

AlgorithmIdentifiers ::= SET OF AlgorithmIdentifier

SignedData ::= SEQUENCE {
 version CMSVersion,
 digestAlgorithms AlgorithmIdentifiers,
```

- Implementace pochází z 80. let (a je to na ní znát)
  - př.: výčtový typ (enumerace) se implicitně zapíše do tolika **bitů**, kolik je třeba, předchází jim ale bit, který signalizuje rozšíření, což znamená zápis **jiným** počtem bitů
- Je možné automaticky generovat parser
- Příklady použití: H.323, X.509

Základem definice protokolů H.323 je metoda nazývaná Abstract Syntax Notation. Je to metoda, jak definovat nějakou datovou strukturu nebo obsah jednotek dat protokolu pomocí formální definice využívající mnemonické identifikátory (včetně enumerace, sady pojmenovaných hodnot) a speciální konstrukty pro vyjádření posloupnosti (záznamu), polí, výběru z několika alternativ (něco jako konstrukt case nebo select z programovacích jazyků) apod. Jako prostředek na formální popis je to velmi užitečný nástroj.

Problémem je jeho defaultní implementace. Jak už jsem naznačil, každá hodnota se zapíše pouze tolika bity, kolik je pro danou položku nutné. Autoři ale počítali s tím, že protokoly se vyvíjejí a do návrhu zabudovali obecný mechanismus, jak rozpoznat, zda hodnota používá původní návrh nebo jeho rozšíření. Před vlastní hodnotou je tedy jeden bit, který říká, zda je hodnota rozšířená nebo ne, a tím vlastně určuje, kolik bitů ve skutečnosti zabírá... Implementace takového kódování je extrémně složitá a řeší se nákupem knihovny, která z textového zápisu v ASN.1 vytvoří kód, jenž zápis a čtení realizuje.

Kromě této extrémně úsporné varianty zápisu hodnot existují i varianty, které se alespoň drží hranice bajtů, takže jejich dekódování je o řád jednodušší.

### Session Initiation Protocol

- Náhrada složitého H.323 jednodušším protokolem
- RFC 3261, port TCP i UDP 5060
- Architektura protokolu se podobá HTTP, informace se přenášejí ve formě hlaviček
- Neřeší vlastní přenos dat (obvykle používá RTP/RTCP)
- Řeší jen signalizaci (vyhledání partnera a navázání spojení)
- Dohodu o parametrech datových kanálů obvykle řeší SDP (Session Description Protocol, RFC 8866), jeho data se přenášejí zabalená do těla SIP zpráv
- Koncový uzel se může registrovat u registrátora, tím lze uskutečnit propojení na běžnou telefonní síť

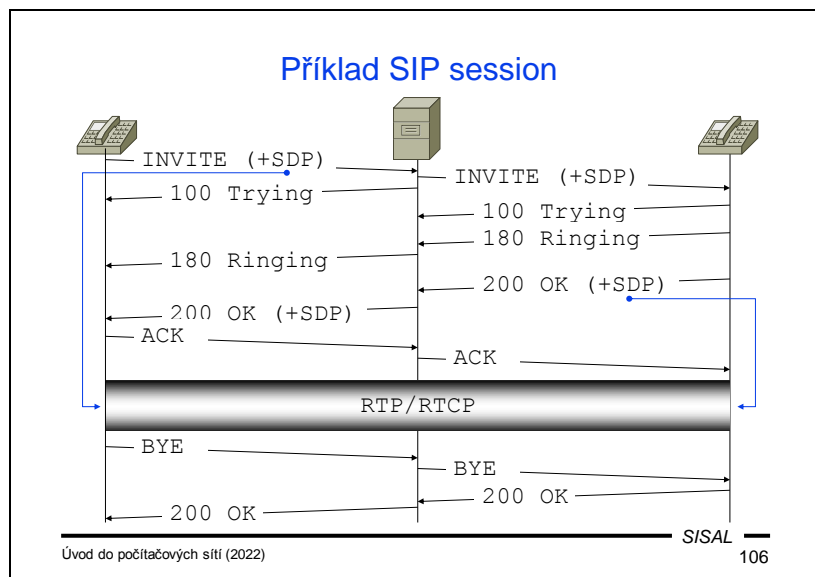
Session Initiation Protocol je modernější řešení pro úkoly související s navazováním spojení a domluvě o vlastnostech zařízení. Na rozdíl od protokolů rodiny H.323 je to textový protokol se strukturou zpráv, která se podobá HTTP, na rozdíl od něj se ale může provozovat nad TCP i UDP. Výhodou tak je, že zprávy protokolu jsou volně čitelné. Tento protokol dnes postupně vytlačuje H.323 a stává se standardem pro budování telefonní sítě v nově projektovaných budovách.

Zajímavostí je, že tento protokol jako jeden z prvních už v iniciálním návrhu zavádí pojem **proxy**, tedy článku komunikačního řetězu, který usnadňuje komunikaci přes hranice různých sítí, včetně privátních. Podobně jako u SMTP se během přenosu do zprávy vkládají hlavičky, které zaznamenávají cestu (tady se jmenují **Via** a **Record-Route**) a podle nichž může protistrana a všechny uzly po cestě správně směřovat odpověď. Stejně tak jsou součástí hlaviček údaje jako identifikátor hovoru, volající i volané SIP URL (obvykle číslo plus doména) aj.

Samotný SIP řeší pouze problém vyhledání cíle, nalezení cesty a navazování spojení. Další stupeň, tedy dohodu o vlastnostech zařízení a parametrech datových kanálů, řeší Session Description Protocol. Je to opět textový protokol skládající se z řádek formátu *keyword=value* a je přenášen pomocí zpráv SIP protokolu. SIP zpráva, která nese SDP informaci, má jako MIME typ uveden **application/sdp** a v těle zprávy je SDP blok.

Jakmile se protistrany dohodnou na podobě datových kanálů a otevřou je, začnou po nich posílat audio (a případně i video) data, a to pomocí protokolů RTP/RTCP.





#### Ukázka SIP hovoru:

- Volající zařízení vyšle příkaz **INVITE**, který obsahuje volané URL a zároveň nabídku datových kanálů jako **SDP** zprávu.
- Příkaz dorazí na nejbližší proxy, která začne řešit nalezení cíle. Podle své konfigurace a podle cílového URL najde další uzel v cestě k volanému zařízení. My si pro jednoduchost tento úsek zkrátíme a představíme si situaci, kdy za proxy už je přímo cílová síť. V tom případě proxy odešle požadavek cílovému zařízení, ale zkontroluje přitom obsah **SDP** a upraví ho tak, aby byl pro protistranu použitelný – de facto v něm provede úpravy odpovídající funkci překladu adres (**NAT**).
- Zároveň odešle volajícímu zařízení dočasnou odpověď **100 Trying**.
- Jakmile volané zařízení zpracuje požadavek, odešle rovněž dočasnou odpověď **100 Trying**. Tato odpověď má charakter *node-to-node* a proxy ji tudíž nepřeпоšle volajícímu zařízení.
- Cílové zařízení začne vyzvánět a informuje o tom volající zařízení další dočasnou odpovědí **180 Ringing**. Ta má naopak charakter *end-to-end*, takže tu proxy přeпоšle.
- Pokud volaný účastník telefon zvedne, jeho zařízení pošle konečnou odpověď **200 OK**. Tato odpověď zároveň nese **SDP** zprávu s nabídkou datových kanálů na volaném zařízení.
- Jakmile tato zpráva dorazí na proxy, ta opět zkontroluje obsah **SDP**, upraví ho podle potřeb překladu adres a zprávu pošle volajícímu zařízení.
- Pro dokončení úvodní fáze je ještě nutné, aby volající zařízení potvrdilo příjem **SDP** zprávy, a proto celá fáze končí posláním potvrzovací zprávy/příkazu **ACK**.
- Od tohoto okamžiku mohou obě zařízení začít posílat multimediální data v **RTP /RTCP** po datových kanálech, které jim nabídla protistrana.

- Ukončení hovoru se odehraje tak, že libovolné koncové zařízení pošle příkaz BYE a protistrana odpoví 200 OK. Na tuto dvojici správ samozřejmě reaguje i proxy a zruší svoje datové kanály, které otevřela na obě strany.

## Souhrn 5

- K čemu slouží IMAP protokol?
- K čemu slouží HTML?
- Popište co se odehraje v HTTP/1.1 protokolu, pokud uživatel požádá o stránku se dvěma vloženými obrázky.
- Stručně popište účel a vlastnosti nejznámějších metod HTTP.
- K čemu slouží cookies a jaké představují riziko?
- Porovnejte protokoly telnet a SSH.
- Porovnejte protokoly H.323 a SIP.