

# Procesy a věci v počítačové síti

Šárka Vavrečková

Slezská univerzita

Šárka Vavrečková

# Procesy a věci v počítačové síti

Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě

**Procesy a věci v počítačové síti**

**RNDr. Šárka Vavrečková, Ph.D.**

Recenzenti: Doc. RNDr. PaedDr. Hashim Habiballa, Ph.D., PhD.
Doc. RNDr. Petr Bujok, Ph.D.

Vydavatel: Slezská univerzita v Opavě
Na Rybníčku 626/1, 746 01 Opava

Autorská práva: © autor, 2022
Obálka: © Anna Novotná, 2022
Vydáno v Opavě roku 2022
ISBN: 978-80-7510-520-2 (online)
978-80-7510-521-9 (tisk)

Sázeno v systému LaTeX

# Anotace

Hlavním tématem této monografie je komunikace v Internetu věcí. Po stručném uvedení do problematiky počítačových sítí najde čtenář vysvětlení toho, jak funguje Internet věcí, jaké protokoly se u těchto zařízení používají a který hardware a software se pro Internet věcí dá použít. Poslední kapitola je na pomezí teoretické a praktické informatiky: ukazuje, jak využít membránový systém v roli prostředku pro modelování komunikace v síti Internetu věcí tak, aby bylo možné tento prostředek naprogramovat v některém vhodném programovacím jazyce. Popisovaný membránový systém lze využít i pro implementaci membránového firewallu.

# Summary

The main topic of this monograph is communication in the Internet of Things. After a brief introduction to computer networks, the reader will find an explanation of how the Internet of Things works, what protocols are used in these devices, and which hardware and software can be used for the Internet of Things. The last chapter is on the borderline between theoretical and practical computer science: it shows how to use a membrane system as a means to model communication in an IoT network so that this means can be programmed in a suitable programming language. The described membrane system can also be used to implement a membrane firewall.

# Předmluva

Tématem této monografie jsou počítačové sítě se zaměřením na sítě Internet věcí a možnosti využití membránových systémů pro modelování komunikace v síti Internetu věcí. Kniha stojí na pomezí teoretické (membránové systémy) a praktické (Internet věcí) informatiky, pojednání o spojení membránových systémů a počítačových sítí je založeno na několika publikacích autorky z posledních let.

Cílem první kapitoly je ujednotit, shrnout a snad některým čtenářům připomenout pojmy, postupy a metody z oblasti počítačových sítí, na kterých jsou postaveny následující kapitoly. Vedle vlastností sítí jsou zde stručně popsány možnosti komunikace v počítačové síti a její základní parametry jako je spojovost a spolehlivost, také jsou krátce probrány možnosti zabezpečení síťové komunikace.

Druhá kapitola je o protokolech. Po vysvětlení principu spolupráce protokolů a jejich skládání do síťového zásobníku se text zaměřuje na protokol HTTP, na kterém jsou vysvětleny některé aspekty komunikace – různé verze protokolu HTTP jsou pro tento účel dobře použitelné. V této kapitole se seznámíme také s mechanismem REST, který je v oblasti webových aplikací a také sítí Internetu věcí čím dál populárnější. Počítáme zde s kombinací mechanismu REST s protokolem HTTP, třebaže možností je více. Závěr kapitoly je věnován příkladům zachycení stavu komunikace.

V třetí kapitole se zaměříme na Internet věcí. Po vysvětlení specifik sítí Internetu věcí se čtenář seznámí s komunikačními modely pro zařízení Internetu věcí. Následují sekce o protokolovém zásobníku a protokolech typických pro tyto sítě. Stručně je popsána celá řada protokolů na různých vrstvách pro místní sítě a také protokoly pro rozlehlé sítě, podrobněji se věnujeme protokolům MQTT (coby zá-

stupci aplikačních IoT protokolů), ZigBee (pro nižší vrstvy síťového zásobníku) a v návaznosti na předchozí kapitolu využití HTTP REST pro sítě Internetu věcí. Zejména části o MQTT a HTTP REST jsou doprovázeny příklady. Poslední sekce kapitoly doplňuje téma o možnostech řešení Internetu věcí ve vlastní režii.

V poslední kapitole se završuje hlavní téma knihy. Kapitola pojednává o využití membránových systémů pro modelování či zobecnění procesu přenosu dat v síti Internetu věcí. Membránové systémy lze použít jako prostředek na modelování paralelní komunikace v systémech s hierarchickou strukturou. Po seznámení s membránovými systémy je nastíněna struktura celého modelu: systém je rozdělen do tří vrstev, z nichž v horní vrstvě jsou umístěna zařízení Internetu věcí, spodní vrstva je membránový systém zajišťující samotnou komunikaci a prostřední vrstva je řídicí vrstva zprostředkující přenos a transformaci zpráv/objektů mezi horní a spodní vrstvou. V poslední sekci kapitoly je nastíněna možnost využití membránového systému pro implementaci firewallu.

Tato publikace byla podpořena Strukturálními investičními fondy Evropské unie OP VVV, projektem „Zvýšení kvality vzdělávání na Slezské univerzitě v Opavě ve vazbě na potřeby Moravskoslezského kraje“ s registračním číslem CZ.02.2.69/0.0/0.0/18_058/0010238.

Vřelé poděkování patří také členům mé rodiny, kolegyním a kolegům, a v neposlední řadě ochotným recenzentům.

V Opavě 15. září 2022 Šárka Vavrečková

# Obsah

**Předmluva** **v**

**1 Pojmy a standardy k počítačovým sítím** **1**
1.1 Propojení a komunikace . . . . . 1
1.1.1 Jak je síť poskládána . . . . . 1
1.1.2 Komunikace mezi uzly sítě . . . . . 4
1.2 Spolehlivost a bezpečnost . . . . . 5
1.2.1 Navazování spojení a spolehlivost . . . . . 5
1.2.2 Bezpečnost na síti . . . . . 7
1.3 Standardy a standardizační instituce . . . . . 9

**2 Protokoly** **15**
2.1 Proč máme protokoly . . . . . 15
2.2 Referenční model a protokolový zásobník . . . . . 17
2.2.1 Referenční model ISO/OSI . . . . . 17
2.2.2 Spolupráce protokolů . . . . . 18
2.2.3 Protokolové zásobníky . . . . . 22
2.2.4 Síťový model TCP/IP . . . . . 24
2.3 Protokol HTTP . . . . . 25
2.4 Jak na úspornou komunikaci: REST API . . . . . 28
2.5 Stavová komunikace na aplikační vrstvě . . . . . 30

**3 Internet věcí** **35**
3.1 Co je to Internet věcí . . . . . 35
3.2 Komunikace v Internetu věcí . . . . . 36

3.3 Protokolový zásobník pro IoT síť . . . 37
3.3.1 IoT protokoly vyšších vrstev . . . 39
3.3.2 IoT protokoly nižších vrstev pro místní sítě . . . 44
3.3.3 WAN sítě pro IoT . . . 46
3.4 MQTT . . . 50
3.4.1 Témata . . . 51
3.4.2 Relace a zprávy . . . 52
3.4.3 Implementace protokolu MQTT . . . 55
3.5 ZigBee . . . 56
3.5.1 Zařízení v ZigBee síti . . . 56
3.5.2 Protokolový zásobník . . . 58
3.5.3 Rámce a průběh komunikace v síti . . . 60
3.5.4 Napojení na okolní svět . . . 61
3.6 HTTP REST pro IoT . . . 62
3.6.1 HTTP zprávy v mechanismu REST . . . 62
3.6.2 Jak poslat HTTP zprávu . . . 64
3.7 IoT ve vlastní režii . . . 65

**4 Počítačové sítě a membránové systémy 69**
4.1 Membránové systémy . . . 69
4.1.1 Předchozí práce . . . 70
4.1.2 Definice membránového systému . . . 71
4.2 Modelování komunikace v IoT . . . 73
4.2.1 Vrstvený model . . . 73
4.2.2 Membránová vrstva . . . 75
4.2.3 Řídicí vrstva . . . 79
4.3 Membránový firewall . . . 84

**Seznam literatury 87**

**Seznam obrázků 93**

**Seznam zkratek 95**

Kapitola 1

# Pojmy a standardy k počítačovým sítím

V této kapitole si stručně projdeme některé pojmy z oblasti počítačových sítí, které jsou používány v dalších kapitolách. Účelem není sepsat úplný glosář pro oblast počítačových sítí, jde spíše o rychlé uvedení do problematiky a sjednocení terminologie. Text vychází především ze zdroje [46].

## 1.1 Propojení a komunikace

### 1.1.1 Jak je síť poskládána

V síti nejde ani tak o samotné propojení uzlů, ale především o možnost přenosu dat mezi těmito uzly. Propojení (přenosová cesta, spoj) je pouze prostředkem, cílem je *přenos* (transmission): přenos probíhá vždy s využitím některého spoje.

Rozlišujeme přenos[37]

- *simplexní* (simplex) – komunikace probíhá jen v jednom směru,
- *plně duplexní* (full duplex) – komunikace probíhá v obou směrech,
- *poloduplexní* (half duplex) – komunikace sice probíhá v obou směrech, ale ne zároveň (uzly se v komunikaci střídají, nemohou vysílat oba najednou).

Plně duplexní spoj typicky bývá realizován dvěma nebo více simplexními spoji. Ty mohou být i souhrnně v jediném kabelu, například v gigabitovém Ethernetu

s využitím kroucené dvojlinky jsou k dispozici čtyři páry vodičů, z toho dva pro každý směr.[22] U bezdrátových řešení se duplex řeší používáním více frekvenčních pásem a/nebo multiplexováním.

Podle velikosti můžeme sítě rozčlenit do následujících kategorií:[41]

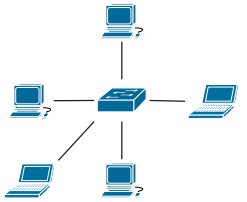
- PAN (Personal Area Network) – malá síť obvykle v rozsahu centimetrů až metrů, typičtí zástupci jsou protokoly NFC a Bluetooth,
- LAN (Local Area Network) – běžná lokální síť rozprostírající se v bytě, domě, případně areálu, typičtí zástupci jsou Ethernet a Wi-fi,
- MAN (Metropolitan Area Network) – síť pokrývající město či souměstí, typický zástupce je WiMAX,
- WAN (Wide Area Network) – rozlehlá síť pokrývající region, stát nebo celý svět, typický zástupce je MPLS, LTE.

Uvedená klasifikace je přibližná, jednotlivé kategorie se značně překrývají. Například Ethernet je typicky LAN technologie, ale ethernetové spoje na optických kabelech se dnes využívají také v MAN a WAN sítích. Bluetooth je sice PAN technologie, ale definuje několik různých tříd určujících napájení a dosah, třída 1 znamená dosah až 100 m.[1]

Pojem *topologie*[41] označuje vyjádření struktury, vztahů mezi entitami. V případě počítačové sítě hovoříme buď o fyzické, nebo o logické topologii. *Fyzická topologie* počítačové sítě popisuje umístění a způsob propojení součástí sítě. Bereme zde v úvahu například to, mezi kterými zařízeními vede kabel, u bezdrátového spoje dosah signálu a asociaci k mezilehlým prvkům. *Logická topologie* určuje komunikační hierarchii. Pokud máme více než jednu podsíť, pak je důležité, které zařízení do které podsítě patří. Zařízení patřící do stejné podsítě mohou navzájem komunikovat jednodušeji, naopak zprávy posílané mezi zařízeními patřícími do různých podsítí lze filtrovat, logovat či jinak zpracovávat, protože procházejí přes některý síťový prvek vyšší úrovně nacházející se mezi podsítěmi. Součástí údajů o logické topologii sítě bývají adresy nebo jiný způsob identifikace zařízení.

Topologií existuje celá řada, pro různé typy sítí bývá vhodná jiná topologie. Zde si představíme několik nejběžnějších, s dalšími se setkáme v kapitole o Internetu věcí.

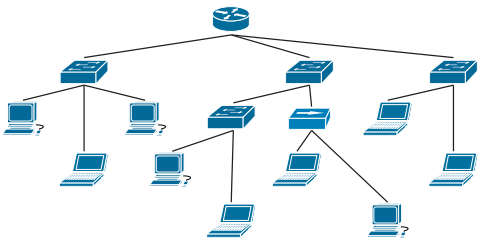
[1] Dosah Bluetooth včetně varianty LE je diskutován např. v [20].



**Topologie hvězda.** V síti je jeden centrální prvek, k němuž jsou připojeny ostatní uzly sítě. Veškerá komunikace probíhá přes tento centrální prvek.

Výhodou je, že pokud je poškozen spoj mezi některým uzlem a centrálním prvkem, síť funguje dál (jen ten jeden uzel nemůže komunikovat). Další výhodou je možnost oddělení komunikačních cest mezi jednotlivými dvojicemi uzlů (oddělení zajišťuje na logické úrovni centrální prvek). Nevýhodou je, že centrální prvek je „úzkým hrdlem“ (je nejvíc vytěžován), také poškození nebo přetížení centrálního prvku znamená kolaps celé sítě.

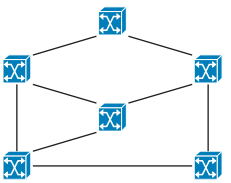
Tato (fyzická) topologie je základem současných lokálních sítí a jako logickou topologii si ji můžeme představit jako připojení zaměstnanců na home office přes VPN koncentrátor do firemní sítě.



**Stromová topologie.** Je to hierarchické propojení sítí s topologií hvězda. Zařízení jsou rozdělena do úrovní a počítá se s oddělením provozu v jednotlivých částech sítě.

Výhodou je usnadnění správy sítě (síť je přehledná a mezi každými dvěma uzly vede právě jedna cesta), nevýhodou je snížená odolnost sítě – nemáme redundantní (tedy záložní) cesty.

Se stromovou (fyzickou) topologií se dnes běžně setkáváme ve středních a velkých lokálních sítích, je základem *strukturované kabeláže*. Typickým zástupcem je Ethernet. Nedostatečnou redundanci cest řeší například v Ethernetu protokol STP.

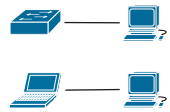


**Topologie mesh (smíšená).** Pro tuto topologii jsou typické *redundantní spoje* (mezi dvěma uzly může vést více než jedna cesta). Uzly v síti jsou víceméně rovnocenné. Pokud je vzájemné propojení prvků úplné (jsou propojeny stylem „každý s každým“), hovoříme o topologii *full mesh*.

Výhodou je vysoká odolnost proti výpadku, nevýhodou relativně vysoká spotřeba kabeláže v případě využití pro fyzickou topologii.

Tento typ (fyzické) topologie je běžný v současných WAN sítích. V nich je třeba zajistit spolehlivost cest, tedy redundance je nezbytná.

S logickou topologií typu full mesh se běžně setkáváme například u směrovacích protokolů (pro účely směrování se mezilehlá zařízení potřebují navzájem „znát“, aby dokázala vybrat nejvhodnější cestu).



**Point-to-point spoj.** Tuto topologii jako fyzickou najdeme především u protokolu NFC (například když telefon naváže spojení s platebním terminálem) nebo Bluetooth.

Jako logická topologie je v současné době hodně používaná – pokud například u hvězdy použijeme jako centrální prvek takové zařízení, které dokáže na logické úrovni oddělit komunikaci s jednotlivými připojenými zařízeními (tj. switch), pak je logickou topologií množina point-to-point spojů, které se navzájem neruší.[41]

### 1.1.2 Komunikace mezi uzly sítě

Na Internetu se nejběžněji setkáváme s komunikační architekturou *klient-server*. V síti existují dva typy uzlů – serverová zařízení, která poskytují služby, a klientská zařízení, která tyto služby využívají. Klient pošle žádost serveru, server pošle odpověď klientovi. Z pohledu každé konkrétní služby se klient od serveru liší, a to především softwarově, tedy role jsou striktně určeny.[41]

Ovšem zařízení, které vzhledem k jedné službě (či protokolu) plní roli serveru, se vzhledem k jiné službě/protokolu může stát klientem. Například webový server je z pohledu protokolu HTTP serverem, ale vzhledem k logování (Syslog) může být klientem (hlásí Syslog serveru svůj stav).

Architektura typu klient-server je výhodná i z bezpečnostního hlediska. Firewall na hranici sítě neustále sleduje probíhající komunikaci: pokud klient z vnitřní sítě odešle žádost serveru na Internetu a server následně pošle do vnitřní sítě odpověď, je to v pořádku. Ale pokud přijde z Internetu žádost, je to považováno za pokus o průnik do sítě, protože na Internetu se počítá jen se servery, které žádosti neposílají.

Tento princip v některých případech může představovat problém, například pokud chceme zprovoznit chat mezi dvěma klienty bez použití serveru, nebo za určitých okolností při posílání dat mezi zařízeními Internetu věcí.
Jinou komunikační architekturu představuje *peer-to-peer komunikace*. Také sice hovoříme o roli klienta a serveru, ale tyto role nejsou striktně vymezeny a každé zařízení může vzhledem k témuž protokolu plnit obě funkce. Pojem, který vcelku vystihuje peer-to-peer technologie, je decentralizace. Nejběžněji se s touto komunikační architekturou setkáme zde:

- SMB, Samba, CIFS pro sdílení dat a služeb (typicky tiskových) v místní síti: tento protokol a jeho varianty najdeme v různých operačních systémech, například když tiskneme přes síť,

- CDN sítě, což jsou sítě pro sdílení obsahu,
- Bitcoin (popřípadě jiné kryptoměny) je peer-to-peer platební síť fungující na principu blockchainu, což je druh decentralizované databáze využívaný kromě kryptoměn také pro jiné účely.[2]

CDN (Content Delivery Network) slouží pro sdílení jakéhokoliv obsahu. Může se jednat o multimediální obsah (např. streamování videa) nebo třeba o aktualizace (CDN využívají i běžné operační systémy včetně Windows pro rozesílání aktualizačních balíčků) či o přístup k sociálním sítím. Princip CDN je jednoduchý: obsah uložený na původním serveru je replikován (zrcadlen) do sítě serverů po celém světě, takže je možno využít tu kopii, která je nejblíž. Pokud zařízení odešle žádost na server, tato žádost je mechanismem DNS přesměrována na vhodné zrcadlo v CDN síti.[17]

## 1.2 Spolehlivost a bezpečnost

### 1.2.1 Navazování spojení a spolehlivost

V počítačové síti probíhá přenos dat jedním ze dvou základních způsobů:[22]

- *spojovaný přenos* (connection oriented) – dva uzly sítě nejdřív navážou spojení, tedy vytvoří se relace, a pak v rámci relace probíhá přenos dat,
- *nespojovaný přenos* (connectionless) – žádné spojení se nenavazuje, odesílatel jednoduše sestaví PDU příslušného protokolu a odešle.

Spojovaný přenos má výhodu ve větší kontrole nad přenášenými daty, ale na druhou stranu je třeba přes síť dostat mnohem více metadat: samotné navazování spojení vyžaduje, aby se obě strany navzájem představily a dohodly na parametrech spojení, při posílání dat v rámci relace obvykle probíhá potvrzování a záhlaví spojových protokolů typicky bývá rozsáhlejší.[8]

Nespojovaný přenos to má s vlastnostmi přesně opačně: je úspornější co se týče množství přenesených (meta)dat, ale sám o sobě není moc spolehlivý. Odesílatel prostě odešle PDU a už se nestará.

[2] Se širokým využitím blockchainu se setkáme zejména v projektu Ethereum. Kromě využití pro účely kryptoměny (podobně jako Bitcoin) zde najdeme projekty Decentralized Finance (DeFi), Non-fungible Tokens (NFT), Decentralized Identity, Decentralized Science a další. Informace jsou na webu https://ethereum.org/en/ [cit. 2022-09-13].

(Ne)spojování komunikace se nejčastěji řeší na transportní vrstvě. Tam pracují především dva základní protokoly – TCP, který nabízí spojovanou službu přenosu dat, a UDP, který používáme pro nespojovaný přenos. Až na výjimky jsou protokoly jiných vrstev nespojované, protože pokud potřebují komunikovat s navázáním spojení, spolehnou se na spojovaný transportní protokol.[21]

**Další informace**

Na transportní vrstvě si ve skutečnosti konkuruje poněkud více protokolů než jen TCP a UDP. Podrobné informace najdeme v různých zdrojích, zajímavé porovnání výkonnostních parametrů protokolů TCP, UDP, SCTP a DCCP při použití nad 4G sítí najdeme například v článku
https://www.sciencedirect.com/science/article/pii/S1877050917311754 [cit. 2022-09-12].

Další důležitou vlastností protokolů je (ne)spolehlivost. Protokoly tedy můžeme rozlišit na *spolehlivé* (reliable) a *nespolehlivé* (unreliable), přičemž nespolehlivost neznamená, že bychom se na daný protokol nemohli spolehnout: nespolehlivé protokoly fungují stylem „best effort“, tedy doručení není zaručeno, ale protokol udělá vše pro to, aby se to povedlo... [22]

Spolehlivost se většinou (ne vždy) týká spojových protokolů jako je například TCP, zajišťuje se typicky tak, že provoz je příjemcem potvrzován. Nemusí být nutně potvrzována každá PDU, ale potvrzení bývá očekáváno po odeslání určitého předem dohodnutého množství dat.

Ne vždy je spolehlivost nutná. Například u multimediálních přenosů celkem nevadí, když se sem tam nějaký paket ztratí, lidské oko a ucho to nepozná. Pokud naopak potřebujeme spolehlivý přenos, tak není nutné, aby na něm spolupracovaly průřezem přes celý síťový zásobník pouze spolehlivé protokoly, stačí, když je spolehlivý jeden. Například protokol IP je nespolehlivý (nepotvrzuje, nepožaduje znovuposlání ztracených paketů), ale vůbec to nevadí, pokud na transportní vrstvě použijeme spolehlivý protokol (TCP).

Protokoly TCP a UDP jsou na transportní vrstvě něco jako zrcadlové obrazy. TCP je spojovaný spolehlivý protokol generující svými záhlavími, potvrzováním a udržováním spojení velký provoz, UDP nespojovaný nespolehlivý protokol s krátkými záhlavími, pro linku velmi úsporný. Pokud potřebujeme „něco mezi“, dají se použít například protokoly DCCP (spojovaný nespolehlivý, ale s kontrolou chyb) nebo SCTP (v základu nespojovaný, ale dokáže vytvořit něco na způsob P2P spo-

jení s více streamy). Oba tyto zmíněné protokoly se dají s výhodou použít například pro multimediální přenosy, VoIP apod., ale také v sítích Internetu věcí.[20]

**Další informace**

Zajímavý popis vlastností, výhod a nevýhod protokolu TCP a srovnání s alternativami – MPTCP, SCTP, DCCP a QUIC – najdeme například na webu http://intronetworks.cs.luc.edu/current2/uhtml/tcpB.html [cit. 2022-09-12].

### 1.2.2 Bezpečnost na síti

Pod pojem bezpečnost sítě toho lze zahrnout velmi mnoho od speciálních hardwarových zařízení (firewall, IDS atd.) přes kryptografii až k ochraně před sociálním inženýrstvím. Pro samotnou ochranu provozu používáme metody kryptografie, ať už se jedná o zabezpečenou autentizaci, šifrování provozu, digitální podepisování/certifikáty nebo vytváření zabezpečených tunelů mezi dvěma či více sítěmi či zařízením a sítí.

V následující kapitole srovnáváme protokoly Internetu věcí podle různých kritérií, je tam zmíněno i šifrování provozu. Proto se zde alespoň stručně podíváme na to, jaké jsou možnosti.

**IKE.** Protokol IKE (Internet Key Exchange) slouží k dohodnutí parametrů pro navázání zabezpečeného připojení. Pomocí IKE si obě komunikující strany navzájem sdělí potřebné parametry, podporované protokoly a jejich verze, klíče nebo informace k jejich vypočtení oběma stranami. Mechanismus IKE je používán především v IPSec.[41]

**IPSec.** Protože se ukázalo, že velkou nevýhodou protokolu IPv4 je zcela chybějící jakéhokoliv zabezpečení, při vývoji IPv6 se myslelo i na to – součástí specifikace IPv6 se stalo také zabezpečení komunikace na síťové vrstvě, a protože se očekává ještě dlouholetá koexistence verzí 4 a 6 protokolu IP, příslušná specifikace se připravila i jako rozšíření pro protokol IPv4. Tato specifikace se nazývá IPSec (IP Security) a najdeme ji v sadě RFC dokumentů, je to otevřený standard.[21]

**Další informace**

Základním dokumentem pro bezpečnost na síťové vrstvě je RFC 6071 *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap*, architektura celého

řešení je nastíněna v RFC 4301 *Security Architecture for the Internet Protocol*. K dokumentům se dostaneme tak jako ke kterýmkoliv jiným RFC dokumentům, například na portálu https://www.rfc-editor.org [cit. 2022-09-13].

IPSec zajišťuje bezpečnou autentizaci, integritu (ověření, zda data nebyla cestou poškozena nebo pozměněna), umí šifrovat komunikaci, dokáže také vytvářet zabezpečené tunely VPN mezi dvěma sítěmi přes Internet (VPN typu site-to-site, například pokud potřebujeme propojit dvě pobočky) nebo mezi klientským zařízením a sítí taktéž před nezabezpečenou síť (VPN typu remote access, například zaměstnanec na home office či pracovní cestě, který se potřebuje bezpečně dostat do firemní sítě). Protože pracuje na síťové vrstvě, je pro transportní a aplikační protokoly „neviditelný".[41] Je to poměrně komplexní mechanismus, který si kromě různých parametrů spojení vede také databázi bezpečnostních politik, v níž má stanoveny bezpečnostní politiky uplatňované na IP pakety – pravidla, jak zacházet s IP pakety podle obsahu jejich záhlaví (zahodit, propustit, šifrovat apod.).[37]

Protože IPSec pracuje na síťové vrstvě, dokáže šifrovat také záhlaví, která přidaly protokoly vyšších vrstev.[37]

**SSL, TLS.** Zatímco IPSec se používá na síťové vrstvě, SSL a TLS patří na vyšší vrstvy. Za SSL (Secure Sockets Layer) stojí společnost Netscape Communications a tento protokol je postupně z Internetu vytěsňován, jeho otevřený potomek TLS (Transport Layer Security, RFC 5246 pro verzi TLS 1.2, RFC 8446 pro verzi 1.3) je naopak čím dál běžnější. Proto budeme dále zmiňovat spíše protokol TLS, třebaže tyto protokoly jsou ve svých vlastnostech a možnostech využití podobné.

TLS zajišťuje autentizaci, integritu a šifrování, také je možno vytvořit TLS tunel pro šifrovanou komunikaci. Aby bylo možné využít TLS, musí se na transportní vrstvě navázat spojení – tj. na transportní vrstvě použijeme obvykle protokol TCP.[41]

Jak je výše uvedeno, TLS pracuje na vyšších vrstvách. Proto není pro aplikace zcela transparentní a mohou ho používat pouze ty transportní a aplikační protokoly a také aplikace, které to „umí". Podporu TLS bez problémů najdeme například ve webových prohlížečích a e-mailových klientech, někdy také v chatovacích programech.[41]

Každý aplikační protokol podporující TLS má na transportní vrstvě určité číslo portu, a to každý jiné. Například zatímco HTTP nešifrovaný obvykle na straně serveru používá port číslo 80, HTTPS (což je HTTP + TLS) pracuje na portu 443.

Protokol SMTP (který používáme, když odesíláme e-mail v e-mailovém klientovi) používá na straně serveru port 25, ale když se zkombinuje s TLS, přechází na port 465.[37]

**DTLS.** Zatímco TLS lze použít pro zabezpečení spojovaného provozu (TCP spojení), pro zabezpečení nespojovaného provozu posílaného přes transportní protokoly UDP, SCTP a další je možné využít DTLS. Odlišnosti mezi TLS a DTLS vycházejí z (ne)spojovosti komunikace. TLS to má bezpochyby snadnější, protože jak během dojednávání parametrů, tak i při přenosu dat, může využít existující TCP spojení. Naproti tomu DTLS je nucen si příslušné služby (sledování stavu spoje, detekce chyb, sledování kolizí a změn pořadí paketů atd.) zajišťovat sám ve vlastní režii.[20]

## 1.3 Standardy a standardizační instituce

Jakékoliv zařízení v počítačové síti musí pracovat takovým způsobem, aby se dokázalo „domluvit“ s jinými zařízeními v síti. Pokud se posílá zpráva, musí být nejen srozumitelná cílovému zařízení, ale také všechna mezilehlá zařízení musí být schopna si s touto zprávou poradit, doručit ji. Tato zařízení tedy musí splňovat určité standardy.

**Definice (Standard, norma [37, 41])**
Standard (v oblasti technologií) je požadavek na splnění určitých konkrétních vlastností pro určitý typ hardwaru, softwaru apod. Je to dokument popisující požadavky, specifikace, návody a popisy pro daný produkt, proces či službu. Rozlišujeme standardy

- *normativní (de iure, podle zákona)* – jejich dodržování je víceméně vyžadováno, obvykle je stanoví příslušná státní instituce,
- *popisné (de facto, doporučení, recommendation)* – jejich dodržování sice není striktně vyžadováno, ale je v zájmu výrobce.

V oblasti technologií obvykle používáme pojem *norma* pro normativní standardy, chybu ovšem neuděláme, když budeme hovořit vždy o standardu.

Následuje přehled vybraných standardizačních organizací a institucí, které vydávají standardy související s počítačovými sítěmi a technologiemi obecně. Informace o dále uvedených organizacích pocházejí z jejich webových stránek.

**Úřad pro technickou normalizaci, metrologii a státní zkušebnictví** je národním standardizačním orgánem pro Českou republiku, zabývá se tvorbou českých norem. Normy vydané tímto ústavem poznáme podle toho, že začínají písmeny ČSN.

**Další informace**

Normy vydané Úřadem pro technickou normalizaci, metrologii a státní zkušebnictví jsou k nahlédnutí na http://www.unmz.cz/urad/csn-online [cit. 2022-04-08] (jen náhledy, za celé znění se platí).

**Poznámka**

V souvislosti s technickými normami se často objevuje pojem *homologace* – otestování vlastností výrobku co se týče bezproblémovosti použití pro daný účel, například zda elektronický výrobek lze používat v rámci infrastruktury dané země (energetická soustava, mobilní sítě apod.). Nejde jen o funkčnost daného výrobku, ale také o jeho případný negativní vliv na infrastrukturu a další výrobky, ve skutečnosti jde o soulad s příslušnými normami.

U elektronických výrobků je vyžadována shoda s právními normami týkajícími se technických parametrů, a to ve formě *Prohlášení o shodě* – CE (Conformity Declaration). Toto prohlášení o shodě musí zajistit výrobce zařízení, a to typicky u některé autorizované organizace provádějící testování. Prohlášení je platné v rámci celé Evropské unie. Problémy nastávají u některých výrobků dovážených z mimoevropských zemí jako je Čína (pokud v čínském e-shopu koupíme zařízení, které nemá CE, pak takové zařízení nemůžeme u nás používat).

Jednou z autorizovaných organizací je i Elektrotechnický zkušební ústav, na jeho stránkách můžeme najít další informace: http://ezu.cz/produkty/ce-certifikat/ [cit. 2022-04-08].

**ETSI** (European Telecommunications Standard Institute) je evropská organizace, ale ve skutečnosti jsou její členové ze všech obydlených kontinentů (státy, významní výrobci komunikačních zařízení, poskytovatelé síťových služeb, výzkumné organizace, atd.).

Rozlišuje se několik typů ETSI standardů – například EN (European Standard), ES (ETSI Standard), TS (ETSI Technical Specification) a další. K nejzná-

mějším patří standardy související s mobilními sítěmi (především GSM, 3G, 4G sítě), chytrými sítěmi, komunikací machine-to-machine, ICT ve zdravotnictví, atd. ETSI standardy jsou dostupné volně bez poplatků.

**Další informace**

http://www.etsi.org/technologies-clusters/technologies [cit. 2022-04-08]

**ITU** (International Telecommunications Union) je celosvětová organizace pro informační a komunikační technologie. Je součástí OSN a sídlí ve Švýcarsku.

*ITU-T* je součást ITU pro standardizaci telekomunikací, její činnost souvisí i s počítačovými sítěmi. Členy s hlasovacím právem jsou zainteresované země, členem bez hlasovacího práva může být kterákoliv organizace. Členství je placené.

ITU-T se dále člení do *studijních skupin* (SG, Study Group), které mají každá své vlastní zaměření. Například SG13 v poslední době vydala několik standardů o cloud computingu, SG16 se zabývá multimédii (přenos videa, obrázky, zvuk apod.), SG17 bezpečností, SG20 chytrými sítěmi (Internet of Things,...). Standardy vytvořené ITU-T jsou otevřené, volně dostupné.

Se standardy ITU-T se setkáváme poměrně běžně: například stojí za protokolem H.323 používaným v multimediálních přenosech a internetové telefonii, protokoly G.992.1 a G.992.2 pro přístupové sítě ADSL, standardem X.509 používaným při šifrování a dalšími.

**Další informace**

http://www.itu.int/en/ITU-T/Pages/default.aspx, http://www.itu.int/pub/R-REC [cit. 2022-04-08]

**ISO** (International Organization for Standardization) je nezávislá organizace vydávající standardy pro komunikační technologie. Jejími členy jsou standardizační instituce z různých zemí (každá země tam má jedno zastoupení), naopak organizace ISO je členem ITU. Sídlo ISO je ve Švýcarsku.

**Další informace**

http://www.iso.org/iso/home/standards_development/list_of_iso_technical_committees.htm [cit. 2022-04-08]

Nejznámějším ISO standardem je referenční model ISO/OSI (standard ISO 7498 a hodně dalších s ním souvisejících), dále se ISO zabývá sítěmi senzorů, vzdáleným přístupem, UPnP, webovými službami a dalšími tématy.

**IEEE** (Institute of Electrical and Electronics Engineers, čteme anglicky „I-triple-E“, tedy [aj tripl i:]) je největší světová organizace sdružující odborníky z oblasti elektroniky, elektrotechniky, informatiky a souvisejících oborů. Nezabývá se jen standardy, ale také pořádá různá setkání, konference, vzdělávací akce, vydává odborné časopisy a vyvíjí další aktivity. Sídlo má v USA. Jednou z aktivních sekcí IEEE je i ta česká.

**Další informace**
http://www.ieee.org [cit. 2022-04-08]

Sdružení IEEE stojí především za skupinou standardů IEEE 802 zejména pro lokální sítě a M2M komunikaci, do které patří například IEEE 802.11 (Wi-fi) nebo IEEE 802.3 pro Ethernet. V kapitole Internet věcí se setkáme také se standardem IEEE 802.15.1 (Bluetooth) a IEEE 802.15.4. Přístup k plnému znění standardů je placený.

**IEC** (International Electrotechnical Commission) se z oblasti počítačových sítí zabývá především standardy pro elektrická a elektronická zařízení. Spolupracuje především s organizací ISO a hodně standardů nese označení „ISO/IEC“, společný technický výbor je označen ISO/IEC JTC 1.

**Další informace**
http://www.iec.ch/ [cit. 2022-04-08]

Známý standard je například ISO/IEC 27000:2016[3] stanovující pravidla pro management informační bezpečnosti v organizacích (technická opatření, personální opatření, bezpečnostní politiky apod.). Často se setkáváme se stručnějším označením ISO 27000 (nebo ISO 27k), na standardu spolupracuje také IEC v rámci výboru JTC 1.

**IETF** (The Internet Engineering Task Force) se zabývá především standardy souvisejícími s Internetem, včetně například protokolů TCP/IP. Velmi úzce spolu-

[3] Základní informace jsou na https://www.iso.org/standard/66435.html.

pracuje s dalšími organizacemi, například IANA (The Internet Assigned Numbers Authority), IAB (Internet Architecture Board, Rada pro architekturu Internetu), W3C (World Wide Web Consortium, standardy pro web) a dalšími. Ve skutečnosti se jedná o jednu z pracovních skupin organizace ISOC (Internet Society), zároveň s IAB a dalšími.

Standardy vydané IETF obvykle začínají zkratkou RFC (Request for Comments) a následuje číslo. Pokud je třeba standard aktualizovat, nemění se původní znění, ale vytvoří se RFC dokument (obsahující popis standardu) s novým číslem. Například pro otevřený směrovací protokol OSPF bylo takto postupně vytvořeno několik dokumentů (verzí), z nichž jsou momentálně aktuální RFC 2328 (OSPF verze 2) a RFC 5340 (OSPF verze 3).

**Další informace**

Oficiální stránky organizace jsou na https://www.ietf.org/, další informace najdeme na stránce mateřské organizace ISOC: https://www.internetsociety.org/.

Všechny dokumenty RFC jsou volně dostupné, hlavní místo přístupu je https://tools.ietf.org/html/. [cit. 2022-04-08]

**TIA a EIA** (Telecommunication Industries Association, Electronic Industries Alliance) jsou americké organizace zaměřující se především na fyzickou úroveň komunikace mezi zařízeními.

**Další informace**

- http://www.tiaonline.org/
- https://www.eia.gov/about/eia_standards.cfm [cit. 2022-04-08]

TIA se zabývá standardy pro nejrůznější typy kabelů a konektorů, připojení antén, mobilní sítě, ICT ve zdravotnictví, chytré sítě. Existují standardy vzniklé při spolupráci těchto společností, například TIA/EIA 568 (kroucená dvojlinka pro Ethernet).

**OASIS Open** je nezisková organizace, která stojí za řadou otevřených standardů zejména z oblasti Internetu věcí, kyberbezpečnosti, blockchainu a dalších. Hodně si zakládá na otevřenosti své činnosti (všechny standardy najdeme na webu organizace

v menu *Standards*) a spolupracuje s řadou velkých technologických společností. Se standardy OASIS Open se setkáme v kapitole Internet věcí.

**Další informace**

https://www.oasis-open.org/ [cit. 2022-04-08]

Kapitola 2

# Protokoly

V počítačové síti můžeme propojit i poměrně hodně odlišná zařízení – nejde jen o to, že na jednom počítači běží Windows v určité verzi, na dalším Windows v jiné verzi, pak je tu Linux, Apple MacOS, server se Solarisem apod., na aktivních síťových prvcích může být Linux, Cisco IOS, ArubaOS, Junos OS nebo něco úplně jiného... V této kapitole se budeme zabývat tím, jak se různá zařízení v síti mezi sebou domluví.

## 2.1 Proč máme protokoly

Ve světě elektroniky se nedorozumíváme přirozeným jazykem, ale pomocí *komunikačních protokolů*. Protokol určuje, jakým způsobem má komunikace začít, jak se dohodnout na parametrech komunikace, jak sdělit druhé straně určitý typ informace, jak má druhá strana potvrdit příjem nebo sdělit, že je třeba přenos opakovat, jak se komunikace ukončuje, atd.

**Definice (Protokol a jeho implementace [21, 37])**
*Protokol* je konvence, podle které probíhá určitý typ (většinou elektronické) komunikace mezi dvěma partnery v komunikaci rovnocennými. Definuje pravidla určující syntaxi (jak jsou které signály poskládány), sémantiku (význam) a postupy pro danou komunikaci relevantní.

Protokol je pouze předpis, který je třeba implementovat, aby mohl být v praxi používán. *Implementace* může být softwarová, hardwarová nebo kombinace softwarové a hardwarové.

Například protokol HTTP (kromě jiného) určuje, jak se má „bavit“ webový prohlížeč s WWW serverem. Je implementován (nasazen) v operačním systému na počítači, u kterého sedíme, a taky na druhé straně – v operačním systému WWW serveru, se kterým komunikujeme.

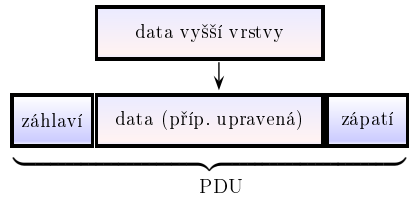
**Poznámka**

Pro návrh protokolu platí jedno důležité pravidlo (vlastně je zachováváno i jinde, například u systému UNIX): protokol by měl být jednoduchý, krátký, jednoznačně implementovatelný. Neměl by být moc složitý, protože čím větší složitost, tím větší pravděpodobnost chyb. Proto žádný protokol není moc univerzální – umí jednu konkrétní věc, a umí ji dobře. V angličtině se to vyjadřuje zkratkou KISS (Keep it Simple, Stupid – ať je to jednoduché, hloupé)[23].

Aby tento princip mohl být dodržován, existují určité konvence pro spolupráci protokolů: to, co protokol neumí, předá ke zpracování jinému protokolu. Takže důležité je nejen to, co protokol umí, ale také to, jak dokáže spolupracovat s jinými protokoly.

Princip protokolů není ani zdaleka jen záležitostí počítačových sítí – protokoly se používají v telekomunikacích, spotřební elektronice, strojírenství, ale například i „uvnitř“ počítače. Každý se setkal třeba s protokolem USB, SATA,...

*Protokolová datová jednotka*[21] (PDU, Protocol Data Unit) je sekvence dat opatřená metadaty (informacemi o datech) vztahující se ke konkrétnímu protokolu.




Obrázek 2.1: Protokolová datová jednotka

Protokol obdrží data, podle potřeby je určitým způsobem zpracuje (strukturuje, rozdělí na menší části, zašifruje, komprimuje, přeloží, určí adresu příjemce apod.) a přidá před ně *záhlaví* (header) se souvztažnými informacemi (délka dat, použitý šifrovací algoritmus, adresa odesílatele a příjemce, atd.). Některé protokoly také přidávají za data *zápatí* (trailer) obsahující například kontrolní součet. Data přenášená v datové jednotce taky nazýváme *payload*.

Pojem „protokolová datová jednotka“ je obecný, použitelný pro jakýkoliv protokol. Ovšem existují názvy PDU specifické pro konkrétní protokoly. Například pojem „paket“ označuje protokolovou datovou jednotku vytvořenou protokolem IP, „segment“ je vytvořen protokolem TCP, atd.

*Otevřený protokol* je protokol zcela volně dostupný, naopak *proprietární protokol* je takový protokol, jehož specifikace není nikde zveřejněna a jeho tvůrce si ji buď nechává jen pro sebe nebo licencuje vybraným obchodním partnerům za poplatek[41].

O rozšířenosti volných specifikací mluví například to, že momentálně je na routerech nejběžnějším směrovacím protokolem otevřený protokol OSPF (tedy kromě velkých páteřních sítí).[19]

Pokud síťové zařízení podporuje pro určitou funkci proprietární protokol, obvykle pro danou funkci taky podporuje některý alternativní protokol s dostupnější specifikací, aby si mohlo „popovídat“ se zařízeními jiných výrobců.

## 2.2 Referenční model a protokolový zásobník

### 2.2.1 Referenční model ISO/OSI

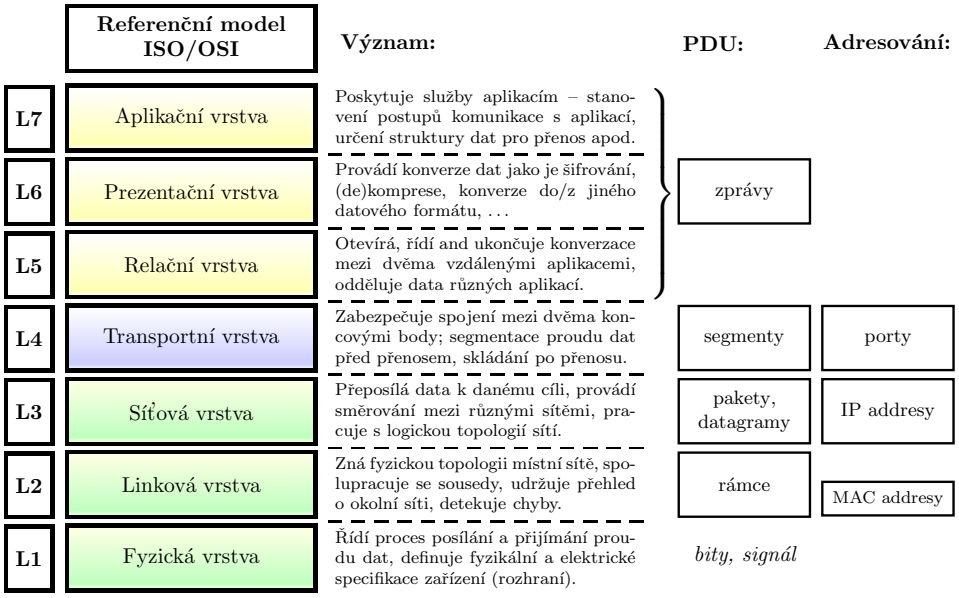
K nejdůležitějším standardům z oblasti počítačových sítí patří skupina standardů popisujících *referenční model OSI* (Open Systems Interconnection) publikovaných organizací ISO, proto se označuje *RM ISO/OSI*. Původní označení standardu je ISO/IEC 7498-1, dnes se jedná o standard ITU-T X.200.[37]

**Další informace**

Dnes je standard ITU-T X.200 dostupný na webu http://www.itu.int/rec/T-REC-X.200-199407-I [cit. 2022-05-10].

OSI definuje sedm vrstev. Každá vrstva plní v komunikaci přes počítačovou síť konkrétní roli, je přesně stanoveno, co se na dané vrstvě může dít, jak má probíhat komunikace v rámci vrstvy a komunikace mezi sousedními vrstvami – jedná se tedy o *konceptuální model* (tj. popisuje logiku návrhu, vztahy mezi součástmi).

Pořadí vrstev a stručný popis najdeme na obrázku 2.2. Každý protokol (vlastně skoro každý) je zařazen vždy na konkrétní vrstvu, čímž je určeno, čím se zabývá a s jakými typy protokolů obvykle komunikuje, protože referenční model určuje i rozhraní mezi protokoly.




Obrázek 2.2: Referenční model ISO/OSI (podle [21])

Při odesílání dat se provádí jejich *enkapsulace* (zabalení, zapouzdření, encapsulation) do PDU a při přijetí *dekapsulace* (rozbalení, decapsulation). To, co je pro nadřízenou vrstvu její vlastní PDU, to je pro jí podřízenou vrstvu pouze sekvence dat, ze kterých vytvoří vlastní PDU.[21].

### 2.2.2 Spolupráce protokolů

Víme, že protokoly nemají být moc komplexní, a tedy potřebují spolupracovat s jinými protokoly. Spolupráce může probíhat buď v rámci jedné vrstvy, nebo mezi sousedními vrstvami, přičemž platí, že spodní vrstva poskytuje služby horní vrstvě.

*Entita* je aktivní prvek na určité vrstvě v modelu ISO/OSI, který má definováno rozhraní – sadu služeb, které může využívat entita z bezprostředně nadřízené vrstvy.[35]

Rozhraní mezi komunikujícími entitami (a tedy mezi vrstvami) se nazývá *SAP* (Service Access Point, přístupový bod služby). SAP tedy propojuje dvě entity v sousedních vrstvách – uživatele služby (service user) a poskytovatele služby (service provider).

Přes SAP se přenáší struktura, kterou označujeme *Interface Data Unit* (IDU, datová jednotka rozhraní)[21, 35, 37]. Tuto jednotku vytvoří a pošle do SAP jedna z komunikujících entit a na druhé straně SAPu ji přijme ta druhá. IDU se skládá ze dvou částí:

- *Service Data Unit* (SDU, služební datová jednotka) je v případě, že jsme na odesílajícím zařízení, obvykle PDU některého protokolu nadřízené vrstvy; jedná se tedy o data, která má zpracovat některý protokol nižší vrstvy,
- *Interface Control Information* (ICI, řídicí informace rozhraní) je informace, kterou přijímající entita potřebuje pro správné převzetí a zpracování SDU.

SDU jsou z pohledu entity data (přicházející od jiné entity), ICI jsou metadata, podobně jako například k souboru potřebujeme kromě dat v něm uložených také informaci o názvu souboru, jeho umístění, přístupových oprávněních, atd.

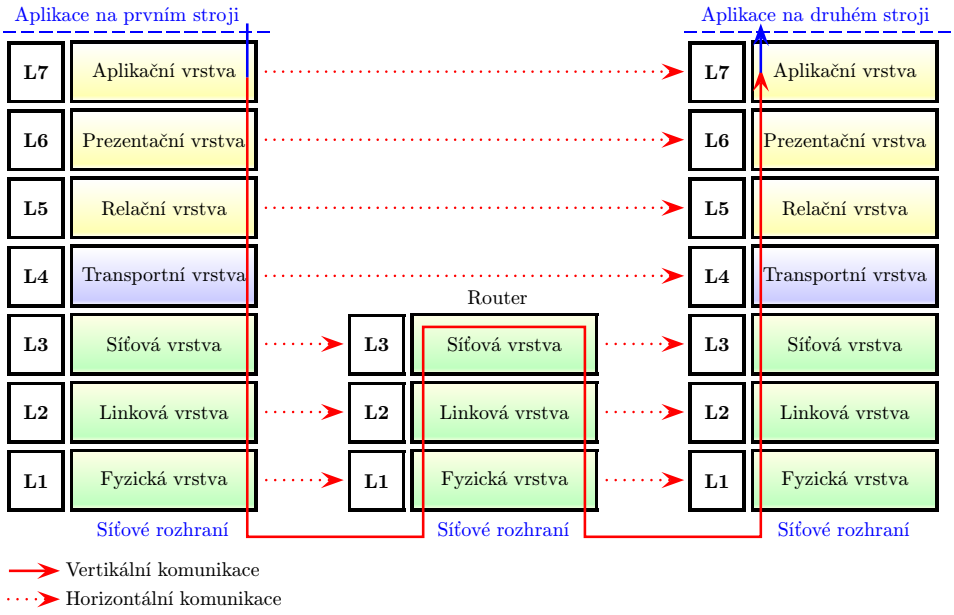
Způsob komunikace v rámci jednoho systému (řetěz střídajících se entit a SAP) se v ISO/OSI nazývá *vertikální komunikace*. *Horizontální komunikace* v ISO/OSI je komunikace mezi dvěma stejnými vrstami umístěnými na různých strojích, a to na logické úrovni. Oba způsoby komunikace jsou naznačeny na obrázku 2.3.

Pod pojmem entita si pro zjednodušení můžeme představit instanci některého konkrétního implementovaného protokolu. SAP může být v jednom okamžiku využíván pouze jedním uživatelem a jedním poskytovatelem, ale jakákoliv entita může zároveň používat více SAPů.

**Postup (Odesílání a přijímání dat po síti)**

Protokol, vlastně entita, při odesílání dat (podle obrázku 2.3 na stroji vlevo)

- obdrží IDU přes SAP od entity vyšší vrstvy, zpracuje informace v ICI (metadata) a z SDU (data) si vyzvedne data – což je PDU některého protokolu vyšší vrstvy,
- pokud je to nutné, stanoveným způsobem je zpracuje či rozdělí na menší bloky, případně jinak zpracuje,
- stanoví příslušné metainformace (tj. informace o informacích), například adresy, velikost dat, informace o tom, jak se má po cestě s daty zacházet apod.,
- přidá záhlaví (header) s metainformacemi a pokud je to třeba, pak i zápatí (trailer), přidá k datům, čímž data „zabalí“ do PDU (paket, rámec, datagram apod.),




Obrázek 2.3: Horizontální a vertikální komunikace v ISO/OSI (volně podle [35])

- vytvoří IDU pro nižší vrstvu – do SDU dosadí svůj vytvořený PDU, do ICI uloží další informace, které vyžaduje přijímající entita (některé vytvoří či vypočte, další převzal od jiného protokolu z téže nebo nadřízené vrstvy), vytvořený IDU pošle přes SAP entitě nižší vrstvy.

Jestliže jsou data naopak přijímána (podle obrázku 2.3 na stroji zcela vpravo), pak protokol

- přijme IDU přes SAP od nižší vrstvy, analyzuje ICI a z SDU vyzvedne data, která jsou vlastně jeho PDU,
- oddělí od PDU záhlaví a zápatí (pokud tam ovšem zápatí je),
- analyzuje metainformace v záhlaví a zápatí, stanoveným způsobem je zpracuje,
- pokud byla data před odesláním rozdělena na více bloků a jedná se o vrstvu, kde jsou bloky kompletovány (pozná se ze záhlaví), postupně shromáždí všechny bloky a zkompletuje,
- vytvoří IDU pro nadřízenou vrstvu, do SDU dosadí „rozbalená“ data, vygeneruje či dosadí potřebné informace do ICI a vše předá přes SAP entitě vyšší vrstvy.

Není řečeno, že se nutně mají zúčastnit všechny vrstvy, některé nejsou pro konkrétní druh komunikace potřebné. Ve skutečnosti se také na zpracování dat v rámci jedné vrstvy může podílet více protokolů než jen jeden[18, 21, 37].

**Poznámka**

Z toho vyplývá, že například při odesílání se prvek, který vznikl jako PDU některého protokolu na vyšší vrstvě, stává SDU pro některý protokol nižší vrstvy[21].

Jak se vlastně komunikuje přes takový SAP, co vše se na tomto přístupovém bodu děje? Ke každému SAPu jsou definována komunikační *primitiva*, což jsou jednoduché funkce, například *Request* (žádost o poskytnutí služby k nižší vrstvě, navazuje komunikaci přes SAP směrem dolů na straně odesílajícího stroje), *Indication* (upozornění na potřebu komunikace od nižší vrstvy k vyšší vrstvě, na straně přijímajícího stroje), *Response* (odpověď na Indication), *Confirm* (odpověď na Request). Například komunikace typu klient-server se z pohledu ISO/OSI skládá z těchto fází:

- klient odešle požadavek: na straně klienta jde směrem dolů primitivum *Request*,
- server přijme požadavek: na straně serveru jde nahoru primitivum *Indication*,
- server odešle odpověď: na straně serveru jde dolů primitivum *Response*,
- klient přijme odpověď: na straně klienta jde nahoru primitivum *Confirm*.

Kromě toho existují i další primitiva, také specifická pro konkrétní protokoly.

Tato primitiva mohou mít, podobně jako běžné funkce, parametry – to je právě struktura IDU, o které se píše v textu výše.[8, 4]. V operačních systémech se primitiva implementují typicky jako rutiny v API (volání procedury).

**Poznámka**

Zvídavého čtenáře určitě napadlo, že musí existovat způsob, jak se například síťová vrstva (L3) „dozví“, na kterou adresu má být vlastně dotyčný paket poslán. Přes záhlaví PDU to být nemůže, protože dovnitř záhlaví vyšších vrstev se protokol IP nedostane (je to pro něj prostě součást dat, která je třeba poslat), navíc v některých záhlavích tato informace ani není. Ano, dozví se to z parametrů primitiv. Další informace najdete na http://www.erg.abdn.ac.uk/users/gorry/course/intro-pages/service-prim.html. [cit. 2022-06-08]

*Socket* je kombinace síťové adresy (používané na vrstvě L3) a čísla portu (používaného na vrstvě L4). Pokud tento pár zapisujeme „ručně“ (třeba do adresního řádku webového prohlížeče), umístíme mezi oba údaje dvojtečku.

Místo síťové (tedy číselné) adresy může být použita doménová adresa[37, 41]. Například pokud máme komunikovat se serverem `www.neco.cz` na portu 8080, bude se jednat o socket `www.neco.cz:8080`.

Sockety (sockets) najdeme na vrstvách L5–L7, víceméně fungují jako rozhraní mezi aplikačními protokoly a transportní vrstvou. Pro aplikace jsou dostupné jako Socket API (tj. rozhraní pro programování aplikací) ve formě knihoven obsahujících funkce pro práci se sockety (pro vytvoření socketu, akceptování na druhé straně spojení, naslouchání, čtení, zápis do socketu). Speciálním typem socketu je *stream socket*, který zaručuje dodání více bloků posílaných dat ve správném pořadí, a tedy na transportní vrstvě spolupracuje pouze s protokoly zajišťujícími spojovaný přenos (třeba TCP). Se sockety se setkáme jak v UNIXových systémech (včetně Linuxu a MacOS), tak i ve Windows (WinSock API).

**Další informace**

Informace k WinSock API najdeme například na https://msdn.microsoft.com/en-us/library/windows/desktop/ms740632(v=vs.85).aspx.

Informace o UNIX Sockets jsou například na https://www.tutorialspoint.com/unix_sockets/, na manuálových a info stránkách a v dalších zdrojích.[cit. 2022-09-18]

### 2.2.3 Protokolové zásobníky

Pojmy protokolový zásobník nebo protokolová sada používáme, když chceme určit, které protokoly konkrétně mají spolupracovat, být používány na tomtéž zařízení.

**Definice (Sada protokolů, protokolový zásobník [37, 41])**

*Sada protokolů* (Protocol Suite) je definice (určení) skupiny protokolů, které spolupracují. *Protokolový zásobník* (Protocol Stack) je implementace některé sady protokolů.

Protokolový zásobník k některé protokolové sadě nemusí nutně implementovat všechny protokoly v sadě zahrnuté, protože sada může pro tentýž účel definovat několik alternativních protokolů; při implementaci do formy protokolového zásob-

níku se typicky vybírá jen jeden z alternativních protokolů. Takže protokolová sada může být rozsáhlejší než protokolový zásobník, který je její implementací.

Nemusí nutně jít o specifikaci protokolů pro naprosto všechny vrstvy referenčního modelu ISO/OSI, mohou být specifikovány pouze některé, přičemž se předpokládá spolupráce s jiným (doplňujícím) protokolovým zásobníkem. Z níže uvedených je nejznámější síťový zásobník TCP/IP, který je na spodních vrstvách ISO/OSI doplňován například některou sadou pro lokální sítě (Ethernet, Wi-fi).

**TCP/IP Protocol Suite** (taky se nazývá *Internet Protocol Suite*) je sada navzájem spolupracujících protokolů, kterou potřebujeme na koncových zařízeních připojených k rozsáhlé síti (typicky Internetu). Kromě protokolů obsažených přímo v názvu (TCP, IP) zahrnuje ještě další, nejvíc na aplikační vrstvě. Přímo jsou určeny protokoly na vrstvách L3–L7, pro nižší vrstvy je pouze specifikováno rozhraní.[21]

Tento protokolový zásobník je formalizován jako síťový model TCP/IP, kterému se budeme podrobněji věnovat v následujícím textu.

**IPX/SPX** je konkurenčním protokolovým zásobníkem k TCP/IP od společnosti Novell vytvořeným pro síťový operační systém Novell Netware – IPX pracuje na L3 místo protokolu IP, SPX na vrstvě L4 místo TCP. Byl projektován spíše pro menší sítě, zatímco TCP/IP je určen i pro rozlehlé sítě. V současné době se už téměř nepoužívá.[41]

**Protokolové sady pro lokální sítě** jsou například Ethernet (IEEE 802.3), Wi-fi (IEEE 802.11) a další. Obvykle implementují pouze vrstvy L1 a L2, nad ně se nasouvá většinou protokolový zásobník TCP/IP.

**Protokolové sady pro rozlehlé sítě** jsou například ATM, Frame Relay, MPLS a další. Taky se obvykle napojují na TCP/IP, ale každá „trochu jinak“.

Například ATM se podsouvá pod síťovou vrstvu (L3), ale mezi ni a svou implementaci vrstvy L2 vsouvá speciální přizpůsobovací vrstvu. Frame Relay implementuje vrstvu L2, na L1 předpokládá některé vhodné fyzické rozhraní, většinou dle standardů EIA/TIA (spoléhá na ISO/OSI).

Oproti tomu MPLS se vsouvá mezi vrstvy L2 a L3, tedy MPLS paket v sobě zabaluje paket z vrstvy L3 (většinou IP paket) a je zabalen do rámce vrstvy L2 (například do ethernetového rámce). Taky může běžet nad ATM nebo Frame Relay, a tedy lze využít technologie ze starších zařízení. Taky dokáže běžet nad PPP a dalšími protokoly pro přístupové sítě.

**Protokolové sady pro mobilní sítě** jsou například sady pro LTE, GPRS, CDMA, UMTS a další. Implementují obvykle vrstvy L1 a L2 a předpokládají některé konkrétní protokoly i na vyšších vrstvách, ale ve skutečnosti záleží, o jaký typ zařízení jde (koncové zařízení bude potřebovat jinou sadu protokolů než základnová stanice nebo jiná specializovaná zařízení v mobilní síti).[22, 37]

**Další protokolové sady** pro různé typy sítí se specifickým provozem, například M2M (Machine-to-Machine) sítě, ve kterých přenášená data negenerují lidé, ale stroje. Sem můžeme zařadit i protokolové sady pro IoT sítě, protože zařízení Internetu věcí se do značné míry bez lidí obejdou a mají poměrně specifické požadavky na síťový provoz. Sítím Internetu věcí se budeme věnovat v kapitole 3 Internet věcí.

### 2.2.4 Síťový model TCP/IP

Referenční model ISO/OSI je velmi komplexní, tudíž složitý. Existuje také několik zjednodušených variant, z nichž je nejznámější právě síťový model TCP/IP, který je také nazýván DoD model (USA Department of Defense Model, tedy Model Ministerstva obrany USA), dalším typickým názvem je Internet protocol suite[21, 22].

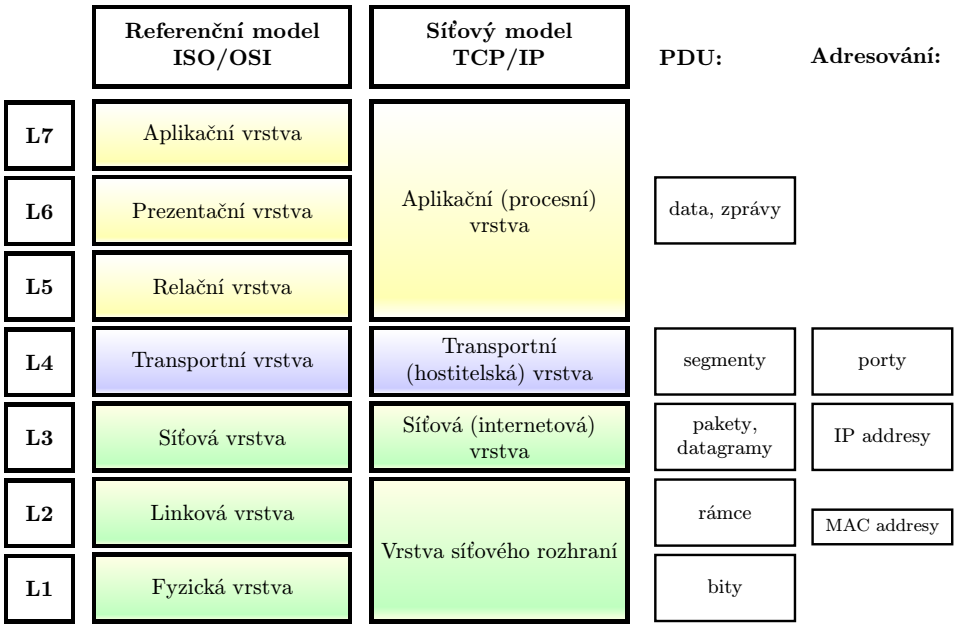
Síťový model TCP/IP vznikl začátkem 70. let 20. století (protokol TCP byl vytvořen roku 1973), zatímco referenční model ISO/OSI je přibližně o deset let mladší. Takže sice většina knih nejdřív popisuje ISO/OSI a pak do něj „napasovává“ TCP/IP, ale historicky to vlastně bylo naopak[21].

Součásti modelu TCP/IP jsou standardizovány organizací IETF a dostupné v RFC dokumentech. RFC dokumentů je velmi mnoho, také existuje jakýsi krátký výukový průvodce – RFC 1180 „A TCP/IP Tutorial“, který vysvětluje princip modelu, jeho strukturu, základní protokoly, datové toky v modelu a protokoly, které se typicky na TCP/IP napojují.[19]

Síťový model TCP/IP je ve skutečnosti formální popis protokolové sady (zásobníku) TCP/IP, jeho napojení na spolupracující zásobníky a obecně možnost zapojení jiných protokolů. Skládá se ze čtyř vrstev, jejichž vztah k vrstvám modelu ISO/OSI je naznačen na obrázku 2.4.

Vztahy vyjádřené na obrázku platí i co se týče funkčnosti – aplikační (procesní) vrstva TCP/IP plní tutéž roli jako vrstvy L5–L7 v ISO/OSI. TCP/IP je navržen tak, aby[21]

- byl co nejvíc decentralizovaný (žádná centrální správa),
- byl co nejodolnější vůči různým (i kritickým) podmínkám provozu a co nejodolnější vůči přenosovým chybám,




Obrázek 2.4: Srovnání modelů RM ISO/OSI a TCP/IP (podle [21])

- nechával co nejvíc práce na koncových zařízeních (jádro sítě má být rychlé a pružné), a aby byl spravovatelný distribuovaně (každá část si spravuje „to svoje)“,
- dokázal propojit i sítě s hodně odlišnou síťovou architekturou a technologiemi (aby byl co nejuniverzálnější při zachování předchozích vlastností).

## 2.3 Protokol HTTP

Protokol HTTP je jeden z nejpoužívanějších aplikačních protokolů, který většinou využívá služeb transportního protokolu TCP a tedy jde o komunikaci s navázáním spojení – TCP handshake.[21]

Na různých verzích protokolu HTTP si ukážeme, jak moc či málo lze umístit do navázaného spojení a jak se tento parametr může vyvíjet přes různé verze téhož protokolu. Protokol HTTP je typickým zástupcem komunikace typu klient-server, modelu Request-Response (tj. klient pošle žádost, server odpoví). Informace uvedené v této sekci vycházejí především ze zdroje [25].

**HTTP/0.9** byl jakousi „předběžnou“ verzí. Klient si po navázání relace se serverem vyžádal stránku metodou GET (jediný request, žádná URL, jen interní adresa k dokumentu, jinou metodu než GET nepodporoval), server tuto stránku poslal a relace byla ukončena.

**HTTP/1.0** byl navržen pro přenos jednoduchých stránek. Ovšem už bylo možné připojovat i další soubory, například s kaskádovými styly či obrázky. Rozšířila se skupina podporovaných metod na straně klienta (GET, HEAD, POST), na straně serveru bylo už možné používat různé kódy (včetně HTTP OK) a komunikující zařízení se už dokázala navzájem představit – server mohl zjistit, jakému typu klienta stránku posílá. Ovšem každý z posílaných souborů musel mít vlastní relaci, relace se navazovaly sekvenčně (když první skončila, mohla být navázána druhá, atd.).

**HTTP/1.1** (což je momentálně nejpoužívanější verze) přišel s tzv. persistent connections, tedy spojeními trvajícími delší dobu, ne pouze pro přenos jednoho souboru (to vpodstatě uměl i HTTP/1.0, ale bylo nutné toto chování zapnout pomocí volby Keep-alive). V rámci jedné relace je možné přenést postupně více souborů, není nutné tolikrát provádět TCP Handshake. Odpovědi je třeba zasílat ve stejném pořadí, v jakém byly poslány dotazy. Server si vede frontu dotazů (Request Queue) a postupně tuto frontu vyprazdňuje, tedy na dotazy odpovídá.

V HTTP/1.1 lze používat *pipelining*, což znamená, že klient může v rámci spojení poslat další dotaz ještě dřív než dostane odpověď na předchozí dotaz. Klient může poslat další dotaz až tehdy, když si je jist, že předchozí dotaz byl poslán správně, proto server pro dotaz pošle ACK (potvrzení) a až poté řeší odpověď. Toto chování však může být problematické, zejména pokud provoz jde přes proxy, proto ho klientské aplikace moc nepoužívají.[1]

Pravděpodobněji se setkáme s tím, že klientské aplikace raději otevřou několik paralelních relací na tentýž server a jednotlivé požadavky řeší paralelně tímto způsobem. Nevýhodou je, že podobně jako v HTTP/1.0 nám poněkud roste režie (zejména TCP handshaky), ale na druhou stranu mohou provádět více dotazů paralelně. Ve většině webových prohlížečů se setkáváme s počtem paralelních spojení v řádu jednotek, obvykle do 6 nebo 10.

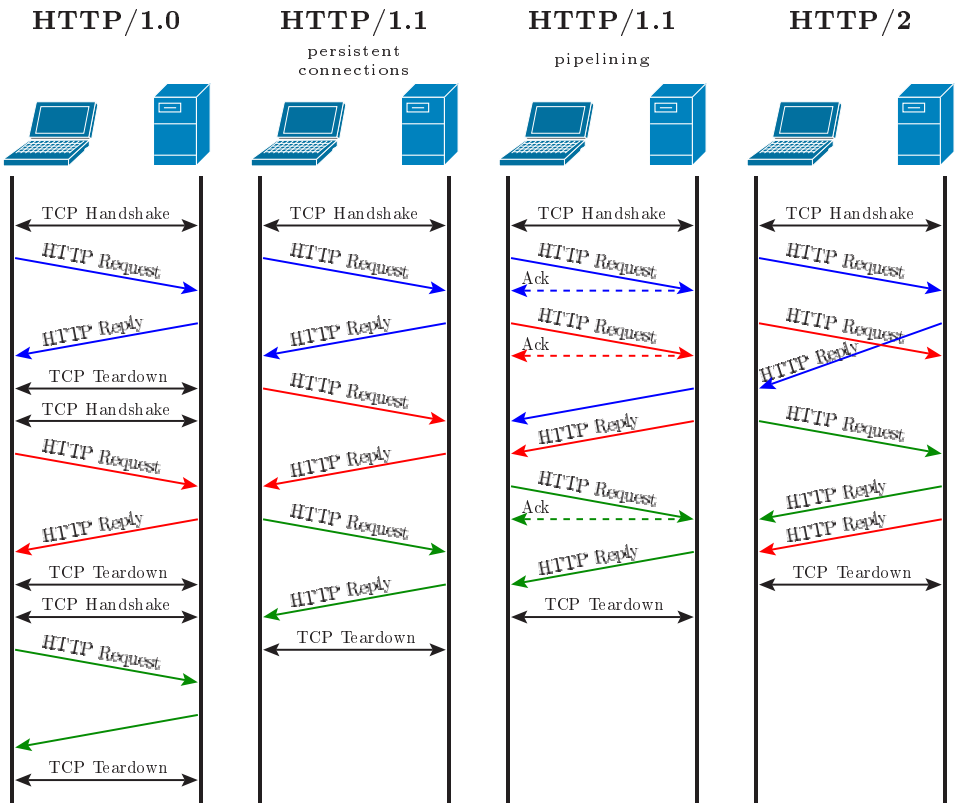
**HTTP/2** jde ještě dál. Přišel s multiplexováním spojení, kdy v rámci jednoho spojení (navázaného jediným TCP handshakem) existuje více streamů pro jednotlivé žádosti/odpovědi, tyto streamy fungují asynchronně (tj. navzájem zcela nezá-

[1] Podrobnosti můžeme najít například na https://engineering.cred.club/head-of-line-hol-blocking-in-http-1-and-http-2-50b24e9e3372 [cit. 2022-07-07].

visle, klidně paralelně). Taktéž server rozlišuje různé typy obsahu a některé v rámci spojení prioritizuje. Novinek je ještě více, podrobnosti najdeme například v [25].

Na obrázku 2.5 je srovnání komunikace ve verzích HTTP 1.0, 1.1, 1.1 s pipeliningem a 2, barevně jsou odlišeny jednotlivé požadavky a k nim příslušející odpovědi.

HTTP je klient-server protokol, který pracuje na principu Request-Response. Request je zpráva posílaná klientem, response je odpověď serveru. Klient své dotazy formuluje pomocí metod GET, POST, PUT, DELETE a dalších, server jako odpovědi posílá zprávy, v jejichž záhlaví najdeme stavový kód určující, jak se server s dotazem vypořádal.




Obrázek 2.5: Rozdíly mezi verzemi protokolu HTTP v komunikaci klient-server: klient je vždy vlevo, server vpravo (volně podle [10, 25])

**Další informace**

Popis HTTP zpráv a stavových kódů najdeme ve standardu RFC 7231, jehož plné znění je na https://datatracker.ietf.org/doc/html/rfc7231. [cit. 2022-07-01]

## 2.4 Jak na úspornou komunikaci: REST API

REST (Representational State Transfer) je vlastně sada pravidel navržená původně pro komunikaci s webovými aplikacemi. Zatímco při komunikaci s běžným webem si klient prostě vyžádal stránku (případně další její součásti) a zpět v odpovědi obdržel to, co bylo vyžádáno, u „klikacích" webových aplikací se předpokládá (z pohledu uživatele) prakticky neustálá konverzace mezi klientem a serverem. Proto bylo potřeba vymyslet způsob, jak tuto komunikaci co nejvíce zefektivnit.[13]

Nejčastěji se setkáváme s webovými aplikacemi, které jsou založeny na HTTP REST, nicméně samotný REST není přímo vázán na HTTP (jak ostatně zjistíme v následující kapitole). Mechanismus REST předpokládá bezstavovou komunikaci, tj. nepáruje se odpověď ke konkrétnímu dotazu, třebaže odpověď typicky k nějakému dotazu patří.

REST je pouze o datech, nikoliv o metodách. Účelem je efektivní zpřístupnění dat klientovi. Konkurence (například SOAP či RPC) je oproti tomu spíše o metodách, procedurách.[2]

Takže co je potřeba pro použití REST:

- určení zdrojů na serveru, adresace pomocí URI, data by pro usnadnění orientace měla být v hierarchické struktuře a navíc opatřená metadaty,
- nosná platforma starající se o to, co v REST není (procedury), obvykle HTTP,
- určení toho, co se s daty dá provádět, a také namapování na procedury nosné platformy,
- hypermédia (zdroje jsou navzájem propojené a klient mezi nimi může libovolně přecházet).

Poslední uvedenou vlastnost si můžeme jednoduše představit jako hypertextové odkazy mezi dokumenty.

[2] Srovnání REST a SOAP najdeme například na https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest [cit. 2022-08-02].

Zdroje (resources) na serveru jsou uspořádány ve stromě, jehož kořen označujeme lomítkem. Samotné názvy zdrojů mohou být různé, jen by měly dávat smysl. Například ve zdroji `/articles` bychom očekávali články, a pokud následuje další lomítko a název souboru, pak zřejmě půjde o konkrétní článek; případně články mohou být členěny do košatější struktury.

Co s daty může dělat klient, to popisuje zkratka CRUD (Create, Read, Update, Delete). Na HTML se tyto operace mapují na metody POST (=Create), GET (=Read), PATCH nebo PUT (=Update) a DELETE (=Delete). Takže pokud například klient chce ze serveru zpravodajsky-server.com článek o stavu rybolovu v roce 2022, může to vypadat takto:

```
GET /articles/2022/stav-rybolovu
HOST: zpravodajsky-server.com
```

Pokud je třeba článek nebo jakýkoliv jiný objekt ve zdrojích serveru teprve vytvořit, použije klient metodu POST. Jestliže je potřeba existující objekt ve zdrojích jen pozměnit, pošle klient zprávu PATCH, a případné odstranění objektu ze zdrojů serveru provede metoda DELETE.[10, 13, 24]

Na straně serveru se u REST setkáváme s tím, co známe z klasického HTML: například kód 200 (HTTP OK) znamená, že vše je v pořádku a požadovaný objekt je posílán, kód 201 (Created) je odpovědí na REST operaci Create, ale také se samozřejmě setkáváme s kódy začínajícími trojkou (většinou přesměrování) nebo čtyřkou (chyba na straně klienta) a pětkou (chyba na straně serveru).

Protože v REST musíme s daty nějakým způsobem manipulovat, potřebujeme s daty přenášet i metadata. Většinou se pro tento účel používá formát JSON, můžeme se setkat také s XML nebo některou jeho odnoží. V našem příkladu může na serveru existovat seznam článků ve formátu JSON, a tedy pokud chceme zjistit, jaké články na webu najdeme, vyžádáme si právě tento soubor.[13]

Co vlastně REST přináší jiného než HTTP? Především REST je o zdrojích, HTTP je pro REST pouze prostředek pro manipulaci s nimi. A dále máme u webových služeb předpřipravené API (hovoříme o REST API, také se můžeme setkat s označením RESTful API). Grafické rozhraní totiž server určí klientovi na začátku, ale téměř veškerá další komunikace je právě o datech (obsahu), a klient potřebuje vědět, jak si potřebná data vyžádat či je poslat. REST API ostatně nabízí i jeden z nejpoužívanějších redakčních systémů – WordPress, také vyhledávací engine Elasticsearch je RESTful.[3]

[3]WordPress REST API je popsáno na https://developer.wordpress.org/rest-api/, Elastic-

REST dnes používají prakticky všechny webové aplikace včetně sociálních sítí (své API má pěkně sestaveno například Twitter[24], což je praktické z pohledu různých webových a síťových aplikací na Twitter přistupujících), dále se s ním setkáme často i v IoT sítích. K využití REST pro IoT se vrátíme v sekci 3.6 HTTP REST pro IoT.

## 2.5 Stavová komunikace na aplikační vrstvě

Protokol HTTP je používán spíše jako bezstavový, REST v principu bezstavovost předpokládá. Ale někdy také na webu potřebujeme komunikovat stavově.

*Stavová komunikace* je taková komunikace, kde odpověď je přímo navázaná na dotaz. Klient ví, ke kterému dotazu přiřadit došlou odpověď (a taky pozná, když přišla nevyžádaná odpověď), server ví, na které dotazy už klientovi odpovídal a tedy stačí poslat jen update.[41]

Většinou si vystačíme s bezstavovou komunikací. Je to jednodušší, klient ani server si nemusejí udržovat přehled o stavu komunikace, prostě jen posílají dotazy a odpovědi. Ale jindy se nám hodí mít o stavu přehled.

**Postup (Udržování stavu komunikace pomocí cookie)**

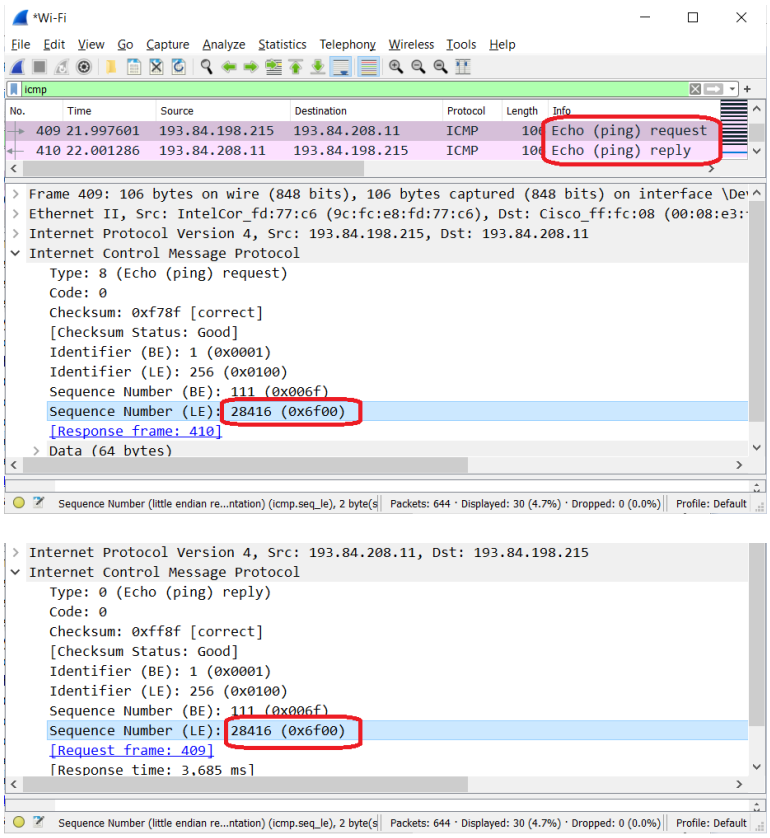
Pokud zůstaneme u *webové komunikace*, pak například internetový obchod potřebuje vědět, co zatím klient naložil do košíku, jaké zboží už si prohlédl, a v neposlední řadě i o kterého klienta vlastně jde (zejména když je registrovaný a chceme mu v objednávce předvyplnit uložené údaje).

K tomu se většinou používají cookies. V cookie na straně klienta je uložen identifikační řetězec, stejný řetězec máme v databázi klientů a do databáze si průběžně ukládáme veškeré informace s klientem související. Podle typu cookie se pak tato informace přenáší buď s každým HTTP dotazem nebo v jiné frekvenci. V souvislosti s HTTP se dají využít i jiné „berličky“ přidávající stavovost, cookie je asi nejjednodušší.[37]

search REST API na https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html [cit. 2022-08-28].

**Postup (Stav ověřování dostupnosti)**

*Protokol ICMP* při použití zprávy Echo/Echo reply má stavovost přímo zabudovanou, a to ve formě identifikátoru *Sequence Number*, který je součástí záhlaví: stejný v dotazu i odpovědi.




Obrázek 2.6: Hodnota Sequence Number ve zprávě ICMP Echo a tatáž hodnota pro odpověď ICMP Echo reply

Na obrázku 2.6 je screenshot se zachycenými zprávami ICMP Echo a ICMP Echo reply. Wireshark vypisuje Sequence number ve dvou řádcích, zvlášť pro kódování big-endian (BE) a little-endian (LE).

**Poznámka**

Existují dva základní způsoby, jak v paměti reprezentovat vícebytové číslo. Kódování big-endian umísťuje v paměti nejvýznamnější byte (MSB, Most Significant Byte, tedy ten, který „na papíře“ píšeme zcela vlevo) na nejnižší adresu, následující byte na vyšší adresu, atd., tedy nejméně významný byte (LSB, Least Significant Byte) je na nejvyšší adrese. V případě přenosu po síti se jedná o adresu umístění ve streamu, tedy u big-endian přijde nejdřív MSB, na konci poslední část čísla LSB.

Little-endian to má naopak: na nejnižší adrese je LSB, kdežto na nejvyšší adrese najdeme MSB. Existuje i „něco mezi“, což je kódování mixed-endian (middle-endian).[35, článek Little Endian vs. Big Endian]

V sítích může nastat problém tehdy, když jsou zprávy posílány mezi zařízeními používajícími různá kódování. Na platformě x86/amd64 se setkáme s ukládáním vícebytových čísel v kódování little-endian. ARM má implementováno obojí a na těchto procesorech lze přepnout mezi big- a little-endian. V současné době naprostá většina provozu je v kódování little-endian.[35]

V této sekci monografie najdete několik screenshotů programu Wireshark. Tento program je volně ke stažení a je to jeden z nejdůležitějších pomocníků při prověřování síťového provozu.

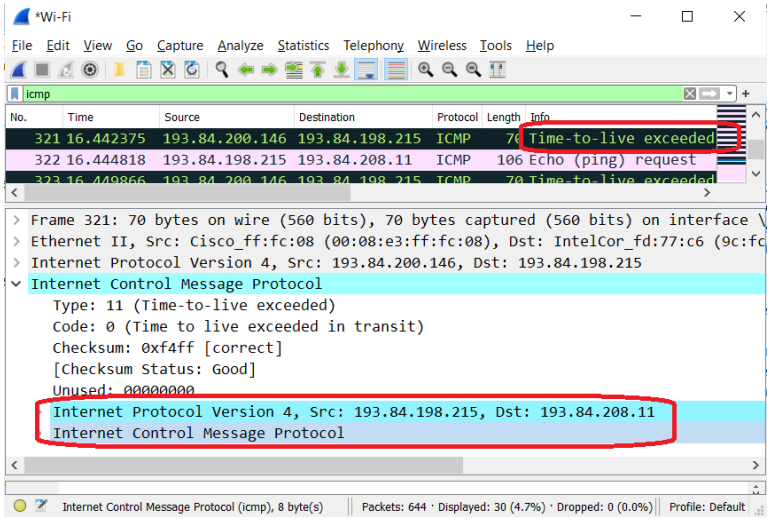
**Další informace**

Pokud se čtenář ještě nesetkal s programem Wireshark: program se dá stáhnout na https://www.wireshark.org/ [cit. 2022-09-08], tamtéž najdeme dokumentaci a nápovědu.

**Postup (Jak sdělit odesílateli, který paket byl zahozen)**

Jiné typy ICMP zpráv také potřebují stavový režim. Například zpráva ICMP Time exceeded se kromě jiného posílá tehdy, pokud router zjistí u přeposílaného paketu hodnotu TTL=0. Paket je zahozen a touto zprávou router sděluje původnímu odesílateli, že daný paket nebylo možno doručit.

Na obrázku 2.7 je zpráva ICMP Time exceeded informující o nedoručení paketu směřujícího na adresu 193.84.208.11. Paket byl odeslán jako součást mechanismu `traceroute`, tedy s nízkým TTL a očekáváním, že se ozve ten router na cestě, na

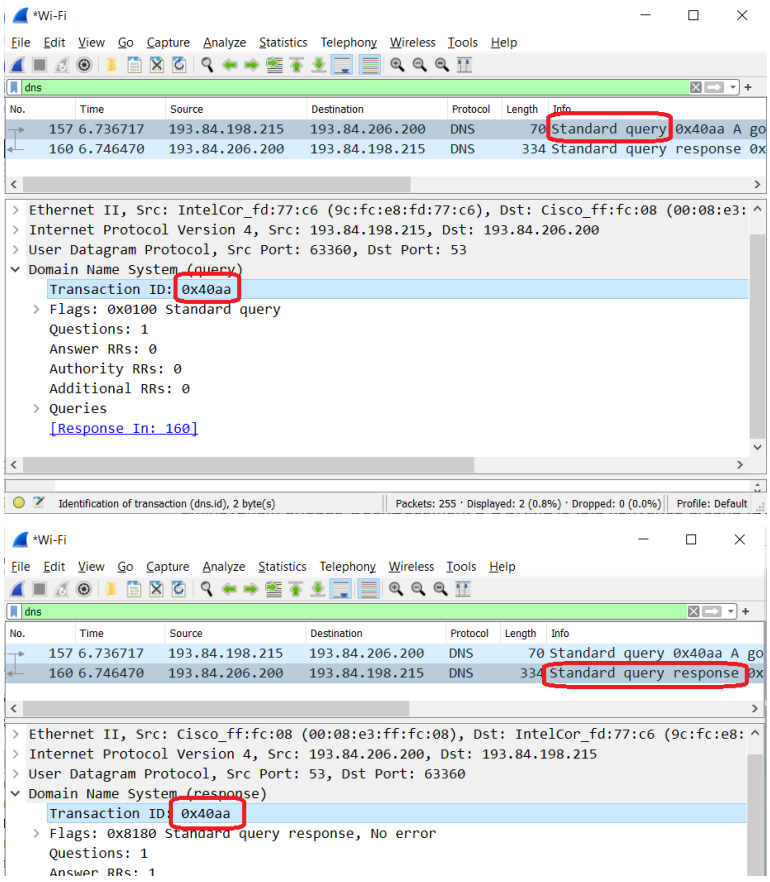



Obrázek 2.7: Vložený paket ve zprávě ICMP Time exceeded

němž TTL vyprší. Původní odesílatel, který je ICMP zprávou informován o nedoručení paketu, může tuto zprávu spojit s konkrétním odeslaným paketem například podle identifikačního čísla, které je součástí každého IP záhlaví (a tudíž je součástí dat reportovaných ICMP zprávou).[21]

**Postup (Párování DNS dotazu a odpovědi)**

Dalším protokolem, pro který je stavové chování důležité, je *DNS*. Stavovost je DNS zprávě zajištěna číslem v poli Transaction ID (jak je vidět na obrázku 2.8). Stejné číslo najdeme v DNS dotazu i DNS odpovědi, což je důležité hlavně tehdy, pokud má klient rozpracováno více DNS dotazů najednou. Navíc v DNS odpovědi najdeme nejen samotné odpovědi, ale také pole s dotazy.[21]




Obrázek 2.8: Hodnota Transaction ID v DNS zprávě

Kapitola 3

# Internet věcí

Jen málokdo v současném světě (alespoň v Evropě) se se zařízeními Internetu věcí nikdy v životě nesetkal. Doma, v práci, v obchodě, ale i na ulici jsme obklopeni senzory, chytrými zařízeními, obrazovkami a dalšími „věcmi“ připojenými do sítě, a ani si to často neuvědomujeme.

## 3.1 Co je to Internet věcí

V názvu této kapitoly je slovo „věci“ – o jaké věci tedy jde? Co tyto věci mohou umět, obsahovat? Mohou obsahovat různé senzory (pro detekci teploty, vlhkosti nebo přímo vody, světla, vzdálenosti, RFID čipy), dále aktuátory (motory) vytvářející pohyb, například motory na otevření oken, dveří či vrat, spuštění žaluzií, také se může jednat o alarmy, ovládání světel, informační displeje. Můžeme také přidat některá zařízení z vybavení domácnosti (televize, ledničky, vysavače a další) nebo třeba mobilní telefony a síťová zařízení. Tato zařízení se mohou nacházet doma, v zaměstnání, na ulici ve formě chytrých parkovacích míst či propojených kamer, ale také v průmyslových provozech nebo na polích, kde sledují vlhkost půdy či nálety škůdců.

Zařízení Internetu věcí (Internet of Things, IoT) jsou tedy více či méně komplikovaná zařízení, která v sobě tyto prvky kombinují, ale to není všechno. Důležité je jejich vzájemné propojení a také automatizace provozu, která od lidí nevyžaduje příliš mnoho interakce. Typicky se jedná o komunikaci typu M2M (Machine-to-

Machine), do které lidé obvykle přímo nezasahují.

Skutečná ucelená a všeobecně uznávaná definice IoT vlastně neexistuje, třebaže několik víceméně odpovídajících se najít dá. V článku [20] najdeme celou řadu definic převzatých z různých zdrojů, k tomu několik částečných definic.

Z toho všeho můžeme vyextrahovat následující:

**Definice (podle [20], [45])**
Internet věcí (Internet of Things, IoT) je síť různých typů chytrých objektů (věcí) a zařízení. Věci mohou být (ne nutně přímo) připojeny do Internetu a komunikují navzájem s minimální potřebou zásahu člověka. Zapouzdřují v sobě různou funkcionalitu jako je odečítání dat ze senzorů, jejich analýza, zpracování a samospráva založená na komunikačních protokolech a specifických kritériích, politikách.

## 3.2 Komunikace v Internetu věcí

Také v IoT můžeme rozlišovat zařízení typu klient a server, stejně jako v běžných počítačových sítích. Serverové zařízení poskytuje informace a klientské zařízení je schopno si tyto informace vyžádat. V běžných počítačových sítích se obvykle setkáváme s komunikací typu klient-server, ale ve specializovaných sítích (včetně IoT) nemusí být toto rozdělení rolí až tak striktní a často se tyto sítě blíží způsobu komunikace peer-to-peer (totéž zařízení může pracovat jako klient, ale zároveň jako server, jen pro jiné typy dat). Připojené senzory sice bývají zdrojem dat a tedy není problém, když iniciují navázání komunikace, ale někdy musí iniciativa mířit i opačným směrem.

Různé komunikační protokoly pro IoT také mohou různě zatěžovat linky. Spojení může/nemusí být navazováno, podle toho, který protokol je na transportní vrstvě. Navazování komunikace více zatěžuje linky, záhlaví posílaných PDU bývají větší, ale na druhou stranu lze jednoduše potvrzovat zprávy a zajistit spolehlivost doručení.[4]

Co se týče konkrétních *komunikačních modelů* (také komunikačních paradigmat), tj. způsobu, jak konkrétně komunikace a datový tok mezi klienty a servery probíhá, v nich tedy určitý rozdíl je. Zatímco v klasických počítačových sítích se prakticky výhradně setkáváme s modelem Request-Response, pro svět IoT je naopak praktičtější model Publish-Subscribe.

Je třeba upozornit, že o komunikačních modelech hovoříme většinou v souvislosti s vyššími vrstvami modelu ISO/OSI, obvykle s aplikační vrstvou. Pod nimi obvykle pracují protokoly ze síťového zásobníku TCP/IP.

**Komunikační model Request-Response** je založen na jednoduchém přeposílání zpráv: klient zašle zprávu serveru coby žádost (request), server zpět pošle odpověď (response) také ve formě zprávy. Klient ovšem musí „vědět“, komu svou žádost poslat, tedy potřebuje znát adresu (či jiný identifikátor) serveru. Na straně serveru to je jednodušší, protože klient může svou adresu (identifikátor) poslat v žádosti. V tomto modelu nepoužíváme zprostředkovatele, komunikace probíhá přímo mezi klientem a serverem.

**Komunikační model Publish-Subscribe** na to jde jinak. Kromě původce dat (publisher) a příjemce dat (subscriber, odběratel) bývá v síti ještě zprostředkovatel (broker, server, controller). Publisher generuje data a posílá je brokerovi, nezná své odběratele. Broker si vede databázi požadavků na konkrétní typy informací (tzv. témata), ve které je uvedeno, o jaká témata má zájem který subscriber. Pokud broker obdrží data patřící do určitého tématu, pak zkontroluje v databázi, kteří odběratelé mají o téma zájem, a těmto odběratelům data přepošle.[20]

Na obrázku 3.1 je ukázka sítě se třemi publishery, dvěma subscribery a brokerem. Každý subscriber si objednal dvě témata, která mu dále budou přeposílána.

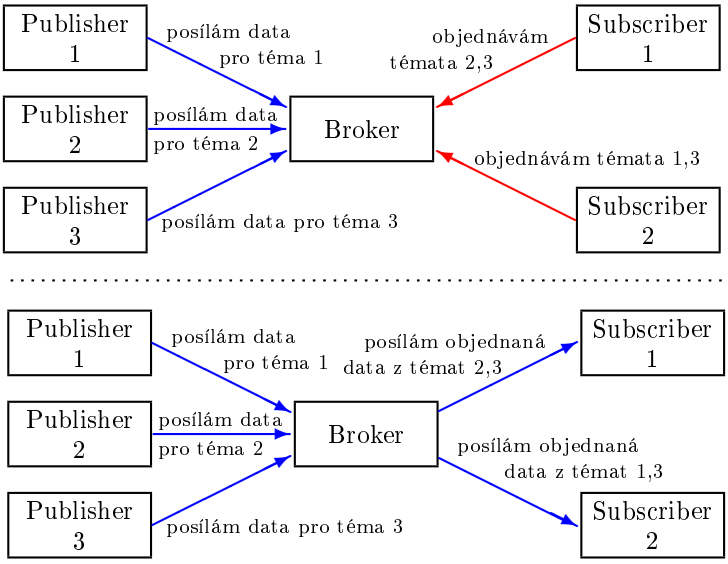
**Poznámka**

Pro zařízení Internetu věcí lze použít celou řadu protokolů, například MQTT, CoAP, XMPP nebo jednoduše HTTP ideálně s architekturou REST. Většina z nich dokáže pracovat v režimu Request-Response i Publish-Subscribe, nicméně jak bylo výše uvedeno, druhá možnost je pro IoT výhodnější.

V tabulce 3.1 najdeme srovnání základních vlastností několika aplikačních protokolů, o kterých jsme se zatím zmínili. K těmto protokolům se budeme vracet i v dalších sekcích, zde nás zatím zajímá v nich zabudovaná podpora komunikačních modelů Request-Response a Publish-Subscribe.

## 3.3 Protokolový zásobník pro IoT síť

Síťovými modely a protokolovými zásobníky jsme se zabývali v kapitole o protokolech: na obrázku 2.2 na straně 18 je zobrazen referenční model ISO/OSI, na ob-




Obrázek 3.1: Ukázka modelu Publish-Subscribe: nejdřív registrace k odběru témat, pak plnění objednávek (podle [48])

| **Protokol** | **Standard** | **Transportní vrstva** | **Zabezpečení** | **Request -Response** | **Publish -Subscribe** |
|---|---|---|---|---|---|
| AMQP | OASIS | TCP | TLS | ✓ | ✓ |
| CoAP | IETF | UDP | DTLS | ✓ | ✓ |
| DDS | OMG | TCP/UDP | TLS/DTLS/DDS | | ✓ |
| HTTP/2.0 | IETF | TCP | TLS | ✓ | ✓ |
| MQTT | OASIS | TCP | TLS | | ✓ |
| XMPP | IETF | TCP | TLS | ✓ | ✓ |
| REST HTTP | IETF | TCP | TLS | ✓ | |

Tabulka 3.1: Srovnání základních vlastností několika aplikačních protokolů pro IoT (volně podle [10])

rázku 2.4 na straně 25 máme srovnání modelu ISO/OSI a síťového modelu TCP/IP. Teď se zaměříme na protokolové zásobníky pro sítě internetu věcí.

Funkcionalita v IoT síti je specifická především na aplikační vrstvě, protože tam používáme krátké zprávy ve speciálním formátu, ale také na nejnižších vrstvách, protože potřebujeme (ne vždy) energeticky úsporný bezdrátový přenos.

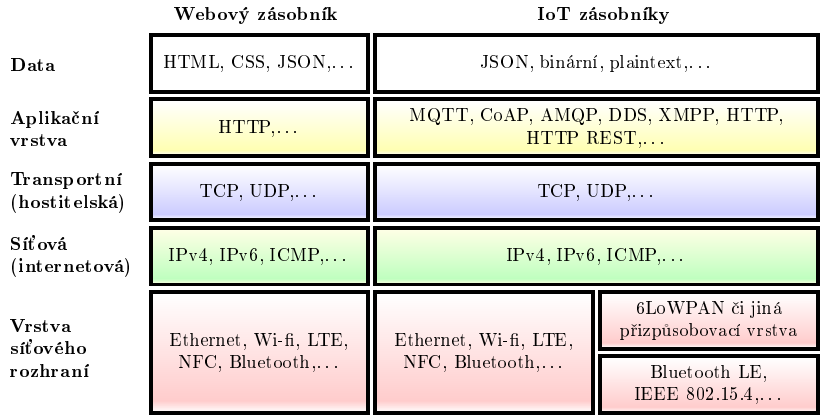
Jak vidíme, svět protokolů IoT není vůbec jednoduchý. Existují otevřená řešení (od různých standardizačních organizací), ale taky ta proprietární od konkrétních výrobců. Vzájemnou kompatibilitu nelze očekávat, a tedy při pořizování těchto produktů je třeba si ověřovat podporované protokoly.

Ovšem pozor – podle obrázku 3.2 to vypadá, že protokoly IEEE 802.15.4, Bluetooth LE a další se prostě podsunou pod TCP/IP, případně s přizpůsobovací vrstvou. Tak jednoduché to bohužel není, naznačená struktura by spíše odpovídala bráně, která potřebuje kromě zmíněných protokolů komunikovat také s „normální“ sítí a tedy musí mít implementován protokolový zásobník TCP/IP. Zařízení, která nejsou bránou a nepotřebují TCP/IP, mají trochu jinou horní část protokolového zásobníku. K tomuto tématu se ještě vrátíme v sekci 3.5 o ZigBee na straně 56.

Na obrázku 3.2 jsme si mohli všimnout, že v IoT si nevystačíme jen s protokoly známými z webových služeb, ale potřebujeme i jiné. Na následujících stranách se stručně podíváme na některé z nich, dva budou následně probrány důkladněji.

### 3.3.1 IoT protokoly vyšších vrstev

Na aplikační vrstvě může být nám známý protokol HTTP, případně REST, nebo něco jiného. Možností je hodně a bohužel vzájemně nekompatibilních. Jejich funkcionalita se až tak úplně nekryje s aplikační vrstvou TCP/IP, proto se často můžeme setkat spíše s pojmem *datové protokoly*.




Obrázek 3.2: Srovnání webového protokolového zásobníku a protokolových zásobníků pro IoT (volně podle [10, 14, 20])

**MQTT.** Protokol MQTT (MQ Telemetry Transport) vznikl ve spolupráci společností IBM a Eurotech, později (v roce 2014) byl standardizován organizací OASIS, díky čemuž se stal otevřeným protokolem. Je to aplikační protokol zajišťující posílání jednoduchých zpráv, jejich organizaci do front a práci s těmito frontami (písmeno „q“ v názvu znamená „queue“, fronta). Je určen pro asynchronní M2M (machine-to-machine) komunikaci, především pro sítě senzorů, čímž je dána vhodnost pro IoT sítě. Posílané zprávy jsou čistě textové, další formáty nejsou podporovány.[11] Projekt najdeme na [29].

Jak jsme viděli v tabulce 3.1, protokol MQTT podporuje pouze komunikaci typu Publish -Subscribe, tedy v síti máme centrální prvek zprostředkovávající komunikaci mezi ostatními zařízeními v síti. Protože na nižší vrstvě je používán protokol TCP, lze zprávy zabezpečit šifrováním pomocí TLS.[20]

Protokolu MQTT se budeme podrobněji věnovat v sekci 3.4 MQTT.

**DDS.** Ostatní zde uvedené protokoly jsou standardizovány organizacemi IETF nebo OASIS, za protokolem DDS (Data Distribution Service) stojí standardizační organizace OMG[1]. Existují také otevřené varianty tohoto protokolu, známá je například OpenDDS[2], za kterou stojí společnost Object Computing[3].

Stejně jako MQTT, také DDS podporuje pouze komunikační model Publish-Subscribe. Je určen pro realtimovou komunikaci v decentralizované peer-to-peer síti (což je pro Publish-Subscribe netypické, v tom se od MQTT liší), nepředpokládá se žádný centrální prvek. Data ve formě čistého textu jsou posílána v úsporném binárním formátu, přesto tento protokol generuje více provozu na síti než MQTT.[20]

Protože není k dispozici centrální prvek, je pro uzly v síti k dispozici mechanismus dynamického zjišťování dostupných dat a jejich publisherů. Protokol definuje prostor GDS (Global Data Space), což je jakýsi virtuální prostor, do kterého se připojuje každý publisher a subscriber a kde dochází k předávání dat.[10] GDS můžeme chápat jako určitou formu adresace a směrování zpráv.

Na transportní vrstvě lze využít jak protokol TCP, tak i UDP. Při použití TCP se šifruje pomocí protokolu TLS, při použití UDP máme k dispozici DTLS.[20]

**XMPP.** Protokol XMPP (eXtensible Messaging and Presence Protocol) byl od začátku vyvíjen organizací IETF jako otevřený protokol (základním dokumentem je RFC 6120). Umí přenášet krátké zprávy, je použitelný pro chat, hlasové i vi-

[1] https://www.omg.org/ [cit. 2022-07-02]
[2] https://opendds.org/ [cit. 2022-07-02]
[3] https://objectcomputing.com/ [cit. 2022-07-02]

deohovory, ale také pro IoT sítě. Zvládá přenosy i jiných než textových dat a na transportní vrstvě používá protokol TCP, tedy navazuje spojení. Oproti MQTT přenáší poněkud více dat, je tedy méně šetrný k síti, a také více zatěžuje výpočetní čipy na zařízeních, na kterých běží (je výpočetně náročnější). Textové zprávy totiž přenáší ve formátu XML s poněkud „upovídanými“ značkami a s dodatečnými metainformacemi, což se dá alespoň částečně řešit komprimací. Nicméně komprimace vyžaduje určitý výpočetní výkon a čas, což také pro IoT není nejlepší.[2, 20]

XMPP navazuje spojení přes TCP a komunikaci lze šifrovat protokolem TLS. Kromě toho umí řešit autentizaci – ověřování klienta a serveru – pomocí SASL.[20] Výbava vpodstatě odpovídá tomu, co očekáváme od IM klienta (na první pohled to tak nevypadá, ale využití pro IM má s IoT hodně společného).

**CoAP.** Protokol CoAP (Constrained Application Protocol) je dalším dílem organizace IETF, tedy opět otevřený protokol zveřejněný jako RFC 7252. Oproti XMPP není určen pro posílání komplexnějších dat, ale zato generuje poněkud méně provozu na síti. Nicméně latence je při použití tohoto protokolu spíše větší než například u MQTT.[20] V RFC 7252 je uvedeno doporučené použití pro M2M komunikaci, zejména automatizaci správy energií v domech.

CoAP je zajímavý i tím, že (podobně jako HTTP) je použitelný pro REST architekturu, ostatně v IETF na něm pracovala právě pracovní skupina zabývající se RESTful prostředími. Na rozdíl od HTTP (a taky většiny jiných zde jmenovaných aplikačních protokolů) používá na transportní vrstvě protokol UDP.[10]

Protokoly XMPP a CoAP pocházejí od téže standardizační instituce (IETF), proto má smysl, že jejich vlastnosti se liší. Oba protokoly sice podporují komunikaci jak typu Request-Response, tak i Publish-Subscribe, ale první na transportní vrstvě používá protokol TCP se spojovou komunikací, druhý protokol UDP bez navazování spojení a potvrzování paketů. V obou protokolech se používají potvrzovací zprávy, proto CoAP v sobě nese i tu funkcionalitu, která by jinak byla zajištěna na nižší vrstvě – plné řízení komunikace včetně znovuposílání ztracených zpráv.[10] To je ostatně důležité i pro využití CoAP v REST architektuře.

Princip zpráv je podobný HTTP: existuje více různých typů zpráv, na ně se odpovídá zprávou obsahující stavový kód. Protože nosný protokol UDP nepodporuje potvrzování, musí si CoAP zajistit spolehlivost jinak. Klient může posílat zprávy buď s vyžádáním potvrzení nebo bez nutnosti potvrzení, potvrzuje se zprávou ACK a dále může následovat samotná zpráva s odpovědí. Každá zpráva má v záhlaví token, který slouží k párování dotazu a odpovědi, a dále identifikátor

určený ke zjišťování duplikátů zpráv, aby se zbytečně neposílalo více odpovědí na tentýž dotaz.[2]

Zatímco XMPP umí přenášet širokou škálu datových formátů, CoAP přenáší binární data. Protože se na nižší vrstvě napojuje na protokol UDP, pro šifrování provozu je k dispozici protokol DTLS, nicméně se dá použít i IPSec, což u IoT protokolů není obvyklé. CoAP podporuje multicastovou komunikaci (zprávy lze posílat buď jako unicast, nebo jako multicast), jenže to má i nevýhodu v horších možnostech zabezpečení, end-to-end šifrování je prakticky problém.[20] Z výše uvedeného vyplývá ještě jeden fakt: použití protokolu UDP na transportní vrstvě je logické kromě jiného i kvůli multicastu, protože TCP dokáže přenášet pouze unicast.

**AMQP.** Otevřený protokol AMQP (Advanced Message Queuing Protocol) byl standardizován organizací OASIS stejně jako MQTT. Byl navržen pro rychlou výměnu velkého množství zpráv v heterogenním M2M prostředí, včetně továren.[10, 20] S protokolem MQTT mají mnoho společného, včetně běhu nad protokolem TCP, používání front (třebaže AMQP dokáže pracovat i bez centrálního prvku, podobně jako DDS) nebo QoS – například lze určit doručení at-most-once, at-least-once či exactly once.

Princip směrovní v AMQP je založen na existenci komponenty *exchange* (ústředna) a řady front. Různé fronty jsou určeny pro různé subscribery. Zprávy doručené do centrálního prvku (pokud v síti existuje) jsou vyhodnoceny v exchange podle předem stanovených pravidel a následně umístěny do konkrétních front. Z front jsou pak zprávy postupně posílány subscriberům. Tento model se označuje jako point-to-point, je to vpodstatě upravený model Publish-Subscribe.[2]

Díky tomu, že se navazuje TCP spojení, lze komunikaci šifrovat pomocí TLS. Dále je použitelný protokol IPSec a autentizace je podporována díky SASL jako u XMPP. Posílají se binární data, není zde zabudována podpora různých formátů. Mezi IoT protokoly patří AMQP spíše k těm náročnějším na šířku pásma a výkon zařízení.[20]

**OPC UA.** Protokol OPC UA (OPC Unified Architecture) je M2M standard určený pro průmysl v současné době podporovaný většinou výrobců průmyslových zařízení. Určuje především formát posílaných zpráv (to má společné s výše jmenovanými protokoly) a je „service oriented“ – orientovaný na služby. To znamená, že v síti jsou definovány služby, které server poskytuje a klient může od serveru požadovat. Komunikuje se výhradně zabezpečeným kanálem, naprosto každé zařízení má svůj certifikát, který je jednoznačně identifikuje.[51] Tento protokol je zde

zmíněn jen pro úplnost, jeho využití je mimo zaměření tohoto dokumentu, tedy se jím dále nebudeme zabývat.

**Poznámka**

Shrneme si možnosti zabezpečení aplikačních protokolů pro IoT: jak je výše uvedeno, komunikaci těchto protokolů lze šifrovat pomocí TLS (verze 1.2 je dokumentována ve standardu RFC 5246, verze 1.3 v RFC 8446) nebo DTLS (Datagram TLS, verzi 1.2 najdeme v RFC 6347, verzi 1.3 v RFC 9147) podle toho, jestli dotyčný protokol běží nad TCP nebo UDP.

Některé protokoly podporují také IPSec, jehož základním dokumentem je RFC 6071, dva také umí ověřovat komunikující strany s využitím SASL (Simple Authentication and Security Layer, RFC 4422).

V tabulce 3.2 najdeme srovnání vybraných aplikačních IoT protokolů. Oproti předchozí tabulce jsou zde i použitelné metody a formáty posílaných zpráv.

| **Vlastnost** | **RESTful** | **MQTT** | **XMPP** | **AMQP** | **DDS** | **CoAP** |
|---|---|---|---|---|---|---|
| L4 protokol | TCP | TCP | TCP | TCP | TCP/UDP | UDP |
| Model | Req-Resp | Pub-Sub | Req-Resp Pub-Sub | Point-to-Point Pub-Sub | Req-Resp Pub-Sub | Pub-Sub |
| Metody | Post, Put, Delete, Get | Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close | Get, Post, Put, Set, Result | Consume, Deliver, Publish, Get, Select, Ack, Delete, Nack, Recover, Reject, Open, Close | Write, Read, Take, Dispose, Wait | Post, Put, Delete, Get CON, NON, ACK, RST |
| Šifrování | TLS | TLS | TLS, SASL | TLS, IPSec, SASL | TLS/DTLS | DTLS, IPSec |
| QoS | ano | ano | ne | ano | ano | ano |
| Formát zpráv | plain text, XML, JSON,... | plain text | chat,... | binary | ASCII, binary | binary |

Tabulka 3.2: Srovnání některých aplikačních IoT protokolů (podle [2, 10, 20])

### 3.3.2 IoT protokoly nižších vrstev pro místní sítě

Především v souvislosti s níže uvedenými protokoly se můžeme setkat se zkratkou *WSN* (Wireless Sensor Networks). Tento pojem zahrnuje různě velké sítě (od PAN přes LAN až po WAN), které mají jedno společné: propojují zařízení do M2M sítě, při běžné činnosti se nepočítá s interakcí uživatele. V těchto sítích najdeme různé senzory či jiná zařízení komunikující čistě mezi sebou.[20]

Na fyzické vrstvě IoT pracují obvykle v některém ISM pásmu (Industrial, Scientific, Medical; bezlicenční pásmo), většinou na frekvencích kolem 2,4 GHz.[27]

**Bluetooth LE.** Většina lidí zná Bluetooth jako bezdrátovou technologii pro komunikaci na krátkou vzdálenost (tedy WPAN, Wireless PAN), která je implementovaná prakticky v každém mobilu, notebooku, nositelné elektronice a jiných malých zařízeních. Bluetooth LE (Bluetooth Low Energy, také BLE nebo Bluetooth Smart) je varianta Bluetooth upravená tak, aby se co nejvíc snížila spotřeba komunikujícího zařízení. Bluetooth je standardizován jako IEEE 802.15.1, Bluetooth LE se objevilo jako varianta ve verzi Bluetooth 4.0.[27]

Bluetooth LE komunikuje ve stejném frekvenčním rozsahu jako původní standard, tedy v ISM pásmu kolem 2,4 GHz, vždy se navazuje spojení (původní Bluetooth umožňuje komunikaci i bez navázání spojení), může pracovat synchronně i asynchronně. Ovšem místo 40 kanálů širokých 2 MHz dělí pásmo do 79 užších kanálů (šířka jen 1 MHz). To celkem nevadí, protože tento protokol je typicky používán pro přenosy malého množství dat na krátkou vzdálenost. Standard popisuje vrstvu L1 a MAC podvrstvu vrstvy L2, podobně jako například IEEE 802.11 (Wi-fi).

Jedno zařízení je v roli master (navazuje a řídí spojení, případně se může starat o napájení), ostatní jsou v roli slave: například mobilní telefon je v roli master a chytré hodinky či náramek v roli slave. Kromě využití pro komunikaci drobných chytrých zařízení s mobilním telefonem se s tímto protokolem můžeme setkat například ve zdravotnictví, sportu a dalších oblastech.[27, 20]

**IEEE 802.15.4 a ZigBee.** Protokol IEEE 802.15.4 (Low-rate WPANs) specifikuje fyzickou vrstvu a MAC podvrstvu spojové vrstvy, je určen pro M2M zařízení s nízkou spotřebou a na relativně krátké vzdálenosti. V některých aspektech je blízký protokolu IEEE 802.11 (Wi-fi), například používáním přístupové metody CSMA/CA (třebaže trochu jinak pojaté), ale v mnohém se liší. Například kromě pásma 2,4 GHz používá také pásma 915 MHz a 868 MHz – delší vlny (tj. nižší frekvence) jsou energeticky úspornější ve srovnání s kratšími při komunikaci na tutéž

vzdálenost, stačí nižší vysílací výkon.[7] Jsou posílány pouze velmi krátké zprávy (do MAC rámce lze zapouzdřit maximálně 102 B), záhlaví je také úspornější jež u Wi-fi.[16]

Protokol ZigBee je se specifikací IEEE 802.15.4 často ztotožňován, třebaže nejde o synonyma. ZigBee si z této specifikace bere funkcionalitu nižších vrstev a přidává vyšší vrstvy, zejména inteligentní směrování v mesh síti. Rozlišuje se několik typů zařízení podle toho, co konkrétně mají v síti dělat. Zařízení mohou plnit roli ZigBee routeru, který nemusí být nutně jen zdrojem nebo cílem dat, ale umí je i přeposílat mezi jinými ZigBee zařízeními.[42]

Účelem ZigBee je přeposílat zprávy v malých WSN sítích v domácnostech či malých firmách, například propojovat chytrou elektroniku v rámci bytu, zprostředkovávat řízení chytrých žárovek, přeposílat data z různých senzorů v domě.

Protokol ZigBee byl původně pod správou organizace ZigBee Alliance, teď jde o Connectivity Standards Alliance (CSA).[53] ZigBee je otevřený standard s více různými implementacemi.[7]

Protokolům IEEE 802.15.4 a ZigBee se budeme podrobněji věnovat v sekci 3.5 ZigBee.

**Z-Wave.** Tento protokol je brán jako nejbližší (nicméně proprietární) konkurent protokolu ZigBee, je v principu hodně podobný. Je také určen pro M2M LoWPAN sítě se stejnou cílovou skupinou (WSN sítě), umí směrovat mezi Z-Wave zařízeními, posílané PDU jsou spíše krátké a mají záhlaví úspornější než u Wi-fi. Na

| **Vlastnost** | **NFC** | **BLE** | **ZigBee** | **Z-Wave** |
|---|---|---|---|---|
| Topologie | P2P síť | hvězda | hvězda, strom, mesh,... | mesh |
| Frekvenční pásmo | 13,56 MHz | 2,4 GHz | 2,4 GHz, 915 MHz, 868 MHz | Evropa: 868 MHz |
| Dosah | 10 cm | až 100 m | až 100 m (pro 2,4 GHz), jinak až 1 km | 30 m |
| Max. uzlů v síti | 2 | 65 535 | 65 000 | 232 |
| Propustnost | 106–424 kb/s | 2 Mb/s | 250 kb/s | 200 kb/s |
| Šifrování | – | AES | AES | AES |

Tabulka 3.3: Srovnání některých LoWPAN technologií (podle [20])

fyzické vrstvě vysílá v rozdílných pásmech v Evropě (868,42 MHz) a v Americe (908,42 MHz). V ostatních částech světa to je obvykle jedno z těchto pásem, ale jsou výjimky (například v Izraeli to je 916 MHz), ovšem vždy pod 1 GHz.[20, 52]

Fyzickou vrstvu a MAC podvrstvu bere Z-Wave z ITU-T G.9959 a nad ní staví funkcionalitu vyšších vrstev, v čemž se také podobá ZigBee. Stejně jako u standardu ZigBee, zařízení pracující podle standardu Z-Wave nepotřebují TCP/IP, využívají jen funkcionalitu toho, co přímo nabízí Z-Wave – tedy kromě případné brány do jiného typu sítě, kde musí být také podpora zásobníku TCP/IP včetně přizpůsobovací vrstvy.[52]

**Vyrovnávací vrstva.** 6LoWPAN se používá jako podpůrný protokol ve WSN sítích od IETF, je popsaný v RFC 4944 (Transmission of IPv6 Packets over IEEE 802.15.4 Networks). Doplňuje specifikaci IEEE 802.15.4 o možnost připojení do IPv6 sítí.

Zatímco při propojování IoT zařízení navzájem si vystačíme s příslušným základním protokolem (například ZigBee), tak pokud je potřeba brána do běžné TCP/IP sítě, potřebujeme „překladatele“. 6LoWPAN umožňuje propojit IoT síť do běžné sítě určené pro IPv6 pakety (od toho číslice 6 na začátku názvu protokolu), přičemž zachovává výhody IoT komunikace – především v tom, že se v mezích IPv6 snaží o úspornou komunikaci co se týče množství přenášených dat.[14, 16]

### 3.3.3 WAN sítě pro IoT

Technologie Wi-fi, BLE, ZigBee, Z-Wave a podobné využíváme tehdy, když celá síť má být pod naší plnou kontrolou. To však nejde, pokud se propojovaná zařízení nacházejí na příliš velkém prostoru, daleko od sebe, do IoT sítě chceme zapojit i mobilní zařízení, nebo je jiný důvod, proč síť ve vlastní režii není použitelná.

Pro IoT lze na nižších vrstvách použít vedle běžných bezdrátových WAN sítí (v současné době hlavně LTE, 5G) také bezdrátová řešení určená pro větší vzdálenosti, tedy tzv. *LPWAN sítě* (Low Power WAN). Z nich jsou nejznámější Sigfox, LoRaWAN a NB-IoT. Na rozdíl od malých IoT sítí zde potřebujeme poskytovatele služby (operátora), který provozuje příslušnou síť ve všech oblastech, ve kterých máme zařízení k propojení – to mají LPWAN sítě s klasickými mobilními sítěmi společné, dokonce včetně možnosti roamingu. Naše zařízení pak komunikují příslušným protokolem se sítí operátora a ten pak slouží jako prostředník (virtuální brána) do běžné sítě s protokolem TCP/IP. Takže pokud si chceme třeba v mobilním telefonu zobrazit momentální stav čidel rozmístěných na poli, nemusíme mít

v mobilu podporu Sigfoxu, pouze se potřebujeme dostat do příslušného webového rozhraní či aplikace.

Tento typ sítí má propojovat zařízení (příp. senzory) rozprostřená na velkém prostoru, například na ulicích města (parkovací senzory, chytré semafory apod.), na polích, vinohradech, v lesích, v továrních halách,... Podobně jako u běžných mobilních sítí, i zde zařízení přímo komunikuje s nejbližší základnovou stanicí patřící k síti operátora. Zákazník má ke svému zařízení přístup přes webové rozhraní, speciální aplikaci nebo API (pro vlastní aplikace).[26]

Pro připojení zařízení do sítě Sigfox nebo LoRaWAN nepotřebujeme SIM kartu, ale každé zařízení má jedinečný identifikátor. Tento identifikátor zadáváme při registraci zařízení do sítě operátora.[36, 39] Princip je podobný virtuálním SIM, které se postupně začínají prosazovat u klasických mobilních sítí.

**Sigfox.** Za touto technologií stojí organizace ETSI, třebaže standardizační proces není plně dokončen[39]. Ze zde uvedených LPWAN standardů má největší dosah: mezi zařízením a základnovou stanicí může být až 40 km mimo zástavbu, v městském prostředí až 10 km. U přímé viditelnosti udává zdroj [39] až 200 km. V různých částech světa se používají rozdílná vysílací pásma, v Evropě to je ISM pásmo kolem 868 MHz, šířka kanálu je pouze 100 Hz (ultra-narrow band) a v pásmu je celkem 400 navzájem ortogonálních kanálů. Používá se modulace BPSK.[20, 26]

Pro koncová zařízení (směr uplink) je stanoven limit 140 zpráv denně, zpráva na uplinku může nést maximálně 12 B dat. Downlink zpráva může následovat pouze po uplinku (vyžádání downlinku je na koncovém zařízení) a navíc jsou podporovány pouze 4 downlink zprávy za den. Downlink zpráva může nést maximálně 8 B dat.[26, 39] Limity pro počet a velikost zpráv se mohou zdát velmi přísné (vpodstatě jsou – tím je limitováno použití Sigfoxu), pro senzory to však obvykle postačuje.

Pro zajištění integrity je každá uplink zpráva posílána ve více kopiích v různých kanálech v různých směrech, zařízení není fixováno na jedinou základnovou stanici. Pokud signál přijme více základnových stanic zároveň, použije se algoritmus pro řešení duplicit. Díky tomu se snadno řeší lokalizace a mobilita zařízení: u signálu přijatého více základnovými stanicemi se zjistí časové rozdíly (algoritmus RSSI), čímž zjistíme poměr vzdáleností zařízení k jednotlivým základnovým stanicím, a co se týče mobility, není vyžadován handoff (předávání mezi základnovými stanicemi).[26] Zařízení vpodstatě komunikuje se všemi dosažitelnými základnovými stanicemi zároveň.

Šifrování není v Sigfoxu implementováno, nicméně je možné použít vlastní end-to-end řešení na aplikační vrstvě. Nevýhodou je, že tato funkcionalita není sou-

částí mechanismu, ale na druhou stranu při volbě vhodného zabezpečení máme šifrování plně pod kontrolou. Alespoň integrita je řešena: zprávy jsou digitálně podepsány.[39]

V České republice je operátorem pro tuto síť společnost SimpleCell Networks, do budování sítě Sigfox se zapojil T-Mobile. Služba není určena pro koncové zákazníky, ale pro výrobce zařízení, kteří tuto síť využívají ve svých produktech. Momentálně je propustnost této sítě 100 b/s, což IoT zařízením stačí. Vysílací výkon je do 25 mW a každé komerční zařízení musí být certifikováno, aby tuto podmínku splňovalo.[39] Na webu operátora lze najít i mapu pokrytí signálem.

Na trhu jsou k dostání čipy s podporou Sigfoxu, proto kromě toho, že si lze zakoupit hotové zařízení, je možné vyrobit zařízení vlastní. Existují možnosti například pro Arduino. Se zařízením pak komunikujeme přes síť operátora prostřednictvím HTTPS REST nebo jinak, dokonce lze použít i e-mail. Data jsou ve formátu JSON.[39]

Sigfox vychází ze zde popisovaných LPWAN technologií finančně zřejmě nejlépe a signál má největší dosah. Určitým handicapem je problém s obousměrnou komunikací, takže tato technologie není vhodná pro každé IoT zařízení. Je vhodná například pro monitoring stavu prostředí nebo cokoliv dalšího, kde není potřeba častý downlink a u uplink zpráv od zařízení nepotřebujeme potvrzování. Výhodou může být snadno proveditelný roaming v případě, že zařízení putuje do zahraničí.[3, 40]

**LoRaWAN.** Tato síť, za kterou stojí LoRa Alliance, používá nelicencované ISM pásmo (v Evropě 868 MHz) s šířkou kanálů větší než u Sigfoxu: 125 kHz nebo 250 kHz. Moduluje patentovanou metodou CSS, což (na rozdíl od Sigfoxu) umožňuje snadněji implementovat obousměrnou komunikaci, ovšem v half-duplexu stejně jako u jiných podobných technologií. Díky nízkým frekvencím a kódové modulaci (stejně jako u Sigfoxu) je signál velmi odolný proti rušení a odposlechu.[26]

Tato síť implementuje fyzickou vrstvu a MAC podvrstvu. Na vývoji fyzické vrstvy pracovala francouzská firma Semtech, která si také patentovala modulační metodu CSS. Nicméně vedle proprietární fyzické vrstvy jsou zbývající části standardu – vyvinuté pod záštitou LoRa Alliance – otevřené.[3]

V uvedené kódové modulaci je možno volit mezi šesti různými nastaveními (spreading factors SF7–SF12), čímž zároveň s volbou šířky kanálu měníme poměr mezi dosahem a propustností. V závislosti na volbě parametru spreading factor a šířky pásma může být propustnost od 300 b/s do 50 kb/s, což je o více než dva řády výše než u Sigfoxu. Dosah se udává 5 km (český operátor na [36] udává jen

2 km) ve městě a až 20 km mimo zástavbu. Energetická náročnost této technologie je nízká, podobně jako u Sigfoxu.[26]

Architektura sítě je odlišná od jiných sítí podobného určení. Koncové zařízení komunikuje s LoRa bránou (jednou nebo více) pomocí protokolu LoRa, brána pak běžnými protokoly TCP/IP s podsunutým Ethernetem, LTE, atd. podle možností komunikuje s LoRa síťovým serverem (opět může více bran s jedním serverem). K serveru se pak připojují běžní klienti přes klasickou síť. Bránu si může koncový zákazník vyrobit sám – což přichází v úvahu, když poblíž žádná existující není – nebo využít službu jiné firmy či operátora.[3] Vlastní brána má smysl i tehdy, když přenášená data jsou natolik citlivá, že je nechceme svěřit někomu jinému.

V případě vlastní brány a hlavně LoRa serveru potřebujeme vhodný síťový hardware (dá se sehnat, například pro Raspberry Pi, Mikrotik nebo jiné podobné platformy) a také software. U softwaru máme na výběr mezi různými řešeními, existují také odlehčené varianty zdarma. Nejznámější projekty jsou TTN (The Things Network, software je TTS – The Things Stack)[4], ChirpStack[5] nebo lorawan-server[6]. Hardware a software je třeba vybrat tak, ať jsou kompatibilní.

Stejně jako u Sigfoxu, zařízení není svázáno s jedinou bránou (zde nemluvíme o základnových stanicích), ale může komunikovat s více LoRa bránami najednou. Duplicita zpráv se řeší podobně jako u Sigfoxu, pro lokalizaci zde máme jiný protokol – TDoA, nicméně v principu funguje podobně, a handoff u mobilních zařízení se taktéž nemusí řešit.[26]

Ve zprávě lze poslat až 243 B, není stanoven limit na počet zpráv (pokud ho nestanoví operátor) a komunikace je šifrovaná, používá se algoritmus AES.[26]

Českým operátorem sítě LoRaWAN jsou České Radiokomunikace, informace najdeme na [36] a odkazech vedoucích ze zmíněného webu na https://www.iotport.cz/.

Technologie LoRaWAN z popsaných LPWAN technologií vyniká v průměru nejnižší energetickou spotřebou, nabízí obousměrnou komunikaci a pro IoT dostačující přenosové rychlosti. Dosah je mírně horší. Výhodou je otevřenost většiny specifikace a široké možnosti personalizace včetně vytvoření vlastní LoRa infrastruktury.[3, 40]

**NB-IoT.** Technologie NB-IoT (Narrow-Band IoT) je úzkopásmová technologie, za kterou stojí organizace 3GPP. Oproti Sigfoxu a LoRaWAN staví na klasických mobilních sítích (GSM a LTE), což se týká jak použitých frekvenčních pásem (také pod

[4] Web projektu je https://www.thethingsnetwork.org/ [cit. 2022-08-18]
[5] Web projektu je https://www.chirpstack.io/ [cit. 2022-08-18]
[6] Web projektu je https://github.com/gotthardp/lorawan-server [cit. 2022-08-18]

1 GHz, ale různě podle konkrétních oblastí světa), tak i způsobu využití těchto pásem. Využívá infrastrukturu LTE, jen je potřeba upgrade. Šířka kanálu je 200 kHz, což odpovídá právě situaci v GSM a LTE, propustnost dosahuje až 200 kb/s. Používá multiplex FDMA na uplinku a OFDMA na downlinku, modulace QPSK. Dosah je v městské zástavbě 1 km, mimo zástavbu až 10 km.[40]

Funkčně se NB-IoT velmi podobá LTE, ale vynechává vše, co pro IoT svět není důležité. Tím tato technologie dokázala oproti LTE stáhnout energetickou náročnost, ale pořád zůstává o něco vyšší než u Sigfoxu a LoRaWAN. Signál není tak odolný proti rušení a koncová zařízení se musejí asociovat s jednou základnovou stanicí, jako v LTE. Oproti zmíněným konkurentům může být ve zprávě přenášeno výrazně více dat (až 1600 B), šifrování je k dispozici v rámci LTE funkcionality.

V České republice budují NB-IoT síť operátoři O2 a Vodafone (jednoduše tak, že ve svých LTE sítích provedli softwarový upgrade).

NB-IoT má nevýhodu v tom, že každé zařízení musí mít SIM kartu, ať fyzickou nebo eSIM – i v tom odpovídá své technologické základně: LTE. Finančně vyjde dráž než Sigfox a LoRaWAN, energeticky je taktéž méně výhodná, naopak výhodou je možnost přenášet výrazně větší množství dat, a to v obou směrech.[3, 40]

## 3.4 MQTT

MQTT je otevřený protokol určený pro M2M komunikaci, jehož specifikaci udržuje organizace OASIS[30]. Existuje také odlehčená varianta protokolu MQTT-SN určená pro senzorové sítě, jejíž specifikaci najdeme také na webu [30][7]. Zatímco „plnohodnotný“ MQTT se dá očekávat na zařízeních s implementací protokolového zásobníku TCP/IP, „odlehčený“ MQTT-SN najdeme typicky na zařízeních s nízkou spotřebou propojených některou WSN sítí, například ZigBee.

V předchozí sekci jsme se o protokolu MQTT dozvěděli, že je to aplikační M2M protokol nezávislý na protokolech nižších vrstev. Používá komunikační model Publish-Subscribe a v síti se předpokládá centrální zařízení (broker, controller, MQTT server). Publikující zařízení (publisher, zdroj dat) přispívá do tzv. tématu, odebírající zařízení (subscriber, cíl dat) se přihlašuje k odběru pro konkrétní téma. Téma si můžeme představit jako časopis – někdo do něj přispívá, tedy píše články, někdo zase naopak má časopis předplacený a odebírá ho.

[7] Konkrétně v dokumentu https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf

### 3.4.1 Témata

Témata jsou brokerem realizována jako fronty zpráv, které přišly od publisherů a mají být přeposlány subscriberům. Jeden publisher může publikovat do více témat, jeden subscriber může odebírat z více témat, zařízení může plnit roli publishera i subscribera zároveň. Každé téma má u brokera svou frontu. Navíc jsou témata organizována hierarchicky.[11]

**Příklad**

Můžeme si to představit třeba takto: chytrý dům je vybaven několika teplotními čidly – venkovními na různých stranách domu (abychom zachytili teplotu na slunci i ve stínu v různou denní dobu) a pak vnitřními teploměry v různých místnostech. Také můžeme mít teploměr například ve venkovním bazénu. Témata si sestavujeme a nazýváme sami, taky si určujeme jejich hierarchii, pro tento případ bychom například mohli shrnout vnitřní vs. vnější čidla do skupin:

```
teplomery/venku
teplomery/uvnitr
```

Do těchto skupin pak zařadíme konkrétní teploměry nebo čidla v komplexnějších IoT zařízeních, například:

```
teplomery/venku/zapad
teplomery/venku/vychod
teplomery/venku/sever
teplomery/venku/bazen
...
teplomery/uvnitr/obyvak
teplomery/uvnitr/koupelna
teplomery/uvnitr/sever
...
```

Jedním řetězcem lze reprezentovat i více než jedno téma, můžeme používat zástupné znaky. Používáme dva: symbol „`+`“ zastupuje jednu úroveň, kdežto symbol „`#`“ může zastupovat více úrovní.[31] Například

```
teplomery/venku/+
teplomery/#
teplomery/+/sever
```

První zmíněný řetězec určuje všechny venkovní teploměry, druhý všechny teploměry (ať už venku nebo vevnitř), třetí teploměry ukazující teplotu na severu (ze

seznamu vidíme, že půjde o dva teploměry, jeden venkovní a jeden vnitřní).

Odběratelé (subscribers) pro teplotu mohou být například obrazovky, na kterých se průběžně zobrazuje teplota v různých částech domu, nebo chytrá zařízení, jejichž úkolem je na změny teploty reagovat. Například může jít o automatickou regulaci topení či klimatizace, automatické sklápění žaluzií, řízení větrání apod.

Kromě témat, která si vytvoří každý uživatel podle svého uvážení, existují i systémová témata, jejichž hierarchie začíná řetězcem `$SYS`. Zatímco k uživatelským tématům mohou mít klienti i právo zápisu (pokud jsou pro dané téma v roli publishera), u systémoých témat to nejde, jsou pro všechny kromě brokera „read-only“.[11, 31]

### 3.4.2 Relace a zprávy

Dále se budeme zabývat především protokolem MQTT standardizovaným roku 2014 pod OASIS Open. Informace najdeme na https://www.oasis-open.org/org/, o příslušném Technical Commitee a samotném standardu na [30].

MQTT pracuje na aplikační vrstvě nad protokolem TCP. To znamená, že se vždy navazuje spojení a zprávy jsou posílány v rámci relace. Tato relace může trvat krátce, nebo i velmi dlouho, podle potřeby. Spojení lze šifrovat pomocí TLS (RFC 5246), také se používají WebSockets (RFC 6455) pro zajištění obousměrné komunikace; HTTP není v tomto směru symetrický – klient se dotazuje a server odpovídá, ale u IoT chceme často symetričtější přístup.

Publisher posílá data do tématu pomocí zprávy PUBLISH. MQTT rozlišuje tři úrovně *QoS*, které určuje publisher v záhlaví zprávy PUBLISH:[31]

- QoS 0 (at most once delivery) – takto publikovaná data se nepotvrzují, publisher je jednou pošle a dál už se o ně nestará,
- QoS 1 (at least once delivery) – publisher po odeslání očekává potvrzení zprávou PUBACK, čímž je ujištěn o tom, že publikovaná data dorazila (alespoň jednou), až do chvíle potvrzení má uloženu odeslanou zprávu včetně toho, že zatím je nepotvrzená; broker maže zprávu po přeposlání subscriberům a pak odešle PUBACK, následně smaže zprávu i publisher,
- QoS 2 (exactly once delivery) – účelem je nejen zajistit, že publikovaná data dorazí, ale také odstranit případné duplicity; místo PUBACK se používají zprávy PUBREC (posílá broker: data uložena a poslána subscriberům), PUBREL (posílá publisher: beru na vědomí, můžeš vymazat), broker vymaže uloženou zprávu a pošle PUBCOMP (beru na vědomí, taky mažu).

Protokol MQTT je postaven na krátkých *zprávách*, které slouží různým účelům. Jedná se (kromě dalších) zejména o tyto zprávy:[11, 31]

- CONNECT – žádost o vytvoření relace, posílá klient brokerovi,
- CONNACK – broker odpovídá klientovi na zprávu CONNECT, potvrzuje vytvoření relace,
- PUBLISH – klient v roli publishera posílá data k danému tématu,
- PUBACK – broker potvrzuje publisherovi, že přijal zprávu PUBLISH s nastavením QoS 1 (oznamuje, že zpráva byla přeposlána subscriberům a smazána),
- PUBREC – broker potvrzuje publisherovi, že přijal zprávu PUBLISH s nastavením QoS 2 (oznamuje, že zpráva byla přeposlána subscriberům),
- PUBREL – publisher reaguje na PUBREC (bere na vědomí, že publikovaná data byla přeposlána), posílá tuto zprávu brokerovi,
- PUBCOMP – poslední zpráva při publikování v QoS 2, broker reaguje na zprávu PUBREL a oznamuje, že původní zprávu PUBLISH už smazal,
- SUBSCRIBE – klient si u brokera objednává téma, tedy chce odebírat data do tohoto tématu posílaná,
- SUBACK – broker tímto potvrzuje zprávu SUBSCRIBE,
- UNSUBSCRIBE – klient se u brokera odhlašuje z odběru tématu,
- UNSUBACK – broker tímto potvrzuje zprávu UNSBUSCRIBE.

Další typy zpráv s jejich popisem najdeme především ve specifikaci protokolu, která je zveřejněna na [31].

Textové řetězce posílané ve zprávách (což jsou obvykle názvy témat, včetně zástupných znaků) mají být podle standardu kódovány v UTF-8.[31]

Podrobný popis formátu zpráv je svým rozsahem nad možnosti této publikace, zde se zaměříme jen na některé aspekty, které mohou být využity pro témata probíraná v následující kapitole.

**Postup (Určení platnosti objednávky tématu [31])**

Při připojování tedy posílá klient zprávu CONNECT, v níž nás nyní zajímá příznak *Clean Start* (v [11] je nazýván Clean Session Flag). Pokud je nastaven na 1, broker po přijetí zprávy odstraní veškeré informace o tomto klientovi, které získal během předchozích připojení (včetně dřívějších objednávek), tedy „vyčistí relaci“. Jestliže je však nastaven na 0, dříve získané informace zůstanou zachovány, včetně dřívějších objednávek: klient tedy plynule naváže na své předchozí fungování.

Takže pokud klient při zahájení relace použije příznak Clean Start, předpokládá se, že před koncem relace pošle zprávu UNSUBSCRIBE pro svá objednaná témata. Pokud tento příznak při zahájení relace nepoužije, veškeré jeho objednávky zůstávají v platnosti i po ukončení spojení a zahájení následující relace.

Příznak Clean Start je jeden z několika tzv. Connection Flags, tedy příznaků připojení ve zprávě CONNECT. Z dalších můžeme jmenovat Will Flag, který stanovuje, jak se má broker zachovat k subscriberům odebírajícím téma, s jehož publisherem nečekaně ztratil spojení, nebo příznaky User Name Flag a Password Flag určující, zda je součástí zprávy CONNECT přihlašovací jméno a heslo.[31]

**Postup (Způsoby autentizace [28, 31])**
MQTT umožňuje používat různé způsoby autentizace. Ten nejjednodušší je uvedení přihlašovacího jména a hesla ve zprávě CONNECT (přihlašovací jméno je něco jiného než identifikátor klienta – dokonce více klientů může používat totéž přihlašovací jméno), což ovšem není zrovna nejbezpečnější metoda, i pokud je přenos šifrován. V takovém případě si broker vede tzv. `password_file`, což je soubor ve speciálním formátu, podle kterého broker kontroluje přístupová oprávnění.

Ale existují i další možnosti v podobě autentizačních pluginů. Ve zprávě CONNECT je pole určené pro stanovení autentizační metody. Pokud je toto pole použito, pak po odeslání zprávy CONNECT (a před zprávou CONNACK) je provedena výměna autentizačních informací pomocí zpráv AUTH, přičemž podle zvolené metody lze autentizovat jen klienta nebo obě strany. Úspěšná autentizace je završena odesláním zprávy CONNACK brokerem klientovi. K popisu různých možností autentizace podporovaných MQTT brokerem Mosquitto se dostaneme na webu [28], v menu Documentation – All – Authentication methods.

**Postup (Zanechání zprávy u brokera pro nové odběratele [30])**
Dalším zajímavým mechanismem je možnost uschovávat u brokera vždy nejaktuálnější zprávu PUBLISH pro později se připojivší subscribery. Pokud publisher ve zprávě PUBLISH nastaví příznak RETAIN, je to informace pro brokera, aby tuto zprávu po rozeslání subscriberům nemazal, ale uložil (případně přepsal předchozí zprávu PUBLISH pro totéž téma). Pokud se následně připojí nový subscriber a projeví zájem o toto téma (nebo dříve připojený subscriber požádá o dané téma),

je mu uložená zpráva PUBLISH okamžitě přeposlána, není nutné čekat na to, až příslušný publisher pošle další příspěvek do daného tématu.

Pokud broker má uloženou (starší) zprávu, kterou obdržel s příznakem RETAIN, a dostane další zprávu do téhož tématu také s příznakem RETAIN, tu starší smaže, nahradí novější zprávou a samozřejmě tu novější rozešle subscriberům.

Naopak pokud publisher dříve poslal zprávu s nastaveným příznakem RETAIN (a tudíž ji broker uložil), ale další zpráva do tématu přijde bez příznaku RETAIN, ta původní uložená zůstane kde je, třebaže subscriberům bude okamžitě poslána ta nová. Nově ohlášeným subscriberům však bude poslána ta původní uložená s příznakem RETAIN.

Publisher může požadovat smazání takto uložené zprávy tím, že brokerovi pošle zprávu PUBLISH k danému tématu, ale s prázdným tělem zprávy (tj. bez dat patřících do tématu) s nastaveným příznakem RETAIN – broker reaguje odstraněním dříve uložené zprávy PUBLISH.

### 3.4.3 Implementace protokolu MQTT

Existuje celá řada implementací MQTT, ať už pro roli brokera nebo klienta. V rámci implementace jsou k dispozici MQTT knihovny a pak obslužný software – rozhraní. Z nejznámějších open-source implementací MQTT brokera i klienta můžeme jmenovat například nástroj *Mosquitto*[28] pro Windows i Linux vyvíjený v rámci projektu Eclipse.

Oblíbeným jednoduchým open-source MQTT klientem je *Paho MQTT Client*[8] z projektu Eclipse napsaný v jazyce Python a přístupný také příkazy v Pythonu (MQTT je totiž o komunikaci machine-to-machine, takže uživatelské rozhraní na klientovi je celkem zbytečné).

Další zajímavá implementace je *ActiveMQ*[9] vyvíjená v rámci projektu Apache, dále *RabbitMQ*[10], *emqttd*[11] a další.

[8] Web projektu Paho Client je https://www.eclipse.org/paho/index.php?page=clients/python/index.php, pěkný návod najdeme například na http://www.steves-internet-guide.com/into-mqtt-python-client/ [cit. 2022-08-23].

[9] Web projektu je https://activemq.apache.org/mqtt [cit. 2022-08-18].

[10] Web projektu je https://www.rabbitmq.com/ [cit. 2022-08-18].

[11] Web projektu je https://emqtt.io/docs/ [cit. 2022-08-18].

## 3.5 ZigBee

Protokol ZigBee (dále také ZB) jsme si už v této kapitole už představili. Zde se mu budeme věnovat podrobněji jako zástupci IoT protokolů nižších vrstev používaných ve WSN sítích.

Za tímto protokolem stojí ZigBee Alliance, která ve specifikaci staví na standardu IEEE 802.15.4. Přejímá tak funkcionalitu fyzické vrstvy a MAC podvrstvy, k tomu pak přidává vše, co je potřeba pro fungování kompletního zařízení pro WSN síť.

### 3.5.1 Zařízení v ZigBee síti

V síti ZigBee existují tři typy zařízení:[7, 42]

- ZigBee End Device (také Sensor Node) – koncové zařízení, které umí prostě přijímat a odesílat ZB zprávy, má tedy jen základní (redukovanou) implementaci ZigBee zásobníku, typicky se jedná o jednoduché senzory nebo aktuátory,
- ZigBee Router – oproti ZB koncovému zařízení umí také přeposílat ZB zprávy,
- ZigBee Coordinator – v síti musí být právě jeden, řídí celou komunikaci a obsahuje také funkcionalitu ZB Routeru.

Pokud například jsou přes ZigBee připojeny chytré žárovky, budou pravděpodobně spadat do první nebo druhé kategorie. Pokud budou patřit do té první (ZB koncové zařízení), pak dokážou pouze vysílat informaci o svém stavu nebo přijímat pokyny ke změně konfigurace, ale pokud budou patřit do druhé kategorie (ZB Router), mohou kromě toho i přeposílat zprávy, u nichž nejsou zdrojem ani cílem, a tím rozšiřovat dosah ZigBee sítě.

Zařízení v ZigBee síti nemusejí nutně mít anténu pod proudem neustále, mohou být definovány tzv. *Inactive Periods*, tedy zařízení naslouchá periodicky vždy v určitém úseku časového intervalu.[42] Tato vlastnost je pro IoT velmi důležitá, protože zejména zařízení napájená baterií potřebují nutně šetřit energii.

Zařízení v ZigBee síti mohou být vzájemně propojena v jedné z několika fyzických topologií[42], jejichž ukázky vidíme na obrázku 3.3:

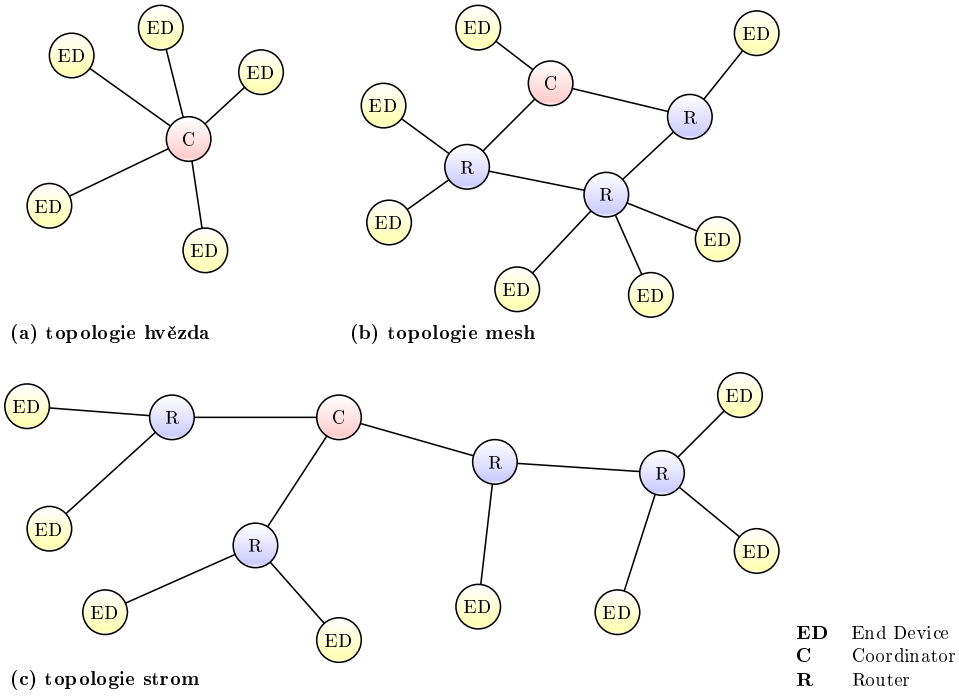
(a) hvězda (star) – v síti máme jeden uzel ZB Coordinator, který pracuje zároveň jako ZB router, přeposílá zprávy mezi jednotlivými koncovými zařízeními,

(b) mesh – v síti máme jeden ZB Coordinator, routery a koncová zařízení, propojení může být libovolné podle vzájemné viditelnosti jednotlivých uzlů,

(c) strom (cluster-tree) – v síti je taktéž jeden ZB Coordinator a libovolný počet dalších typů zařízení, ale na rozdíl od topologie mesh je struktura sítě přísně stromová.

První topologie má výhodu v jednoduchosti, nevýhodou pro použití v IoT je však to, že každé koncové zařízení musí být v dosahu středového uzlu, což může znamenat nutnost použít vyšší vysílací výkon. Tato topologie je tedy považována za nevýhodnou, pokud jsou koncová zařízení napájená baterií. Protože přeposílá pouze jediné zařízení, směrovací protokoly se v této topologii nepoužívají.

Druhá topologie už více odpovídá požadavkům běžné lokální IoT sítě. Koncová zařízení potřebují jen takový vysílací výkon, aby jejich signál dosáhl k nejbližšímu routeru. Navíc pro zprávy existují redundantní cesty, tedy při přerušení jedné cesty může být použita jiná. Zprávy jsou směrovány pomocí speciálního směrovacího protokolu pracujícího na linkové vrstvě (link layer routing). Nevýhodou této topologie je, že Inactive Period je dostupná pouze pro koncová zařízení – routery a Coordinator tuto vlastnost nemohou použít.[42]




Obrázek 3.3: Topologie v ZigBee síti (volně podle [42])

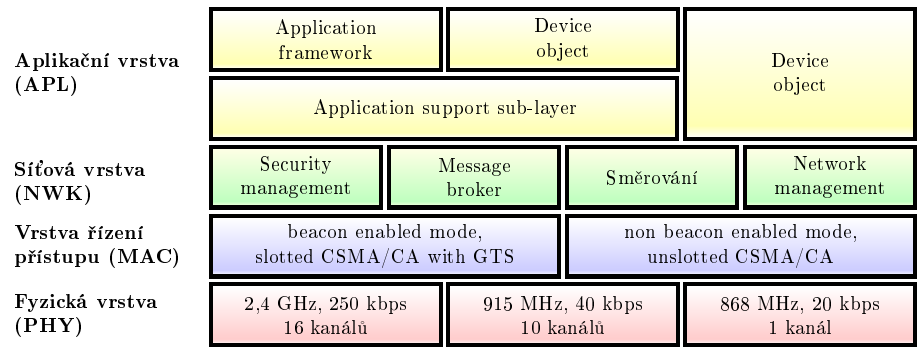
Třetí topologie neumožňuje používat redundantní cesty, ale na druhou stranu je cesta mezi každými dvěma zařízeními jednoznačně daná a je možno používat synchronizační mechanismus „beacon-enabled mode“ definovaný v IEEE 802.15.4 a díky tomu je vlastnost Inactive Period dostupná pro všechna zařízení v ZigBee síti.[42]

Je zřejmé, že pro střední a větší IoT sítě jsou vhodnější topologie mesh nebo strom, přičemž mezi těmito dvěma topologiemi volíme podle toho, zda upřednostňujeme redundantní cesty s volnější strukturou sítě nebo energetickou úspornost i na mezilehlých uzlech sítě.

### 3.5.2 Protokolový zásobník

Na obrázku 3.4 je naznačena struktura protokolového zásobníku na běžném ZigBee zařízení, které nepotřebuje přímý přístup do TCP/IP sítě. Na fyzické vrstvě se řeší signál, frekvenční pásma, kanály, modulace apod. Jak vidíme, jsou tři možnosti. První pásmo je shodné s tím, které najdeme i u základní Wi-fi: 2,4 GHz, proto lze použít i anténu dimenzovanou pro Wi-fi. Ovšem na rozdíl od Wi-fi je toto pásmo jinak využíváno co se týče počtu kanálů (a tedy jejich šířky) a také modulace a multiplexování. Další dvě pásma (v Evropě je z nich používáno pouze pásmo kolem 868 MHz) jsou na nižších frekvencích, což sice znamená nižší rychlost, ale na druhou stranu vysílání na těchto frekvencích je mnohem energeticky úspornější.[16]

Na vrstvě MAC vidíme další paralelu s Wi-fi, také se zde používá varianta přístupové metody CSMA-CA pro vyhýbání se kolizím v používaném frekvenčním pásmu. Na této podvrstvě se také řeší asociace a deasociace zařízení, struktura




Obrázek 3.4: Protokolový zásobník IEEE 802.15.4, ZigBee (podle [7, 16])

a adresace rámců, jejich potvrzování, beacon mechanismus a další provozní mechanismy, ale také zabezpečení (šifrování, ACL, integrita apod.).

Nad podvrstvou MAC může pracovat buď běžný protokol LLC jako v klasických sítích (IEEE 802.2), nebo lze k MAC vrstvě přistupovat přímo ze specifických protokolů jako je ZigBee či Z-Wave.[16, 53]

V horní části obrázku vidíme objekty zařízení (Device Objects). Device object zajišťuje management zařízení: určuje stav zařízení (včetně toho, kdy má být zařízení aktivní a kdy má být síťové rozhraní uspané), způsob adresace, ale také bezpečnostní funkce a další.

Spodní dvě vrstvy na obrázku 3.4 (PHY a MAC) jsou implementovány v IEEE 802.15.4, vrstvy NWK a APL jsou čistě záležitostí ZigBee.

**Profily.** Aby se usnadnilo a především zpřehlednilo určení konkrétních částí zásobníku, které je třeba implementovat na ZigBee zařízení, existují tzv. *profily zařízení*. Několik profilů je předdefinovaných ZigBee Alliancí, například Home Automation (HA), Commercial Building Automation (CBA), Personal Home and Hospital Care (PHHC), Industrial Plant Monitoring (IPM).[42] Zařízení má aktivní jeden konkrétní profil, který určuje, jaká je role daného zařízení.

Výše jmenované profily definované ZigBee Alliancí označujeme jako veřejné (public). Výrobce zařízení může definovat vlastní profily, které označujeme jako soukromé (private), a pokud se výrobce rozhodne takový profil publikovat, pak jde o publikovaný profil (published).[16]

Profil je na zařízení určen sadou objektů zařízení (Device object, na obrázku 3.4 nahoře) a jejich konfigurací. Každý profil nabízí určité služby, které jsou dány právě určením a stavem objektů zařízení.[42] Základními službami jsou objevování zařízení v síti (device discovery) a zjišťování služeb na zařízeních v síti (service discovery).

V aplikační vrstvě máme také aplikační rozhraní (Application Framework), což je rozhraní k aplikacím. Zde jsou definovány *aplikační profily*, jejichž role (směrem nahoru k aplikacím) je podobná jako role profilu zařízení (směrem dolů k síti): máme zde aplikační objekty, které určují, jakou aplikační roli zařízení plní, například že jde o chytrou žárovku nebo pohybový senzor.[53] Každá role vyžaduje specifické zacházení a nabízí specifické služby, například s žárovkou budeme zacházet trochu jinak než s pohybovým senzorem. Zatímco žárovce budeme posílat například zprávy s požadavkem na zapnutí či změnu barvy, pohybový senzor by si s takovými zprávami neporadil.

### 3.5.3 Rámce a průběh komunikace v síti

Protože standard IEEE 802.15.4 definuje spodní vrstvy síťového zásobníku včetně linkové vrstvy, plní podobnou úlohu jako v běžných lokálních sítích protokoly Ethernet (přesněji Ethernet II a IEEE 802.3), IEEE 802.11 (Wi-fi) nebo IEEE 802.15.1 (Bluetooth). K této úloze patří také práce s PDU linkové vrstvy, což jsou rámce. Formát rámce je odlišný od rámců jiných protokolů, zohledňují se specifika M2M komunikace a potřeby mít co nejúspornější záhlaví.

Každé zařízení potřebuje jednoznačnou adresu. Zatímco rámce Ethernet II, IEEE 802.11 a IEEE 802.15.1 používají výhradně klasické 48bitové MAC adresy, IEEE 802.15.4 umožňuje používat dva typy adres – krátké 16bitové nebo dlouhé 64bitové (EUI-64). Dlouhou adresu mají všechna zařízení podporující uvedený standard, generuje se postupem podobným tomu, který se používá pro autokonfiguraci IPv6 adresy. Krátkou adresu (která je jednoznačná pouze v dané WSN síti) obdrží zařízení při připojení do sítě od centrálního prvku[7] a lze z ní pro účely kompatibility vygenerovat pseudo-48bitovou MAC adresu[16].

Protokol IEEE 802.15.4 rozlišuje několik druhů rámců:[7]

- beacon frame s podobnou rolí jako jeho obdoba pro Wi-fi, tedy centrální prvek ohlašuje existenci sítě a sděluje zařízením informace potřebné k připojení,
- datový rámec,
- potvrzovací rámec (ack frame) potvrzující úspěšný přenos jiných rámců,
- příkazový rámec pro řízení sítě na úrovni MAC podvrstvy včetně řízení spolupráce uzlů sítě na přenosu.

Nejmenší možná délka rámce na MAC podvrstvě je 5 B, což právě délka potvrzovacího rámce (záhlaví je velmi krátké, bez adres, a tělo tento rámec nemá žádné). Maximální délka datového MAC rámce je 127 B, z toho 25 B je obvyklá délka záhlaví a zápatí, ovšem část si ještě „ukrojí“ nadřízená vrstva za účelem šifrování, zajištění integrity a dalších bezpečnostních funkcí. Zbývá jen velmi málo prostoru pro data, zejména pokud mají být posílána do TCP/IP sítě: tam si záhlaví vnořených PDU zaberou vcelku hodně prostoru – především protokoly TCP a IPv6.

Při přechodu mezi vrstvami definovanými ve specifikaci ZigBee a vrstvami IEEE 802.15.4 je naštěstí možné použít fragmentaci. Tedy pokud je potřeba poslat delší paket, bude fragmentován do několika rámců. Malá energeticky úsporná zařízení v ZigBee síti však nemají výpočetní kapacity na zpracování příliš velkých paketů, tedy je vhodnější si vystačit s malými objemy posílaných dat, kde jen to je možné.[16]

V protokolu IEEE 802.15.4 je pro MAC podvrstvu určena přístupová metoda CSMA/CA. Algoritmus této metody spolupracuje s fyzickou vrstvou, na níž najdeme mechanismus CCA (Clear Channel Assessment), který buď neustále, nebo při určité hladině napájení, sleduje vysílací pásmo a hlídá, zda je „čistý vzduch“.[42]

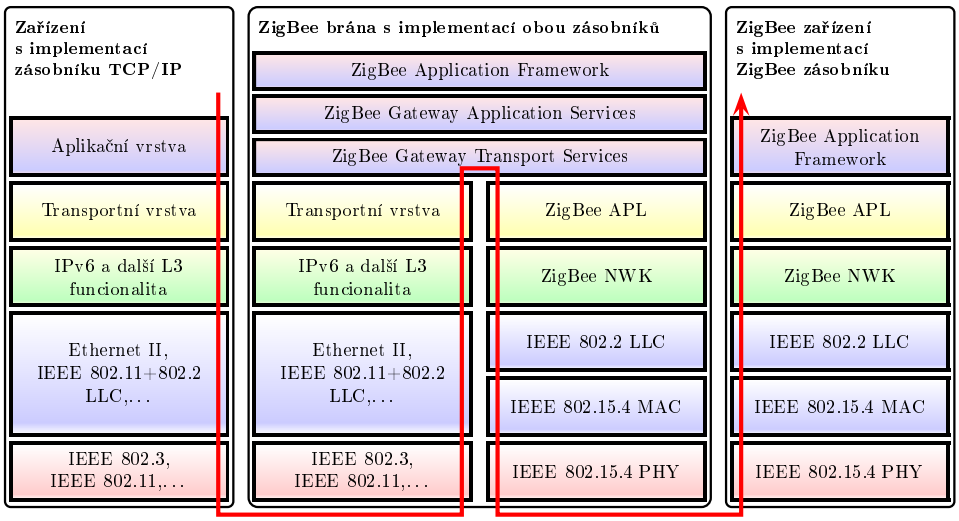
### 3.5.4 Napojení na okolní svět

Zařízení v ZigBee síti může potřebovat spojení se zařízením mimo tuto síť. Lze vydělit dvě základní situace:

1. ZigBee zařízení komunikuje s běžným TCP/IP zařízením, například senzor odesílá data do zobrazovacího zařízení,
2. ZigBee zařízení komunikuje se ZigBee zařízením nacházejícím se v jiné síti.

Pro první případ potřebujeme ZigBee bránu, která „přeloží“ ZB komunikaci do podoby, která nejen může projít přes síť TCP/IP, ale také bude moci být přijata koncovým TCP/IP zařízením, pro druhý případ potřebujeme ZigBee Bridge – most, který je jednodušší, protože řeší jen zapouzdření (vpodstatě tunel) do jiné ZigBee sítě přes IP síť.[16]

Na obrázku 3.5 je naznačena vertikální komunikace mezi TCP/IP a ZigBee zařízením přes ZigBee bránu. Na obrázku jsou ve skutečnosti zobrazena tři zařízení, uprostřed je ZigBee brána s implementací zásobníků ZigBee i TCP/IP.




Obrázek 3.5: Protokoly v ZigBee bráně (volně podle [16])

## 3.6 HTTP REST pro IoT

### 3.6.1 HTTP zprávy v mechanismu REST

V sekci 2.4 od strany 28 je popsán mechanismus REST. Jak víme, je to ve skutečnosti koncept, pro jehož implementaci potřebujeme konkrétní aplikační protokol s vhodnými typy zpráv pro CRUD komunikaci. Z těch protokolů, které jsme zatím procházeli, se pro tento účel dá využít buď HTTP nebo CoAP, případně HTTPS.

**Příklad**

V mechanismu REST, zde s využitím HTTP zpráv, posíláme krátké textové zprávy. Ukážeme si, jak vypadají tyto zprávy v případě, že v domácí či firemní IoT síti je nejdřív registrována meteostanice a následně začne v síti fungovat.

Předpokládejme, že pro IoT máme určenou podsíť 10.4.0.0/24. V této síti je server s adresou 10.4.0.2 a bude přijímat požadavky na portu 8080. V komunikaci bude tato adresa vždy na straně serveru. Nejdřív se meteostanice coby klientské zařízení registruje u serveru. Pošle zprávu POST s daty ve formátu JSON:

**Klient** (meteostanice) **Server** (10.4.0.2)

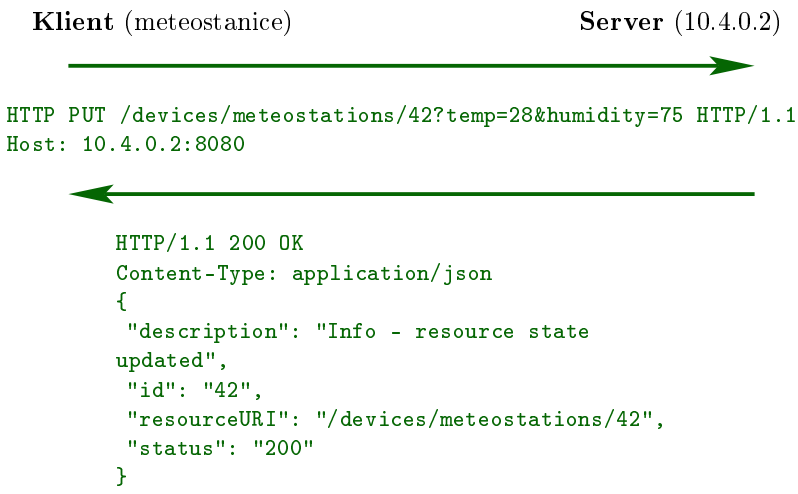


```
HTTP POST /devices/meteostations HTTP/1.1
Host: 10.4.0.2:8080
Content-Type: application/json
{
 "type": "meteostation",
 "name": "severni"
}
```

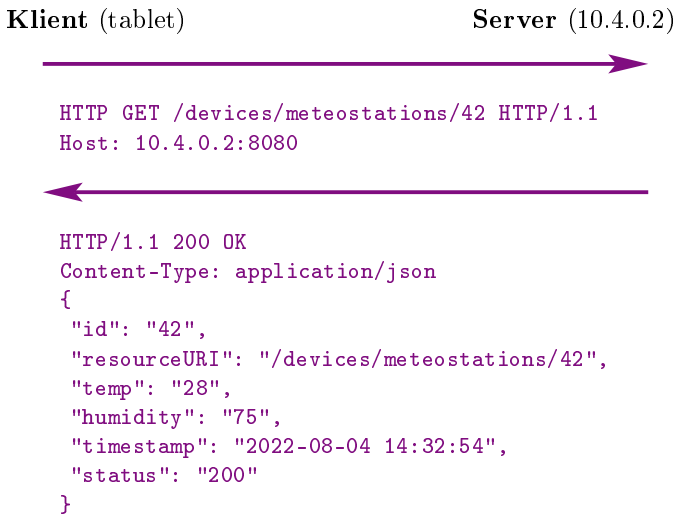


```
HTTP/1.1 201 Created
Content-Type: application/json
{
 "description": "Info - resource registered",
 "id": "42",
 "type": "meteostation",
 "name": "severni",
 "resourceURI": "/devices/meteostations/42",
 "status": "201"
}
```

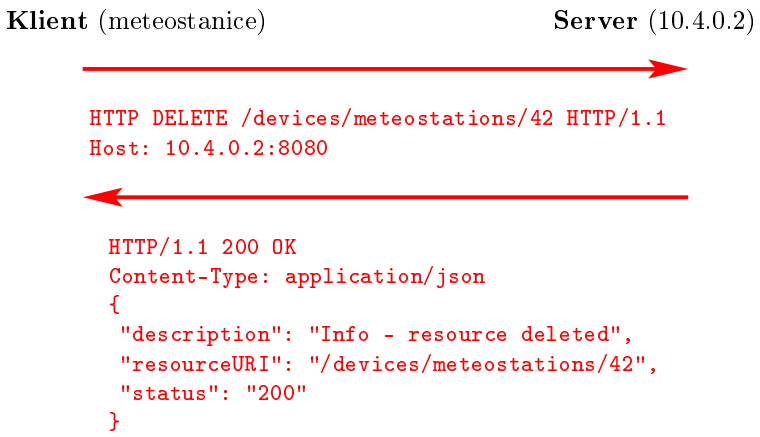
Teď je meteostanice registrována a nadále bude v pravidelných intervalech posílat data na server:




Na serveru běží software, který si do databáze ukládá výsledky získané z různých zařízení včetně naší meteostanice, může jít o webovou aplikaci. Jiné zařízení se podobným způsobem může „zeptat“ na údaje posílané meteostanicí:



Meteostanice bude odpojena a server už s tímto zdrojem nebude nadále pracovat:




Názvy parametrů jednotlivých zařízení i strukturu zdrojů určuje programátor. Obdobné příklady najdeme na webu, jednoduchý příklad je i v článku [10]. Popis HTTP metod použitelných v RESTful aplikacích je na [12], včetně stavových kódů v odpovědích.

V příkladu je meteostanice spíše zdrojem informací, ale nemáme zajištěno ovládání zařízení. V úvahu přichází například možnost zapnutí (probuzení) či vypnutí (uspání) zařízení, u chytré žárovky také například změna teploty či intenzity světla. V těchto případech bude IoT zařízení příjemcem zprávy GET s příslušnými parametry v URI (stejně jak jsme ukázali u metody PUT v příkladu) nebo POST s parametry v těle zprávy například ve formátu JSON. Druhá možnost je čistší, protože důsledkem přijetí zprávy podle metody GET by neměla být změna stavu příjemce.[12] Také by bylo vhodné řešit autentizaci a šifrování.

### 3.6.2 Jak poslat HTTP zprávu

HTTP zprávu posílá aplikace (tj. programátor prostřednictvím funkce z příslušné knihovny pro síťovou komunikaci), ale existují způsoby, jak něco takového může provést i uživatel – příkazem nebo pomocí volby například modulu ve webovém prohlížeči.

**Postup (HTTP komunikace pomocí příkazu)**
O odeslání HTTP zprávy se může postarat kód aplikace (což ovšem někdo musí naprogramovat), nebo lze použít program `curl`. Jde o program původně určený pro UNIXové systémy včetně různých distribucí Linuxu, MacOS, atd., který umí posílat zprávy celé řady (textových) aplikačních protokolů: kromě HTTP například FTP, TFTP, TELNET, LDAP a dalších.

Podrobnosti o programu `curl` zjistíme v manuálové stránce, kterou zobrazíme buď standardním způsobem v systému, kde je program nainstalován, nebo na webu třeba na adrese https://linux.die.net/man/1/curl. Program `curl` se dá doinstalovat i do Windows.[12]

**Příklad**
Pokud máme nainstalován program `curl`, zprávu HTTP POST pošleme například takto (vše bychom napsali na jeden řádek):

```
curl -X POST http://10.4.0.2:8080/devices/meteostations
     -H 'Content-Type: application/json'
     -d '"type": "meteostation", "name": "severni"'
```

První parametr (`-X`) určuje metodu a adresu, další (`-H`) údaj do záhlaví, poslední (`-d`) jsou data ve zprávě. Rozlišují se malá a velká písmena.

**Postup (HTTP komunikace pomocí doplňku webového prohlížeče)**
Další „rychlou“ možností, tentokrát více klikací, je doplněk do webového prohlížeče. Pro Firefox existuje doplněk RESTclient,[13] nebo RESTer[14].

## 3.7 IoT ve vlastní režii

IoT zařízení se dají sehnat už vyrobená, naprogramovaná a se základní konfigurací. Bohatě zastoupené sady různých IoT zařízení se prodávají například pod značkami

[12] `curl` pro Windows je ke stažení na https://curl.se/windows/ [cit. 2022-08-04].
[13] https://addons.mozilla.org/cs/firefox/addon/restclient/ [cit. 2022-08-04]
[14] https://addons.mozilla.org/cs/firefox/addon/rester/ [cit. 2022-08-04]

Jablotron[15], Fibaro[16], Netatmo[17], poněkud méně široký výběr mají i další výrobci.

Jestliže však chceme něco specifického či z jiného důvodu nám výše uvedené sady nevyhovují, jsou i jiné možnosti. V tom případě se musíme postarat o hardware, software na IoT zařízeních a také jejich propojení do sítě.

**Hardware.** Co se týče hardwaru, je z čeho vybírat. Máme k dispozici například komponenty pro platformu Arduino[18] nebo počítač Raspberry Pi[19]. Následující informace najdeme na webech projektů.

Arduino je skládačka s otevřeným hardwarem, jednotlivé komponenty mohou tedy vyrábět různí výrobci (což má příznivý vliv na cenu a dostupnost). Uživatel si koupí některou základní desku, například Arduino UNO, a pak si vybírá takové další komponenty či moduly, které se mu do projektu hodí. Raspberry Pi je na rozdíl od Arduina rozšiřitelný minipočítač, a to s uzavřeným hardwarem. Lze si vybrat mezi několika modely s různou výbavou a spotřebou. Rozšiřitelnost je také možná. Moduly pro základní konektivitu (USB, Wi-fi, Bluetooth, LTE, LoRaWAN, atd.) jsou buď součástí nebo je lze snadno přidat. Také výše probírané ZigBee a MQTT lze zprovoznit na obou platformách, popřípadě využít REST.

Výběr mezi nimi je otázkou preferencí každého uživatele, obě platformy jsou pro IoT vhodné. Obecně můžeme říci, že Arduino se volí spíše tam, kde potřebujeme malé jednoduché zařízení provádějící jednu konkrétní činnost nebo například sbírající data ze senzorů a posílající je aktuátorům, kdežto Raspberry Pi je vhodnější tehdy, když zařízení provádí něco komplexnějšího, výpočetně náročnějšího, více úloh najednou, zpracování multimédií (třeba natočeného videa).

**Další informace**

Uvedené platformy jsou nejznámější, ale nejsou jediné. Zajímavý přehled najdeme například na https://blog.particle.io/iot-hardware-comparison-guide/ [cit. 2022-09-06] nebo ve zdroji [2], případně stačí projít internetové obchody.

**Software.** Vždy záleží, co od zařízení chceme a pro jakou platformu jsme se rozhodli. Buď máme na zařízení operační systém (což je typické například pro Rasp-

[15] https://www.jablotron.com/cz/ [cit. 2022-09-02]
[16] https://www.fibaro.com/cz/ [cit. 2022-09-04]
[17] https://www.netatmo.com/cs-cz [cit. 2022-09-04]
[18] https://www.arduino.cc/ [cit. 2022-09-06]
[19] https://www.raspberrypi.org/ [cit. 2022-09-06]

berry Pi), nebo si vystačíme s jednoduchým obslužným kódem na úrovni firmwaru (což je typické například pro Arduino).

Operační systém pro IoT by měl splňovat určité požadavky – měl by být spíše odlehčený, aby nezatěžoval systém zbytečnými výpočty, a měl by reagovat s co nejmenší latencí. Používá se také pojem *embedded operating system*, protože na takovém zařízení by operační systém „neměl být vidět".

Pro tyto účely se obvykle používají realtimové systémy. U IoT se setkáme například s Contiki, TinyOS, FreeRTOS, Embedded Linux, použitelný je také Android.

**Další informace**

Přehled operačních systémů pro IoT včetně odkazů na weby projektů najdeme například na https://ubidots.com/blog/iot-operating-systems/ [cit. 2022-09-06] nebo ve zdroji [2].

Pro jednoduché microcontrollery včetně Arduina není nutno vymýšlet operační systém, obvykle stačí naprogramovat požadovanou funkcionalitu a použít kód jako firmware. Na Internetu najdeme spoustu návodů, programovat IoT se dnes učí i děti na základních školách. Pro tyto účely se obvykle používá programovací jazyk C, C++, Python, někdy Java či jiný programovací jazyk, záleží na konkrétní platformě a zvoleném vývojovém prostředí.

**Další informace**

Jak je výše zmíněno, hodně návodů a ukázek kódu najdeme na Internetu. Série zajímavých článků o programování elektroniky (zejména IoT) také vychází v časopisu Computer vydatelství Živě.cz, na webu vydavatelství také najdeme související články (stačí do vyhledávacího pole zadat „programování elektroniky"):
https://www.zive.cz/vysledky-vyhledavani/sc-236/?q=programov%C3%A1n%C3%AD%20elektroniky [cit. 2022-09-12].

Kapitola 4

# Počítačové sítě a membránové systémy

V této kapitole se budeme zabývat tématem, které propojuje teoretickou informatiku a počítačové sítě. Membránové systémy můžeme považovat za prostředek pro modelování jednoduchých paralelních procesů typu přenosu objektu mezi různými lokacemi (membránami), případně pokud membrány představují hranice mezi lokacemi, tak přenosu přes membránu, a v počítačových sítích jde přesně o totéž – přenos dat mezi různými uzly sítě, resp. přes určitý typ rozhraní.

Dále budeme předpokládat, že čtenář ovládá základy teorie formálních jazyků a membránových výpočtů. Pokud ne, doporučujeme například zdroje [15] a [34].

## 4.1 Membránové systémy

Historie membránových systémů začíná roku 1998, kdy byly představeny Gheorghem Păunem jako paradigma pro paralelní distribuované výpočty. Podrobné informace najdeme v [32, 33, 34], nebo na webu [43].

Matematické modely membránových systémů nazýváme P Systémy. Základem P Systému je hierarchická struktura membrán podobná membránové struktuře biologické buňky. V membránách se kromě případných vnořených membrán nacházejí objekty a každá membrána má definovaná pravidla pro zacházení s těmito objekty.

Celý systém pracuje paralelně, membrána se svými objekty manipuluje nezávisle na ostatních, asynchronně. Toto opět odpovídá procesům v biologické buňce.

### 4.1.1 Předchozí práce

Použití membrán ve světě IoT není nic nového. Například v článku [49] autoři představují koncept nazvaný „osmotic computing“ jako paradigma sloužící k optimalizaci přístupu ke zdrojům a službám v síti, včetně cloudových služeb. Setkáváme se zde s tzv. mikroslužbami, které sice mohou být poskytovány v cloudu, ale také se mohou z cloudu přesunout na hranici sítě (tedy se využívá edge computing). Na hranici sítě máme svá vlastní zařízení, pod naší kontrolou, tedy přesun mikroslužeb do této oblasti má mnohé výhody. Motivem jsou opět procesy z přírodních věd – osmóza, kde molekuly rozpouštědla mají procházet přes polopropustnou membránu do oblasti s vyšší koncentrací rozpouštěné látky.

Dále je tento koncept rozvíjen v článku [38], kde autoři popisují, jak mikroslužby migrují mezi cloudem a místní sítí a více se zaměřují na svět Internetu věcí.

Jak bylo výše uvedeno, membrány se zde mají použít pro modelování procesů. Autoři výše zmíněných článků používají pojem MELs (MicroElements) pro mikroslužby a zdroje procházející membránami do různých lokací, MELs přecházejí mezi různými zařízeními pomocí tzv. softwarově definovaných membrán (Software-Defined Membranes, SDM). Tutéž terminologii používají i autoři článku [50], kde je princip SDM velmi dobře popsán a je na něm postavena dynamická správa zdrojů a služeb v IoT včetně zahrnutí bezpečnosti komunikace v IoT síti. V článku [9] jsou MELs využívány při zapojení různých zařízení do IoT sítě, především chytrých automobilů a podobných zařízení. V článku [5] autoři používají MELs ve zdravotnické aplikaci.

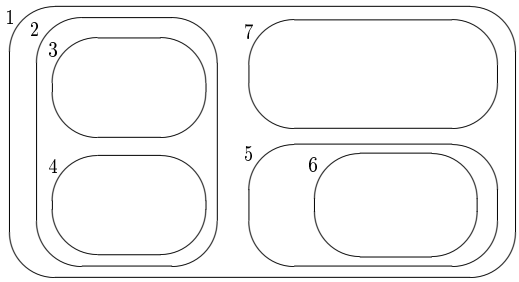
Účelem výše zmíněného SDM je tvořit speciální rozhraní přidané k zařízení, nejde přímo o komunikaci mezi IoT zařízeními. Zde budeme P Systém používat přímo pro reprezentaci komunikace mezi IoT zařízeními, třebaže si lze náš systém také představit jako komunikační rozhraní přidané k zařízením. Membrány neslouží k transformaci dat (třebaže P Systém by to zvládl), ale pouze k jejich přenosu. Objekty v membránách odpovídají zprávám protokolů typu MQTT, CoAP apod., jsou tedy jakýmisi kontejnery pro data. Postup byl poprvé představen v [45], rozveden v [48] a následně dopracován v [44].

Vztah mezi teorií (membránové systémy) a počítačovými sítěmi může být i v opačném směru: v článku [47] je ukázán postup optimalizace membránové struk-

tury s využitím algoritmu používaného protokolem STP pro udržování sítě switchů ve formě stromu bez smyček.

### 4.1.2 Definice membránového systému

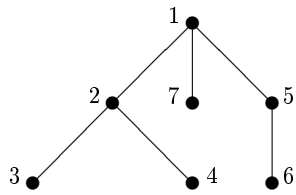
V P Systému máme strukturu navzájem vnořených membrán, z nichž jedna je hlavní. Na obrázku 4.1 je ukázka grafické reprezentace struktury (Vennův diagram) sedmi membrán, membrána 1 je hlavní.




Obrázek 4.1: Příklad membránové struktury (zdroj: [48])

Membránovou strukturu můžeme reprezentovat i jinak.[6] Kromě Vennova diagramu se používá také reprezentace pomocí stromu, kde je vztah nadřízená/vnořená membrána převeden na vztah rodič/potomek – vidíme na obrázku 4.2, nebo pomocí posloupnosti vnořených závorek opatřených návěštími – posloupnost odpovídající našemu příkladu by vypadala následovně:
$[\ [\ [\ ]_3\ [\ ]_4\ ]_2\ [\ [\ ]_5\ ]_6\ [\ ]_7\ ]_1$.

Ať už membránovou strukturu zapisujeme jakkoliv, každá membrána musí být opatřena návěštím. V našem příkladu jsou jako návěští použita čísla, mohou to však být i řetězce nebo jakýkoliv jiný typ jednoznačného identifikátoru.




Obrázek 4.2: Ukázka reprezentace membránové struktury pomocí stromu (zdroj: [48])

Membránovou strukturu lze snadno naprogramovat – reprezentovat v programovém kódu, přičemž způsob reprezentace bychom volili podle konkrétních potřeb a způsobu nakládání s výslednou strukturou. Může jít například o dynamickou datovou strukturu (dynamický strom), pokud nám v daném případě nevadí používat dynamické struktury (to může být problém, pokud budeme potřebovat části struktury zamykat pro vyhrazený přístup). Pokud má jít o statickou strukturu, tedy nechceme přidávat ani ubírat membrány, lze strukturu membrán reprezentovat přímo ve struktuře kódu, což je technika, kterou používáme například při programování překladačů.

**Definice (P Systém [6, 32, 48])**
*Nechť $H$ je množina návěští. Potom P Systém stupně $m$, $m \geq 1$, je*

$$\Pi = (V, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$$

*kde:*

*(i)* *$V$ je neprázdná abeceda, její prvky nazýváme objekty,*

*(ii)* *$\mu$ je membránová struktura skládající se z $m$ membrán, membrány jsou označeny návěštími z $H$,*

*(iii)* *$w_i$, $1 \leq i \leq m$, jsou řetězce reprezentující multimnožiny nad abecedou $V$ (tj. objekty) nacházející se v regionu $i$-té membrány v $\mu$,*

*(iv)* *$R_i$, $1 \leq i \leq m$, jsou konečné množiny evolučních pravidel příslušejících k $i$-té membráně v $\mu$; evoluční pravidlo je uspořádaná dvojice $(u, v)$, také lze zapsat jako $u \to v$, kde*

- *$u$ je řetězec objektů nad $V$,*
- *$v = v'$ nebo $v = v'\delta$, kde $v'$ je řetězec nad množinou $\{a_{here}, a_{out}, a_{in_j} \mid a \in V,\ 1 \leq j \leq m\}$, dále $\delta$ je speciální symbol $\notin V$ značící operaci rozpuštění membrány.*

*Objekty mohou být přenášeny přes hranice membrán pomocí evolučních pravidel podle určení cíle* out *(do rodičovské/nadřízené membrány) nebo* in *(do podřízené membrány určené indexem), nebo mohou zůstat v původní membráně při určení cíle* here.

Podrobnosti a příklady můžeme najít například ve zdrojích [32] a [6].

## 4.2 Modelování komunikace v IoT

Zde použijeme P Systém pro vytvoření zjednodušeného obecného modelu komunikace podobného tomu, co známe z MQTT. Tento model lze využít například pro vizualizaci této komunikace (ostatně P Systémy lze přímo graficky reprezentovat) nebo pro návrh a naprogramování vlastního protokolu podle individuálních požadavků. Uvedený postup je v o něco stručnější podobě publikován v článku [44].

### 4.2.1 Vrstvený model

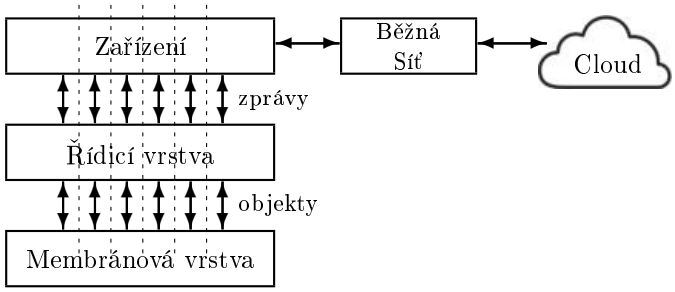
Vytvořit celý model komunikace pouze s využitím P Systému by bylo problematické. Potřebujeme totiž kromě samotného přenosu dat také generovat nové objekty podle toho, co konkrétně zařízení potřebují přenášet, což znamená k datům poskytnutým zařízeními přidávat metadata a vytvořit příslušný objekt. Dále mazat objekty, které už nejsou potřeba, protože jimi přenesená data již byla přijata cílovým zařízením, případně evidovat informace potřebné k práci s objekty. Potřebujeme, aby náš systém byl dostatečně pružný, dynamický, aby dokázal reagovat na případné změny v síti, tedy zřejmě budeme potřebovat i možnost zasahovat do množin pravidel jednotlivých membrán. Máme tyto možnosti:

1. Změníme definici P Systému: potřebujeme možnost vytváření a rušení objektů obsahujících data a metadata, ale také vytváření a rušení pravidel v membránách.
2. Přidáme pomocnou vrstvu mezi zařízení a P Systém, tato vrstva bude plnit výše naznačené úkoly související s objekty.

První možnost by sice byla použitelná, dokonce existují modifikované definice P Systémů bližší našim potřebám, ale není to ideální řešení. Celý návrh by byl zbytečně komplikovaný.

Druhá možnost více vyhovuje našim potřebám. Nemusíme zasahovat do definice P Systému ani do funkčnosti samotných IoT zařízení. Vytvořená řídicí vrstva sice bude ovlivňovat P Systém – modifikovat pravidla, ale důsledek se bude projevovat pouze na činnosti našeho systému, nebude způsobovat kolize ani systémové chyby. Celou strukturu vidíme na obrázku 4.3.

Řídicí vrstva se tedy nachází nad vrstvou obsahující membránovou strukturu a pod vrstvou se samotnými zařízeními, tedy opravdu bude plnit roli komunikačního rozhraní. Pokud bude potřeba, vrstva může generovat také provoz mířící na Internet, stačí k zařízením zařadit i běžný router a posílat mu HTTP(S) zprávy.




Obrázek 4.3: Komunikační architektura se třemi vrstvami (podle [44])

Podobně i pro opačný směr. Membránová vrstva je z našeho pohledu modelem spodních vrstev síťového zásobníku, tedy zajišťuje samotný přenos dat.

Abychom v tomto textu zamezili nedorozumění, budeme entity řídicí vrstvy odpovídající jednotlivým IoT zařízením nazývat komponentami. Ke každému „fyzickému" IoT zařízení tedy máme v řídicí vrstvě právě jednu komponentu. Podobně tomu bude směrem dolů: každé komponentě řídicí vrstvy odpovídá právě jedna membrána P Systému v membránové vrstvě. Na obrázku 4.3 je tento vztah mezi vrstvami znázorněn svislými tečkovanými linkami.

Jednotlivá IoT zařízení vzájemně komunikují pouze zprostředkovaně přes membrány (a tedy vertikálně přes celý zásobník, což je naznačeno tím, že tečkované linky nepřetínají celou membránovou vrstvu). Oddělení datových přenosů do membránové vrstvy má v tomto modelu jeden důležitý benefit: není nutné, aby všechna zařízení měla implementován stejný komunikační protokol – stačí, aby dokázala komunikovat s řídicí vrstvou. Dokonce ani není nutné, aby všechna zařízení dokázala k datům dodávat všechna správná metadata – řídicí vrstva opět může vyrovnávat rozdíly a v rámci svých možností do objektů dodávat tato metadata sama.

Systém navrhujeme tak, aby nebylo nutné zasahovat do definice P Systému, ale určitá změna je nutná: klasický P Systém nedokáže pracovat se sémantikou, obsahuje pouze jednoduché objekty bez další informace (často jsou reprezentovány písmeny abecedy). Objektům musíme dodat sémantickou informaci, což znamená především identifikátory zdrojové a cílové membrány, přenášená data (například hodnoty teploty a vlhkosti zjištěné meteostanicí), případně přístupové údaje (například jméno a heslo) v objektu použitém pro autentizaci.

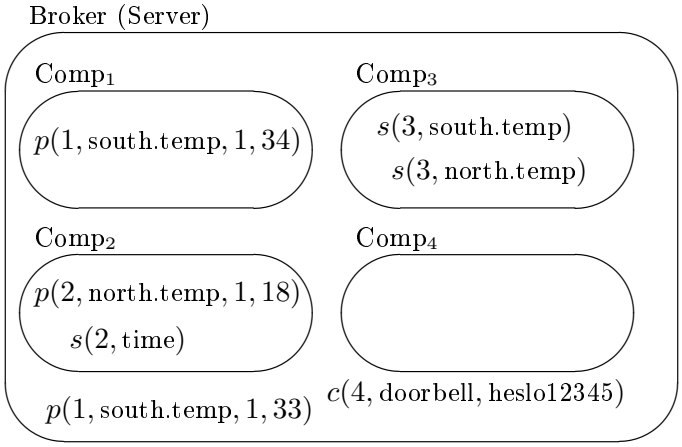
Objekty se sémantickou informací bude vytvářet řídicí vrstva, pravidla v membránách je pouze přenesou do jiné membrány. Nicméně v pravidlech musíme se

sémantikou počítat, protože nelze mít například různá pravidla zvlášť pro různé hodnoty teplot generované teploměrem. Objekt uvedený v pravidle tedy bude mít obecné parametry (proměnné), za něž budeme při použití pravidla dosazovat konkrétní hodnoty. Abychom učinili zadost definici, pravidla nebudou do parametrů (sémantiky) nijak zasahovat, pouze provedou transfer mezi membránami.

Aby řídicí vrstva mohla plnit svou funkci (včetně úpravy pravidel membrán v membránové vrstvě), musí mít uložen momentální stav P Systému (odkazy na membrány a pravidla) a další informace. Jak bylo výše uvedeno, v principu se budeme držet toho, co platí v protokolu MQTT, tedy potřebujeme evidenci témat, objednávek a samozřejmě připojených zařízení. Řídicí vrstva bude na základě těchto informací řídit činnost P Systému v membránové vrstvě: pokud zařízení publikuje data, řídicí vrstva vytvoří nový objekt s těmito daty a doprovodnými metadaty (označení publikujícího zařízení apod.) a předá tento objekt membráně odpovídající danému zařízení. Naopak pokud do určité membrány je v rámci P Systému doručen objekt, řídicí vrstva ho vyzvedne a předá přenášená data odpovídajícímu zařízení. Vztah mezi našimi vrstvami je podobný jako vztah mezi vrstvami v ISO/OSI.

### 4.2.2 Membránová vrstva

Na obrázku 4.4 je příklad membránové struktury IoT systému, ve které máme membrány pro jednoho brokera a několik komponent. Hlavní membrána je přidružena IoT centrálnímu IoT zařízení, které zde budeme nazývat broker.




Obrázek 4.4: Příklad membránové struktury pro IoT síť (podle [44])

Komponenta $\text{Comp}_1$ představuje teploměr (nebo zařízení obsahující kromě jiného i teploměr). Je producentem v tématu `south/temp` – může se jednat o teploměr na jižní straně domu, hlásí teplotu 34 stupňů. V membráně vidíme objekt $p$ (publish), jehož první parametr určuje zdrojovou komponentu, druhý je téma pro publikování. Objekt $p$ bude následně zpracován pravidlem pro objekty $p$ platným v membráně této komponenty, konkrétně bude expedován do hlavní membrány.

Komponenta $\text{Comp}_2$ odpovídá také teploměru, produkuje v tématu `north/temp` a hlásí teplotu 18 stupňů na severní straně domu. Kromě toho odpovídající zařízení posílá objednávku tématu `time` – nachází se zde objekt $s$, komponenta tedy bude kromě publishera plnit také roli subscribera. Jedná se zřejmě o trochu sofistikovanější zařízení, které má displej a zobrazuje čas. Objekty $p$ a $s$ budou zpracovány příslušnými pravidly platnými v membráně $\text{Comp}_2$.

Komponenty $\text{Comp}_1$ a $\text{Comp}_2$ publikují data v pravidelných intervalech, v těchto intervalech se v příslušné membráně objevuje objekt $p$ s patřičnými parametry a následně je pravidly přenesen pryč z publikující membrány. Délka intervalu je stanovena v zařízení, řídicí vrstva do něj nezasahuje. Objekt $p$ pocházející z komponenty $\text{Comp}_1$ z předchozího intervalu se právě nachází v hlavní membráně a čeká, až si ho broker vyzvedne. Druhá komponenta má zjevně trochu delší interval publikování teploty, protože objekt $p$ z předchozího intervalu se už v hlavní membráně nenachází (nebo právě začala publikovat a uvedený objekt $p$ je první vytvořený).

Komponenta $\text{Comp}_3$ zřejmě přísluší zobrazovacímu zařízení. Zařízení bylo aktivováno a právě si pomocí objektů $s$ objednává data z témat, do kterých přispívají oba teploměry. Prvním parametrem obou objektů je identifikátor zařízení, následuje název objednávaného tématu.

Komponenta $\text{Comp}_4$ právě prochází aktivací. V předchozím kroku vygenerovala objekt $c$ (Connect), který byl přenesen do hlavní membrány a čeká na vyzvednutí brokerem (objekt vidíme na obrázku dole). V parametrech objektu jsou kromě identifikátoru odesílatele také přihlašovací údaje – jméno a heslo. Zřejmě půjde o zámek u dveří. Heslo je samozřejmě pouze demonstrační, obvykle volíme spíše silnější hesla...

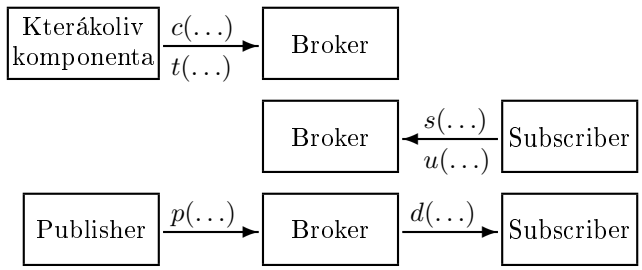
Na obrázku 4.4 jsme v různých membránách viděli celou řadu objektů s parametry. Ve vztahu k zařízením z vyšší vrstvy se jedná o zprávy posílané mezi zařízeními/komponentami. V našem systému používáme tyto druhy objektů:

- $c$(ID, credentials) (Connect) je objekt pro zprávu žádosti o připojení k brokerovi. Tuto zprávu posílá zařízení, které se má připojit do naší IoT sítě

a plnit roli publishera a/nebo subscribera. První parametr je identifikátor zařízení, následují přihlašovací údaje (jméno a heslo, přístupový klíč apod., podle nastavení v síti).

- $t$(ID) (Terminate) je opakem předchozího, odpovídá zprávě s žádostí o ukončení připojení. Tuto zprávu posílá zařízení, které se chce odpojit, ukončit spojení.
- $s$(ID, topic) (Subscribe) je objednávka. Zařízení posílá svou objednávku brokerovi, který si poznamená objednané téma a dále tomuto zařízení posílá vše, co je v v tématu publikováno.
- $u$(ID, topic) (Unsubscribe) se posílá tehdy, když zařízení chce zrušit objednávku daného tématu.
- $p$(ID, topic, retain, data) (Publish) odpovídá zprávě posílané publisherem, a to pro první fázi publikační cesty (od publishera k brokerovi). První parametr určuje odesílatele, druhý téma, do kterého se publikuje. Čtvrtý parametr jsou samotná data. Třetí parametr „retain“ nabývá hodnoty 0 nebo 1: hodnota 1 znamená, že broker si pro dané téma uloží posílaná data do databáze a v případě, že se některé zařízení nově přihlásí k danému tématu, broker mu okamžitě pošle tato uložená data. Tímto může objednatel okamžitě získat momentálně nejnovější hodnotu tématu bez čekání na další aktivitu publishera.
- $d$(publisherID, subscriberID, topic, retain, data) (Deliver) je obdoba objektu $p$, ale pro druhou fázi cesty (od brokera k subscriberům). Oproti předchozímu objektu je zde navíc identifikátor subscribera.

Projděme si postupně jednotlivé objekty. Účelem objektů $c$ a $t$ je aktivace a deaktivace komponent v řídicí vrstvě, odpovídá navázání a zrušení spojení příslušných




Obrázek 4.5: Cesta různých objektů mezi komponentami a brokerem (podle [44])

zařízení s brokerem. Objekt $c$ je vygenerován komponentou jako reakce na zprávu Connect vyslanou příslušným zařízením, podobně objekt $t$ je reakcí na zprávu zařízení žádající ukončení spojení. Při navazování spojení se zařízení autentizuje, tedy posílá autentizační informace. Zde nebudeme řešit formu autentizace, v reálu může jít například o TLS handshake (což by znamenalo poněkud více zpráv než jednu a samozřejmě v obou směrech, zde postup zjednodušíme).

Následuje dvojice objektů $s$ a $u$. Odpovídají zprávám pro objednávku a zrušení objednávky. V obou objektech máme kromě identifikátoru odesílatele název tématu.

Objekt $p$ odpovídá operaci publikování do tématu. Příslušné zařízení buď v pravidelných intervalech nebo na vyžádání (či jinak podle nastavení) odesílá brokerovi data (například meteostanice bude odesílat údaje o teplotě, tlaku, vlhkosti apod.). Podle zprávy je na řídicí vrstvě odpovídající komponentou vytvořen objekt $p$ a odeslán přes membránovou vrstvu brokerovi. Broker zjistí ve své databázi objednávek, které komponenty pro dané téma poslaly objednávku, podle toho vytvoří potřebný počet objektů $d$ – jeden pro každého objednatele (k témuž tématu totiž mohlo poslat objednávku více komponent). Tyto objekty se budou lišit pouze v nově přidaném parametru, identifikátoru příjemce/objednatele. Ostatní parametry jsou stejné jako v původním objektu $p$.

Většina objektů je generována komponentami jako reakce na zprávu odpovídajícího IoT zařízení, jen $d$ je generován brokerem jako reakce na obdržení objektu $p$. Cesta různých druhů objektů je naznačena na obrázku 4.5.

Výše bylo zmíněno, že broker si eviduje různé informace. Na brokerovi tedy potřebujeme mít kromě jiného databázi objenávek (dále bude označena jako *subscriptionsDB*), v níž se dá zjistit, která komponenta má objednáno které téma. Jedná se o vztah M:N (jedna komponenta může mít objednáno více témat a totéž téma si může objednat více komponent), pro zjednodušení budeme databázi reprezentovat množinou uspořádaných dvojic (téma, objednatel).

V membránách potřebujeme pravidla, která budou v rámci P Systému provádět přenos objektů mezi membránami. Nejdřív pravidla pro brokera. Pro každého publishera $i$ publikujícího v tématu $x$ s hodnotou „retain“ nastavenou na $y$ bude mít membrána brokera pravidla

$$p(i, x, y, data) \rightarrow \bigcup_s d(i, s, x, y, data)_{here} \quad \forall (x, s) \in subscriptionsDB$$
$$d(i, s, x, y, data) \rightarrow d(i, s, x, y, data)_{in_s} \quad \text{pro každého objednatele } s$$

První pravidlo nahradí příchozí objekt $p$ tolika objekty $d$, kolik je objednatelů daného tématu, každý vytvořený objekt bude mít jiný druhý parametr určující

objednatele. Původní objekt je tímto odstraněn. Druhé pravidlo přenese všechny vytvořené objekty do membrán odpovídajících objednatelů.

Pro každou komponentu $i$, téma $x$ a hodnotu „retain“ $y$ vytvoříme následující pravidlo (bude se nacházet vždy v membráně komponenty $i$):

$p(i, x, y, data) \rightarrow p(i, x, y, data)_{out}$

Pokud dané zařízení podle své konfigurace a stavu odešle zprávu s publikovanými daty, odpovídající komponenta v řídicí vrstvě vygeneruje objekt $p$ a tento objekt je zpracován právě tímto pravidlem. Jak vidíme, je přenesen ven z membrány dané komponenty a objeví se v hlavní membráně odpovídající brokerovi).

Tato pravidla potřebujeme v příslušné membráně pro všechna témata, v nichž zařízení publikuje, ale mohou existovat i pro témata, v nichž (zatím) nepublikuje.

Pro každou komponentu $i$ a téma $x$ potřebujeme pravidlo

$s(i, x) \rightarrow s(i, x)_{out}$

Podobně jako předchozí, jestliže se v membráně dané komponenty objeví objekt $s$, je expedován ven do membrány odpovídající brokerovi. Taková pravidla opět budeme potřebovat alespoň pro ta témata, do kterých komponenta publikuje.

Zbývají pravidla pro přenos dalších objektů z membrány komponenty $i$ ven do membrány brokera:

$c(i, \text{credentials}) \rightarrow c(i, \text{credentials})_{out}$ s příslušnými přihlašovacími údaji
$t(i) \rightarrow t(i)_{out}$
$u(i, x) \rightarrow u(i, x)_{out}$ pro různá témata $a$

### 4.2.3 Řídicí vrstva

Zatímco na membránové vrstvě řešíme membrány, objekty a pravidla, na řídicí vrstvě budeme s pravidly pracovat pomocí kódu. Budou následovat úseky pseudokódu popisující funkčnost jednotlivých prvků řídicí vrstvy – komponenta a brokera.

Účelem řídicí vrstvy je zprostředkovávání spojení mezi zařízeními a membránovým systémem, tedy zajištění vertikální komunikace v celém systému. Pokud zařízení pošle určitou zprávu, řídicí vrstva ji transformuje na objekt, opatří potřebnými parametry a dodá do odpovídající membrány (tedy objekt se objeví v membráně a může být zpracován membránovým pravidlem). Naopak pokud do membrány dorazí objekt, který nemá být zpracován žádným membránovým pravidlem, je tento

objekt vyzvednut do řídicí vrstvy (tj. z membrány zmizí) a odpovídající komponenta ho zpracuje, případně příslušnému zařízení předá zprávu.

V algoritmu 1 jsou definovány objekty/zprávy, se kterými budeme dále pracovat v kódu uvedeném v následujících algoritmech. Je zde použito dědění – máme rodičovský objekt/zprávu `message` a dále objekty/zprávy odvozené od tohoto rodiče pro různé druhy objektů. Každý objekt/zpráva má svého odesílatele (ID), v dalších parametrech se již liší.

Vlastnosti brokera a jednotlivých komponent jsou popsány v algoritmu 2. V běžných aplikačních IoT protokolech může komponenta publikovat v jakémkoliv množství témat, zde pro zjednodušení povolíme publikování pouze v jednom tématu (`publishTopic`). Ovšem odebírat může v jakémkoliv množství témat (`orderedTopics`).

**Algoritmus 1:** Zprávy a odpovídající objekty (podle [44])

```
// Rodičovská zpráva/objekt s typem a odesílatelem:
message:
  type,  // publish, deliver,...
  ID;    // identifikátor zdroje
// Objekt vygenerovaný publisherem je určen brokerovi (první fáze publikování):
publish (child of: message): ..... p
  topic,
  retain,  // 0 nebo 1
  data;
// Objekt vytvořený brokerem podle p, je určen subscriberovi (druhá fáze):
deliver (child of: publish): ..... d
  subscriberID;
// Objednávka subscribera pro zvolené téma:
subscribe (child of: message): ..... s
  topic;  // téma, které chce odesílatel odebírat
// Odhlášení odběru tématu:
unsubscribe (child of: message): ..... u
  topic;  // téma, od kterého se odesílatel chce odhlásit
// Objekt/zpráva pro navázání spojení:
connect (child of: message): ..... c
  credentials;  // například uživatelské jméno a heslo
// Ukončení spojení:
disconnect (child of: message) ..... t
```

V algoritmu 2 vidíme, že každá entita (ať už běžná komponenta nebo broker) „zná“ své protějšky z okolních vrstev – membránu z membránové vrstvy i zařízení z vrstvy zařízení. Odpovídající membránou k brokerovi je hlavní membrána.

V dalších vlastnostech se již různé entity liší. Zatímco komponenty si evidují informace potřebné k publikování a odebírání objednaných témat, broker si udržuje informace o komponentách, registrovaných tématech, do kterých lze publikovat a která lze odebírat (`topicsDB`), a také momentální stav objednávek (`subscriptionsDB`). Vlastnost `authenticator` můžeme brát jako modul vyřizující požadavky na připojení (zprávy Connect) z bezpečnostního hlediska, tedy kontroluje přihlašovací údaje.

Zaměřme se na algoritmy 3 a 4. Zde najdeme funkce komponent a brokera. Jak bylo výše uvedeno, úkolem komponent řídicí vrstvy je především provádět transformaci zpráv (od zařízení) na objekty (pro membrány) a naopak. Broker dále pracuje

**Algoritmus 2:** Vlastnosti jednotlivých entit v řídicí vrstvě (podle [44])

```
topic:
  topic,
  lastPublisher,
  lastValue;   // poslední publikovaná data v tématu
order:
  topic,
  subscriberID;
// Rodičovská entita pro všechny ostatní entity:
entity:
  ID,   // identifikátor
  membrane,   // odkaz na odpovídající membránu (dolů)
  device;   // odkaz na odpovídající zařízení (nahoru)
component (child of: entity):
  turnedOn,   // 0 (false) nebo 1 (true)
  // Vlastnosti pro publikování:
  publishTopic,
  publishRetain,   // 0 nebo 1
  // Vlastnost pro odběr v tématu:
  topic[] orderedTopics;
broker (child of: entity):
  component[] components,
  authenticator,
  topic[] topicsDB,   // registrovaná témata
  order[] subscriptionsDB;   // databáze objednávek
```

s databázemi, které si vede, tedy například registruje témata a objednávky (což typicky koresponduje s funkčností odpovídajícího zařízení horní vrstvy). Nicméně také bylo zmíněno, že účelem je i jakési vyrovnávání funkčnosti za účelem zvýšení kompatibility v síti – pokud zařízení z horní vrstvy nedokáže komunikovat dostatečně sofistikovaně a neumí pracovat s některými parametry (například retain) nebo nemá plnou implementaci práce s tématy, entita řídicí vrstvy tuto funkcionalitu zastoupí. Stejně tak bezpečnostní funkce (včetně autentizace a autorizace), které nejsou implementovány v horní vrstvě, mohou být zajišťovány řídicí vrstvou.

Broker neprovádí transformaci objektů $p$ na objekty $d$ určené pro jednotlivé odběratele, to je zajištěno pravidly v membránové vrstvě.

**Algoritmus 3:** Funkce komponent (podle [44])

```
// Komponenta přijala objekt od membrány, vytvoří zprávu pro zařízení:
function component.receiveObject(obj)
begin
    if (obj.type == d) then
        device^.processMessage(deliver, obj.publisherID, obj.topic, obj.retain, obj.data)
end
// Komponenta přijala zprávu od zařízení, vytvoří objekt pro membránu:
function component.receiveMessage(mes)
begin
    switch mes.type do
        case connect do
            membrane^.createObject(c, mes.ID, mes.credentials)
        case disconnect do
            membrane^.createObject(t, mes.ID)
        case subscribe do
            membrane^.createObject(s, mes.ID, mes.topic)
        case unsubscribe do
            membrane^.createObject(u, mes.ID, mes.topic)
        case publish do
            membrane^.createObject(p, mes.ID, mes.topic, mes.retain, mes.data)
    end
end
```

Co dál? Systém by se dal obohatit o další typ zpráv/objektů. Zde představujeme pouze základní variantu, na které by měla být patrná funkčnost celého systému, ale jak víme z předchozí kapitoly, aplikační IoT protokoly mívají poněkud

**Algoritmus 4:** Funkce brokera (podle [44])

```
// Broker přijal objekt:
function broker.receiveObject(obj)
begin
    switch obj.type do
        case c do
            if (authenticator.check(obj.ID, obj.credentials)) and
              (device^.processMessage(connect, obj.ID, obj.credentials)) then
                components[obj.ID].turnOn();
        case d do
            device^.processMessage(disconnect, obj.ID);
            components[obj.ID].turnOff();
        case s do
            device^.processMessage(obj.ID, obj.topic);
            subscriptionsDB.add(obj.topic, obj.ID);
            if topicsDB.retainSet(obj.topic) then
                membrane^.createObject(d, topicsDB.lastPublisher(obj.topic),
                  obj.ID, obj.topic, 1, topicDB.lastValue(obj.topic));
        case u do
            subscriptionsDB.remove(obj.topic, obj.ID);
            device^.processMessage(obj.ID, obj.topic);
        case p do
            if obj.retain then topicsDB.storeData(obj.topic, obj.data);
            else topicsDB.unsetRetain(obj.topic) ;
    end
end
```

bohatší portfolium zpráv. Především z důvodu stability, škálovatelnosti a bezpečnosti systému by bylo vhodné přidat zprávy ACK, tedy potvrzování. Po obdržení jakékoliv zprávy z dříve jmenovaných by mělo následovat potvrzení, aby původní odesílatel měl jistotu, že zpracování jeho zprávy proběhlo bez problémů.

Cílem této sekce je představit mechanismus nezávislý na existujících protokolech (třebaže jimi inspirovaný), který může být naprogramován v programovacích jazycích určených pro membránové systémy (například P-Lingua[1]) nebo prostě v některém z jazyků pro programování síťových aplikací.

[1] Web projektu je https://github.com/RGNC/plingua [cit. 2022-09-10].

## 4.3 Membránový firewall

Firewall je filtr síťového provozu – pakety buď pustí dál nebo zahodí (a provede něco dalšího, například zapíše informaci do logu). Činnost firewallu je určena sadou pravidel.

Pokud budeme chtít v našem systému implementovat firewall, můžeme prakticky na kterékoliv vrstvě, třebaže jeho umístění má vliv na to, jakým způsobem bude do provozu ve skutečnosti zasahovat. Jestliže zvolíme membránovou vrstvu a necháme firewall pracovat pomocí membránových pravidel, pak při případné implementaci využijeme stejný programovací jazyk, kterým jsme programovali P Systém. Výhodnější však bude nechat firewall pracovat na řídicí vrstvě (ale ovlivňovat membránovou vrstvu), protože membránová pravidla sama o sobě toho moc neumožňují a nedokážou pracovat se stavy a kontextem.

V každém případě (na membránové i řídicí vrstvě) je výhodou, že se firewall dostane k naprosto veškeré komunikaci (protože membrány nám zde modelují přenos dat) a navíc ke všem potřebným informacím, najde je v parametrech objektů.

Na řídicí vrstvě máme pro firewall dvě základní možnosti ovlivňování provozu. Buď bude od komponent či brokera odebírat objekty či zprávy, které považujeme za bezpečnostní riziko, nebo bude zasahovat do membránové vrstvy a ovlivňovat (přidávat či odebírat) membránová pravidla. První možnost by znamenala mnohem větší zatížení výpočetního systému, protože by firewall pracoval prakticky neustále (což odpovídá tomu, jak běžné firewally v našich sítích pracují). Druhá možnost se jeví zajímavější, protože filtrování bude fungovat „průtokově“ podobně jako analogová zařízení – pokud pro daný objekt není v membráně pravidlo, tak se prostě dál nedostane. Samotný firewall bude pracovat pouze ve chvílích, kdy mu změníme konfiguraci a on tyto změny bude muset promítnout do množiny membránových pravidel.

Dobře, tak tedy zvolíme druhou možnost. I zde je v případě zasílání publikovaných dat na výběr:

- Ovlivňujeme pravidlo, které v hlavní membráně (odpovídající brokerovi) nahrazuje objekt $p$ sadou objektů $d$ pro jednotlivé subscribery. Pokud chceme zakázat konkrétnímu subscriberovi odebírat data z daného tématu, jednoduše odstraníme z pravé strany pravidla objekt určující jako cíl právě toho subscribera. Ostatní odběratelé v pravidle zůstanou. Pokud bychom naopak chtěli nově konkrétnímu subscriberovi odběr z tématu povolit, přidáme odpovídající objekt na pravou stranu příslušného pravidla.

- Můžeme přidávat či odebírat pravidla pro objekty $d$ v hlavní membráně (pravidla přenášející tyto objekty do podřízených membrán).

První možnost nabízí poměrně značnou pružnost systému – například můžeme zajistit, aby v případě, že do tématu přispívá více publisherů, mohl konkrétní subscriber odebírat data pouze od některých z nich: v hlavní membráně bude v tomto případě několik pravidel pro totéž téma (ale s jiným ID publishera), objekt omezovaného subscribera bude na pravých stranách jen některých z těchto pravidel.

Zatímco první možnost zasahuje do pravých stran pravidel, ale počet pravidel zůstává nezměněn, druhá možnost odstraňuje (nebo přidává) celá pravidla. To můžeme považovat za nevýhodu. Naopak výhodou druhé možnosti je, že odfiltrovanou zprávu můžeme dále zprocesovat, například přenést do speciálně pro tento účel vytvořené membrány – karantény, ve které objekt počká na prověření administrátorem nebo některým automatickým nástrojem k tomu určeným.

Předpokládejme, že zvolíme první možnost, jen s určitou obměnou: k zásahu do membránových pravidel bude docházet vždy, když od některého subscribera dorazí brokerovi objednávka tématu. Firewall posoudí, zda tuto žádost vyřídit kladně či záporně, a pokud kladně, pak zasáhne do pravidel (přidá na pravou stranu pravidel vytvářejících objekty $d$ pro dané téma objekt žádajícího subscribera). Na řídicí vrstvě budeme vést databázi s konfigurací firewallu, podle které se firewall bude rozhodovat. Tedy pokud přijde objednávka, tedy objekt $s$, provede se následující kód:

```
if firewall.allow(obj.ID, obj.topic) then
  membrane^.adjustRules_addSubscription(obj);
```

Funkce membrane^.adjustRules_addSubscription(obj) najde v membráně pravidlo nahrazující objekt $p$ pro dané téma a na pravou stranu přidá objekt $d$ s uvedením žádajícího subscribera jako cíle.

Pokud bychom chtěli rozlišit povolené objednávky nejen podle tématu, ale také podle zdroje publikovaných dat, potřebovali bychom v databázi firewallu evidovat i tuto informaci, a kód uvedené funkce by byl trochu delší (musel by protřídit pravidla nejen podle tématu, ale také podle zdroje publikovaných dat).

Vhodná reakce by měla být také na zprávy odhlášení z tématu a ukončení spojení – objekt $d$ pro daného subscribera by měl být z pravidel zase odebrán. To je preventivní opatření pro případ, že se do sítě dostane zařízení, jehož ID je (v důsledku chyby nebo útoku) stejné jako ID některého dříve odhlášeného zařízení.

# Seznam literatury


[1] About Z-Wave Technology [online]. *Z-Wave Alliance* [cit. 2022-08-08]. Dostupné z: https://z-wavealliance.org/about_z-wave_technology/

[2] Al-fuqaha, Ala, Mohsen Guizani, Mehdi Mohammadi, Moussa Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 2015, Volume 17(4). Pages 2347–2376. DOI: 10.1109/COMST.2015.2444095. ISSN 1553-877x.

[3] Augustin, Aloÿs, et al. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. *Sensors*, 2016, Volume 16(9), 1466. ISSN 1424-8220 (IF 2,677). [cit. 2022-05-10] Dostupné také z: http://www.mdpi.com/1424-8220/16/9/1466/htm

[4] Bagad, Vilas S. a Iresh A. Dhotre. *Computer Networks*. Fourth Revised Edition. Pune: Technical Publications, 2009. ISBN 978-81-843-1563-9.

[5] Buzachis, A., Boruta, D., Villari, M. and Spillner, J. Modeling and Emulation of an Osmotic Computing Ecosystem Using Osmotic Toolkit. *Australasian Computer Science Week Multiconference*, New Zealand (2021). DOI 10.1145/3437378.3444366.

[6] Busi, Nadia. Causality in Membrane Systems. In: *Membrane Computing*. WMC 2007. Lecture Notes in Computer Science, vol 4860. Springer, 2007. pp. 160–171. ISBN 978-3-540-77312-2. DOI https://doi.org/10.1007/978-3-540-77312-2_10

[7] Cunha, André, Anis Koubaa, Ricardo Severino, Mário Alves. Open-ZB: an open-source implementation of the IEEE 802.15.4/ZigBee protocol stack on

TinyOS. In: *IEEE 4th International Conference on Mobile Adhoc and Sensor Systems, MASS 2007*. Pisa: 2007. pp. 1–12. 10.1109/MOBHOC.2007.4428602.

[8] DA SILVA, Mario Marques. *Multimedia communications and networking*. Boca Raton: CRC Press, Taylor & Francis group, 2012. ISBN 978-1-4398-7484-4.

[9] DATTA, Soumya KANTI, Christian BONNET. Next-Generation, Data Centric and End-to-End IoT Architecture Based on Microservices. 2018 *IEEE International Conference on Consumer Electronics – Asia (ICCE-Asia)*, 2018, pp. 206–212, doi: 10.1109/ICCE-ASIA.2018.8552135.

[10] DIZDAREVIĆ, Jasenka, Francisco CARPIO, Admela JUKAN, Xavi Masip-Bruin. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *Association for Computing Machinery*. New York, NY, USA, 2019, 51(6), 29 p. ISSN 0360-0300.

[11] EGLI, P. *MQTT – Message Queueing Telemetry Transport: Introduction to MQTT, a protocol for M2M and IoT applications*. Zurich University of Applied Sciences (2017). DOI: 10.13140/RG.2.2.13210.54721.

[12] GUPTA, Lokesh. HTTP Methods [online]. *Rest API Tutorial* [cit. 2022-08-04]. Dostupné z: https://restfulapi.net/http-methods/

[13] HANÁK, Drahomír. Stopařův průvodce REST API [online]. *itnetwork.cz* [cit. 2022-08-01]. Dostupné z: https://www.itnetwork.cz/programovani/nezarazene/stoparuv-pruvodce-rest-api/

[14] HEZAM, Tameem Abdulbaset Abdulwahid. Overview of Future Internet Architecture Projects and Protocols and Applications. Final Academic Report [online]. *Figshare* [cit. 2022-07-29], 2021. DOI 10.6084/m9.figshare.14938434.v1. Dostupné z: https://www.researchgate.net/publication/353142150_OVERVIEW_OF_FUTURE_INTERNET_ARCHITECTURE_PROJECTS_AND_PROTOCOLS_AND_APPLICATIONS

[15] HOPCROFT, John E., Rajeev MOTWANI a Jeffrey D. ULLMAN. *Introduction to automata theory, languages, and computation*. 3. ed., New international ed. Harlow: Pearson Addison-Wesley, 2014. ISBN 978-129-2039-053.

[16] HOSSEN, Md. Sakhawat, Razib H. KHAN, Abdullah AZFAR. Interconnection between 802.15.4 Devices and IPv6: Implications and Existing Approaches [online]. In: *IJCSI International Journal of Computer Science Issues*, Vol. 7, Issue 1, No. 1. 2010. ISSN 1694-0814

[17] How Content Delivery Networks Work? [online] *Beluga CDN* [cit. 2022-09-06]. Dostupné z: https://www.belugacdn.com/how-content-delivery-networks-work/

[18] Hura, Gurdeep S. a Mukesh Singhal. *Data and computer communications: networking and internetworking*. Boca Raton, FL: CRC Press, 2001. ISBN 08-493-0928-X.

[19] IETF. Internet Standards: RFCs [online]. *IETF.org* [cit. 2022-07-07]. Dostupné z: https://www.ietf.org/standards/rfcs/

[20] Kassab, Wafaa, Khalid Darabkh. A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations. *Journal of Network and Computer Applications*, vol. 163 (2020), ISSN 1084-8045, 49 p., DOI: doi.org/10.1016/j.jnca.2020.102663

[21] Kozierok, Charles M. *The TCP/IP Guide* [online]. [cit. 2022-09-14] Dostupné na: http://www.tcpipguide.com/free/index.htm

[22] Lammle, Todd. *CCNA: výukový průvodce přípravou na zkoušku 640-802*. Brno: Computer Press, 2010, 928 s. ISBN 978-802-5123-591.

[23] Lasser, Jon. *Rozumíme UNIXu*. Praha: Computer Press, 2002, 252 s. ISBN 80-722-6706-X.

[24] Malý, Martin. REST: architektura pro webové API [online]. *Zdroják.cz* [cit. 2022-08-01]. Dostupné z: https://zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[25] Manzoor, Jawad, Drago, Idilio a Sadre, Ramin. (2016). The curious case of parallel connections in HTTP/2. *The 12th International Conference on Network and Service Management (CNSM)*, 2016, pp. 174–180, DOI 10.1109/CNSM.2016.7818414.

[26] Mekki, Kais, Frédéric Chaxel, Fernand Meyer. A Comparative Study of LPWAN Technologies for Large-scale IoT Deployment. *ICT Express*, Volume 5, 2019. pp. 1–7. DOI: 10.1016/j.icte.2017.12.005

[27] Mikhaylov, Konstantin, Nikolaos Plevritakis, Jouni Tervonen. Performance Analysis and Comparison of Bluetooth Low Energy with IEEE 802.15.4 and SimpliciTI. *Journal of Sensor and Actuator Networks* 2(3), 2013. ISSN 2224-2708.

[28] Eclipse Mosquitto: An open source MQTT broker [online]. *Eclipse Mosquitto* [cit. 2022-08-23]. Dostupné z: https://mosquitto.org/

[29] MQTT-smarthome [online]. *Github* [cit. 2022-07-25]. Dostupné z: https://github.com/mqtt-smarthome/mqtt-smarthome

[30] OASIS Message Queuing Telemetry Transport (MQTT) TC [online]. *OASIS Open* [cit. 2022-07-25]. Dostupné z: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt

[31] OASIS MQTT Committee. MQTT Specifications [online]. *OASIS* [cit. 2022-08-03]. Dostupné z: https://mqtt.org/mqtt-specification/

[32] PĂUN, Gheorghe, Grzegorz ROZENBERG. A Guide to Membrane Computing. In: *Theor. Comp. Science* 287(1), Elsevier Science Publishers Ltd., 2002. pp. 73–100. ISSN 0304-3975.

[33] PĂUN, Gheorghe. *Membrane Computing: An Introduction*. New York: Springer, 2002. 420 p. ISBN 978-3-540-43601-0.

[34] PĂUN, Gheorghe, Grzegorz ROZENBERG, Arto SALOMAA. *The Oxford Handbook of Membrane Computing*. New York: Oxford University Press, 2010. ISBN 0199-55667-9.

[35] PETERKA, Jiří. *eArchiv* [online]. [cit. 2022-08-02] Dostupné z: http://www.earchiv.cz

[36] Připojení k IoT síti LoRaWAN. *České radiokomunikace* [online]. [cit. 2022-08-09]. Dostupné z: https://www.cra.cz/pripojeni-k-iot-siti-lorawan

[37] PUŽMANOVÁ, Rita. *TCP/IP v kostce*. 2., upr. a rozš. vyd. České Budějovice: Kopp, 2009, 619 s. ISBN 978-80-7232-388-3.

[38] SHARMA, Vishal, et al. Managing Service-Heterogeneity using Osmotic Computing. *International Conference on Communication, Management and Information Technology* (ICCMIT 2017), 7 stran, Warsaw, Poland, 2017. arXiv:1704.04213

[39] *Sigfox Česká republika* [online]. [cit. 2022-08-09]. Dostupné z: https://sigfox.cz

[40] Sigfox. LoRa. NB-IoT. Easy guide to who does it best [online]. *IoT Solutions* [cit. 2022-08-18], Marsa, Malta: 2022. Dostupné z: https://www.iotsolutions.com.mt/post/sigfox-vs-lora-vs-nb-iot-who-s-doing-it-best

[41] SOSINSKY, Barrie A. *Mistrovství – počítačové sítě*. Brno: Computer Press, 2010. ISBN 978-802-5133-637.

[42] Tennina, Stefano et al. *IEEE 802.15.4 and ZigBee as Enabling Technologies for Low-Power Wireless Systems with Quality-of-Service Constraints*. Springer, edice Springer Briefs in Electrical and Computer Engineering, 2013. ISBN 978-3-642-37367-1.

[43] *The P Systems Webpage* [online]. [cit. 2022-05-12]. Dostupné z: http://psystems.eu

[44] Vavrečková, Šárka. Membrane System as a Communication Interface between IoT Devices. *Proceedings of the 22th Conference Information Technologies – Applications and Theory (ITAT 2022)*, Slovakia, CEUR Workshop Proceedings, vol. 3226, 2022. pp. 184–190. ISSN 1613-0073. Dostupné také z: http://ceur-ws.org/Vol-3226/ [cit. 2022-09-30]

[45] Vavrečková, Šárka. Modeling Communication in Internet of Things Network using Membranes. *Proceedings of the 21th Conference Information Technologies – Applications and Theory (ITAT 2021)*, Slovakia, CEUR Workshop Proceedings, vol. 2962, 2021. pp. 195–201. ISSN 1613-0073. Dostupné také z: http://ceur-ws.org/Vol-2962/ [cit. 2022-04-12]

[46] Vavrečková, Šárka. *Počítačová síť a internet* [online]. [cit. 2022-09-14] Slezská univerzita v Opavě, 2017, 196 stran. ISBN: 978-80-7510-245-4. Dostupné z: http://vavreckova.zam.slu.cz/pocsit.html

[47] Vavrečková, Šárka. The Spanning-Tree Algorithm and Generating a Membrane Structure. In Holeňa, Martin; Horváth, Tomáš; Kelemenová, Alica; Mráz, František; Pardubská, Dana; Plátek, Martin; Sosík, Petr. *Proceedings of the 20th Conference Information Technologies – Applications and Theory (ITAT 2020)*. 2718. vyd. Slovensko: CEUR Workshop Proceedings, 2020. s. 201–208. ISSN 1613-0073. Dostupné také na: http://ceur-ws.org/Vol-2718/paper26.pdf [cit. 2022-09-14]

[48] Vavrečková, Šárka. Využití membránového systému pro simulaci komunikace v síti Internetu věcí. In: *Kognice a umělý život XX, sborník z 20. sborníku konference*. Gabriela Šejnová, Michal Vavrečka, Juraj Hvorecký (Eds). Praha: ČVUT, 2022. ISBN 978-80-01-07007-9.

[49] Villari, Massimo, et al. Osmotic computing: A new paradigm for edge/cloud integration. In *IEEE Cloud Computing* 3.6 (2016) 76–83.

[50] Villari, Massimo, et al. Software Defined Membrane: Policy-Driven Edge and Internet of Things Security. In *IEEE Cloud Computing* 4(4), pp. 92–99, July/August 2017, doi: 10.1109/MCC.2017.3791014.

[51] VOJÁČEK, Antonín. Průmyslová komunikace OPC UA: 1.díl – popis protokolu [online]. *Automatizace HW* [cit. 2022-08-18]. Dostupné z: https://automatizace.hw.cz/prumyslova-komunikace-opc-ua-1dil-popis-protokolu.html

[52] Z-Wave Tutorial-frequency, frame, protocol, PHY, MAC, Z-Wave security basic tutorial [online]. *RF Wireless World* [cit. 2022-08-08]. Dostupné z: https://www.rfwireless-world.com/Tutorials/z-wave-tutorial.html

[53] ZigBee: The full-Stack Solution for All Smart Devices. ZigBee Specification [online]. *Connectivity Standards Alliance* [cit. 2022-08-08]. Dostupné z: https://csa-iot.org/all-solutions/zigbee/

# Seznam obrázků


2.1 Protokolová datová jednotka . . . . . 16
2.2 Referenční model ISO/OSI . . . . . 18
2.3 Horizontální a vertikální komunikace v ISO/OSI . . . . . 20
2.4 Srovnání modelů RM ISO/OSI a TCP/IP . . . . . 25
2.5 Rozdíly mezi verzemi protokolu HTTP v komunikaci klient-server: klient je vždy vlevo, server vpravo . . . . . 27
2.6 Hodnota Sequence Number ve zprávě ICMP Echo a tatáž hodnota pro odpověď ICMP Echo reply . . . . . 31
2.7 Vložený paket ve zprávě ICMP Time exceeded . . . . . 33
2.8 Hodnota Transaction ID v DNS zprávě . . . . . 34

3.1 Ukázka modelu Publish-Subscribe: nejdřív registrace k odběru témat, pak plnění objednávek . . . . . 38
3.2 Srovnání webového protokolového zásobníku a protokolových zásobníků pro IoT . . . . . 39
3.3 Topologie v ZigBee síti . . . . . 57
3.4 Protokolový zásobník IEEE 802.15.4, ZigBee . . . . . 58
3.5 Protokoly v ZigBee bráně . . . . . 61

4.1 Příklad membránové struktury . . . . . 71
4.2 Ukázka reprezentace membránové struktury pomocí stromu . . . . . 71
4.3 Komunikační architektura se třemi vrstvami . . . . . 74
4.4 Příklad membránové struktury pro IoT síť . . . . . 75
4.5 Cesta různých objektů mezi komponentami a brokerem . . . . . 77

# Seznam zkratek

| | |
|---|---|
| 3GPP | The 3rd Generation Partnership Project |
| 6LoWPAN | (IPv)6 Low-Power Wireless Personal Area Network |
| ACK | ACKnowledgement |
| AES | Advanced Encryption Standard |
| APL | APplication Layer |
| AMQP | Advanced Message Queuing Protocol |
| API | Application Programming Interface |
| ATM | Asynchronous Transfer Mode |
| AUTH | Authentication |
| BLE | Bluetooth Low Energy, Bluetooth LE |
| BPSK | Binary-Phase Shift Keying |
| CBA | Commercial Building Automation |
| CCA | Clear Channel Assessment |
| CDMA | Code Division Multiple Access |
| CDN | Content Delivery Network |
| CIFS | Common Internet File System |
| CoAP | Constrained Application Protocol |
| CON | CONfirmation, CONfirmable |
| CONNACK | CONNection ACKnowledgement |
| CSA | Connectivity Standards Alliance |
| CSMA/CA | Carrier Sense, Multiple Access/Collision Avoidance |
| CSS | Chirp spread spectrum |

| | |
|---|---|
| DCCP | Datagram Congestion Control Protocol |
| DDS | Data Distribution Service |
| DoD Model | Department of Defense Model |
| DTLS | Datagram Transport Layer Security |
| EIA | Electronic Industries Alliance |
| ETSI | European Telecommunications Standard Institute |
| FDMA | Frequency Division Multiple Access |
| FTP | File Transfer Protocol |
| GDS | Global Data Space |
| GPRS | General Packet Radio Service |
| GSM | Groupe Spécial Mobile (z francouzštiny) |
| HA | Home Automation |
| HTTP | HyperText Transfer Protocol |
| HTTPS | HyperText Transfer Protocol Secure |
| IAB | Internet Architecture Board |
| IANA | Internet Assigned Numbers Authority |
| ICI | Interface Control Information |
| ICMP | Internet Control Message Protocol |
| ICT | Information and Communication Technologies |
| IDS | Intrusion Detection System |
| IDU | Interface Data Unit |
| IEC | International Electrotechnical Commission |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| IKE | Internet Key Exchange |
| IM | Instant Messaging |
| IoT | Internet of Things |
| IP | Internet Protocol |
| IPM | Industrial Plant Monitoring |
| IPSec | Internet Protocol Security |
| IPX | Internetwork Packet Exchange |
| ISM | Industrial, Scien-tific, Medical |
| ISO | International Organization for Standardization |
| ISOC | Internet Society |
| ITU | International Telecommunications Union |

| | |
|---|---|
| JSON | JavaScript Object Notation |
| LAN | Local Area Network |
| LDAP | Lightweight Directory Access Protocol |
| LE | Low Energy |
| LLC | Logical Link Control |
| LoRaWAN | Long Range Wide Area Network |
| LoWPAN | Low-Power Wireless Personal Area Network |
| LPWAN | Low-Power Wide Area Network |
| LTE | Long Term Evolution |
| M2M | Machine-to-Machine |
| MAC | Media Access Control |
| MAN | Metropolian Area Network |
| MEL | MicroELement |
| MPLS | Multiprotocol Label Switching |
| MQ | Message Queue |
| MQTT | MQ Telemetry Transport |
| MQTT-SN | MQTT Sensor Network |
| NB-IoT | Narrow-Band Internet of Things |
| NFC | Near-Field Communication |
| NON | NON-confirmable |
| NWK | NetWorK Layer |
| OASIS | Organization for the Advancement of Structured Information Standards |
| OFDMA | Orthogonal Frequency Division Multiple Access |
| OMG | Object Management Group |
| OPC UA | Open Platform Communications – Unified Architecture |
| OSI | Open Systems Interconnection |
| OSPF | Open Shortest Path First |
| P2P | Peer-to-Peer |
| PAN | Personal Area Network |
| PDU | Protocol Data Unit |
| PHHC | Personal Home and Hospital Care |
| PHY | PHYsical Layer |
| PPP | Point-to-Point Protocol |

| | |
|---|---|
| QoS | Quality of Service |
| QPSK | Quaternary Phase Shift Keying |
| QUIC | Quick UDP Internet Connections |
| REST | Representational State Transfer |
| RFC | Request for Comments |
| RSSI | Received Signal Strength Indication |
| RST | ReSeT |
| SAP | Service Access Point |
| SASL | Simple Authentication and Security Layer |
| SATA | Serial AT Attachment |
| SCTP | Stream Control Transmission Protocol |
| SDM | Software Defined Membrane |
| SDU | Service Data Unit |
| SF | Spreading Factor |
| SIM | Subscriber Identity Module |
| SMB | Server Message Block |
| SMTP | Simple Mail Transfer Protocol |
| SPX | Sequenced Packet Exchange |
| SSL | Secure Sockets Layer |
| TCP | Transmission Control Protocol |
| TDoA | Time Difference of Arrival |
| TFTP | Trivial File Transfer Protocol |
| TIA | Telecommunication Industries Association |
| TLS | Transport Layer Security |
| TTL | Time to Live |
| TTN | The Things Network |
| TTS | The Things Stack |
| UDP | User Datagram Protocol |
| UMTS | Universal Mobile Telecommunications System |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| USB | Universal Serial Bus |
| VoIP | Voice over IP |
| VPN | Virtual Private Network |

| | |
|---|---|
| W3C | World Wide Web Consortium |
| WAN | Wide Area Network |
| WPAN | Wireless Personal Area Network |
| WSN | Wireless Sensor Network |
| WWW | World Wide Web |
| XML | eXtensible Markup Language |
| XMPP | eXtensible Message and Presence Protocol |
| ZB | ZigBee |

**Procesy a věci v počítačové síti**
**RNDr. Šárka Vavrečková, Ph.D.**

© Filozoficko-přírodovědecká fakulta v Opavě,
Slezská univerzita v Opavě, 2022

Recenzenti: Doc. RNDr. PaedDr. Hashim Habiballa, Ph.D., PhD.
Doc. RNDr. Petr Bujok, Ph.D.

Obálka: Anna Novotná

Vydavatel: Slezská univerzita v Opavě
Na Rybníčku 626/1, 746 01 Opava

Tisk: Profi-tisk group s.r.o.
Chválkovická 223/5, Chválkovice, 779 00 Olomouc

Vydání: první
Rok vydání: 2022
Náklad: 150 výtisků

ISBN: 978-80-7510-520-2 (online)
978-80-7510-521-9 (tisk)

Publikace je neprodejná.

Tato publikace byla podpořena Strukturálními a investičními fondy Evropské unie OPVVV z projektu „Zvýšení kvality vzdělávání na Slezské univerzitě v Opavě ve vazbě na potřeby Moravskoslezského kraje“, CZ.02.2.69/0.0/0.0/18_058/0010238.



ISBN 978-80-7510-520-2