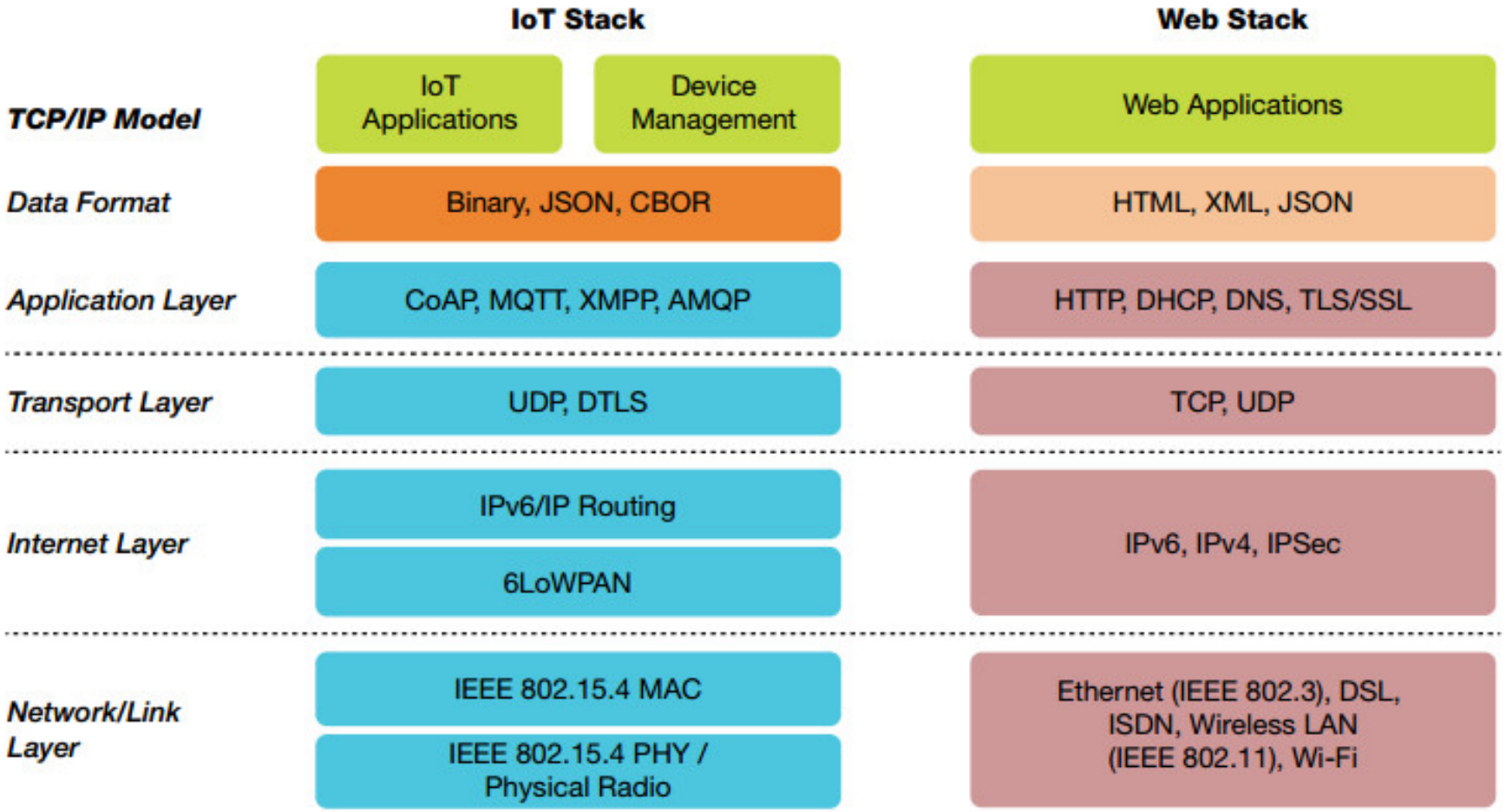
Návrh systémů IoT

2. Modely komunikace a komnunikační rozhraní

Stanislav Vítek
Katedra radioelektroniky
České vysoké učení technické v Praze

IoT protocol stack



Formáty výměny dat - JSON

JavaScript Object Notification

- kolekce párů název:hodnota
- 4 primitivní datové typy (řetězec, číslo, log. hodnota, NULL)
- 2 strukturované datové typy (objekt, pole)

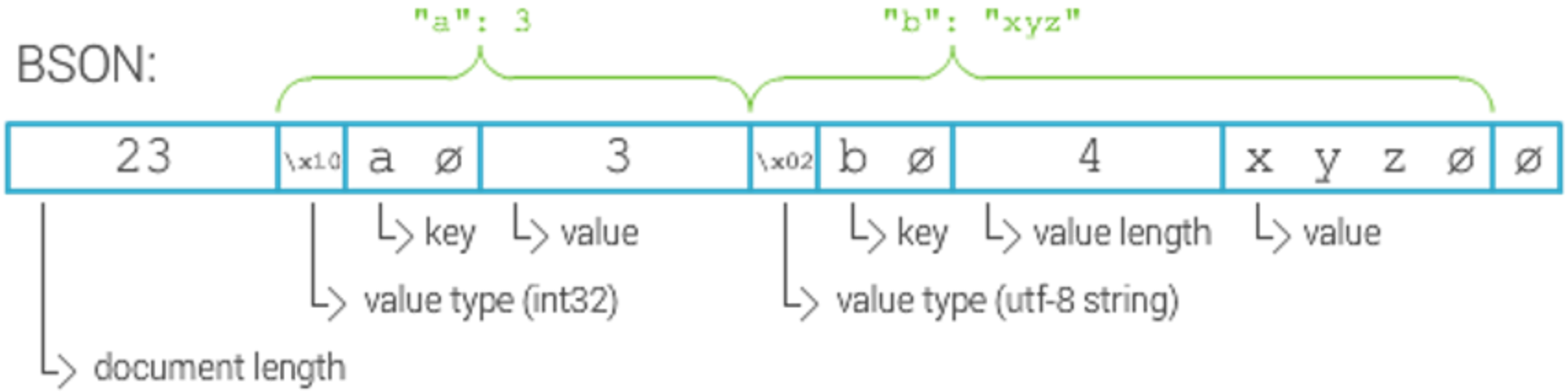
```
{"members":[
  {
    "name": "John Lennon",
    "birthdate": ["year": 1940, "month": 10, "day": 9]
  },
  {
    "name": "Paul McCartney",
    "birthdate": ["year": 1942, "month": 6, "day": 18]
  }
]}
```

Formáty výměny dat - Binary JSON (BSON)

JSON:

```
{
  "a": 3,
  "b": "xyz"
}
```

BSON:



Protokoly relační/aplikační vrstvy

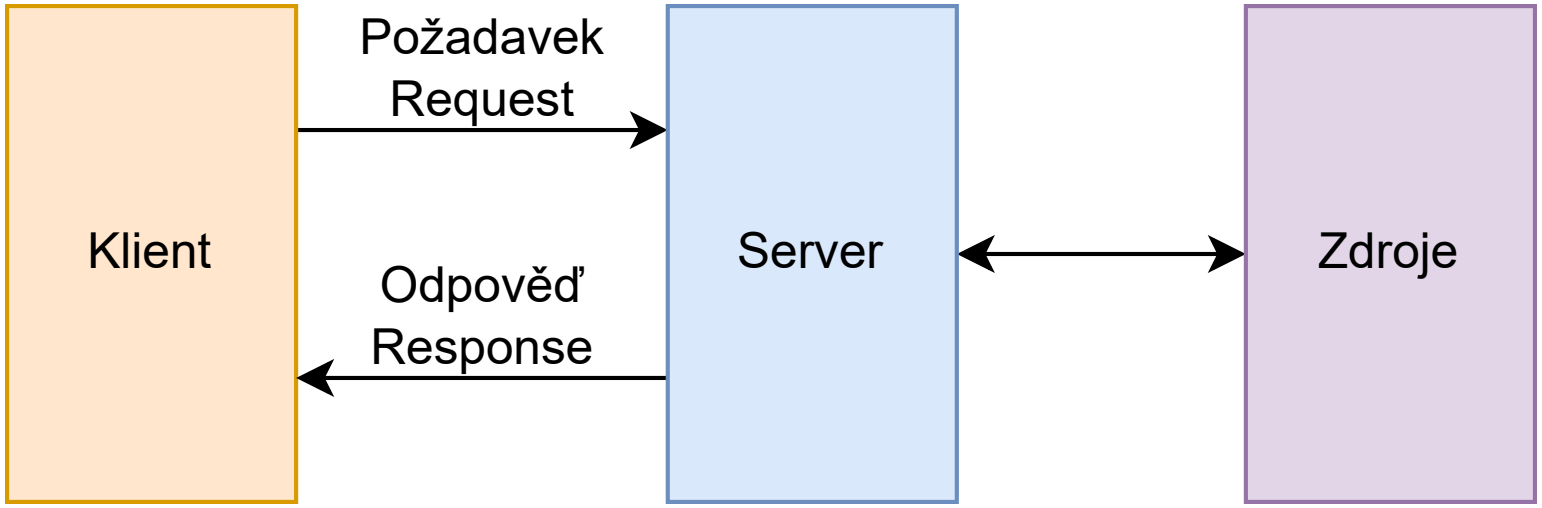
1. Poskytování abstrakce "zprávy" (elementární jednotky dat)komunikace mezi koncovými body internetu věcí).
2. Poskytování primitiv pro datovou komunikaci/výměnu zpráv aplikacím vyšší vrstvy internetu věcí.
3. Implementace specifických síťových paradigmat (např. Publish-Subscribe nebo Request-Response).
4. Poskytování dodatečných mechanismů spolehlivosti nebo zabezpečení.
5. Někdy přizpůsobení již existujících (nikoliv nativně M2M) řešení.

Komunikační modely a protokoly

1. **Request-Response**
    - HTTP
    - CoAP
2. **Publish-Subscribe**
    - MQTT, AMQP
3. **Push-Pull**
4. **Exclusive Pair**

Komunikační model Request-Response 1/2

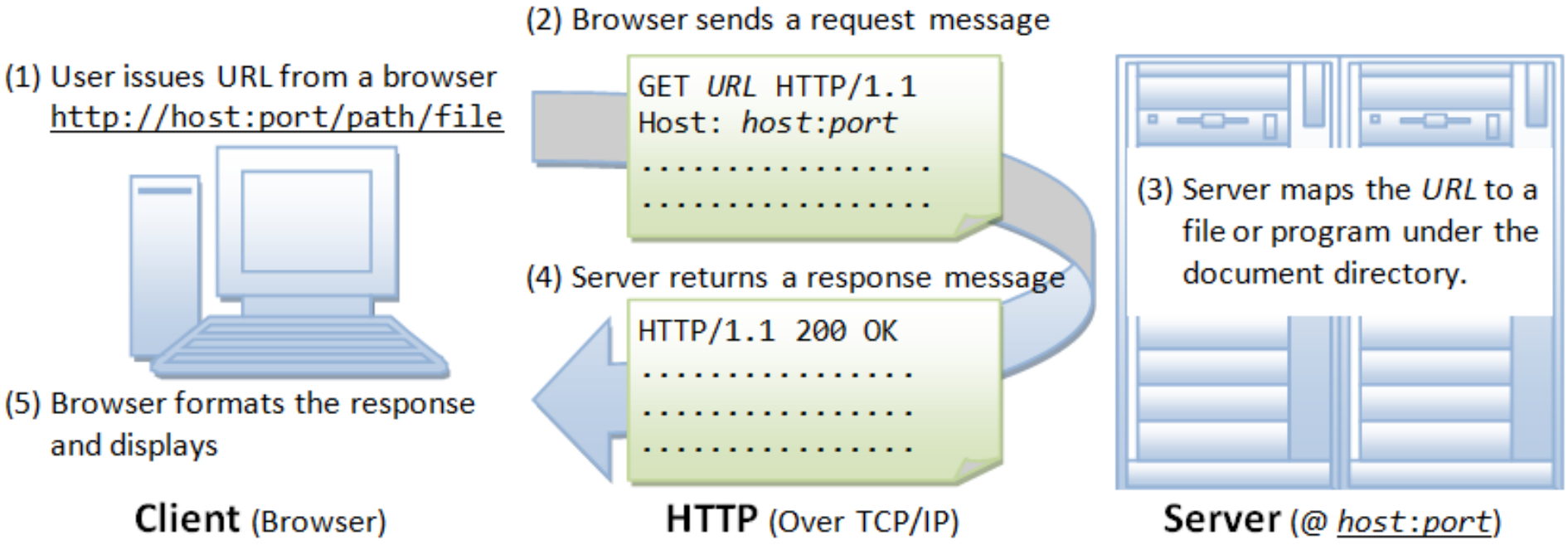
- Aplikace (klient) posílá požadavky službě (server)
- Server obdrží požadavek, rozhodne se jak odpoví, načte data, načte zdroj reprezentace (šablona odpovědi), sestaví odpověď a odešle ji klientovi



Komunikační model Request-Response 2/2

- Model Request-Response je bezstavový model. Každá dvojice požadavek-odpověď je nezávislá na ostatních.
- Příkladem je protokol HTTP
  - HTTP funguje jako protokol dotaz-odpověď mezi klientem a serverem.
  - Klientem může být webový prohlížeč a serverem aplikace v počítači, která podporuje webové stránky.
  - Klient (prohlížeč) odešle serveru požadavek HTTP a server vrátí klientovi odpověď.
- Další příklady: CoAP (Constrained Application Protocol)

Protokol HTTP - Hypertext Transfer Protocol



- typicky TCP/IP spojení, nepodporuje UDP
- podpora přenosu binárních dat (obrázky, videa)
- bezestavový
- možnost mezipaměti (cache)

Pozn. 1:

Stavový a bezestavový protokol

Bezestavový protokol

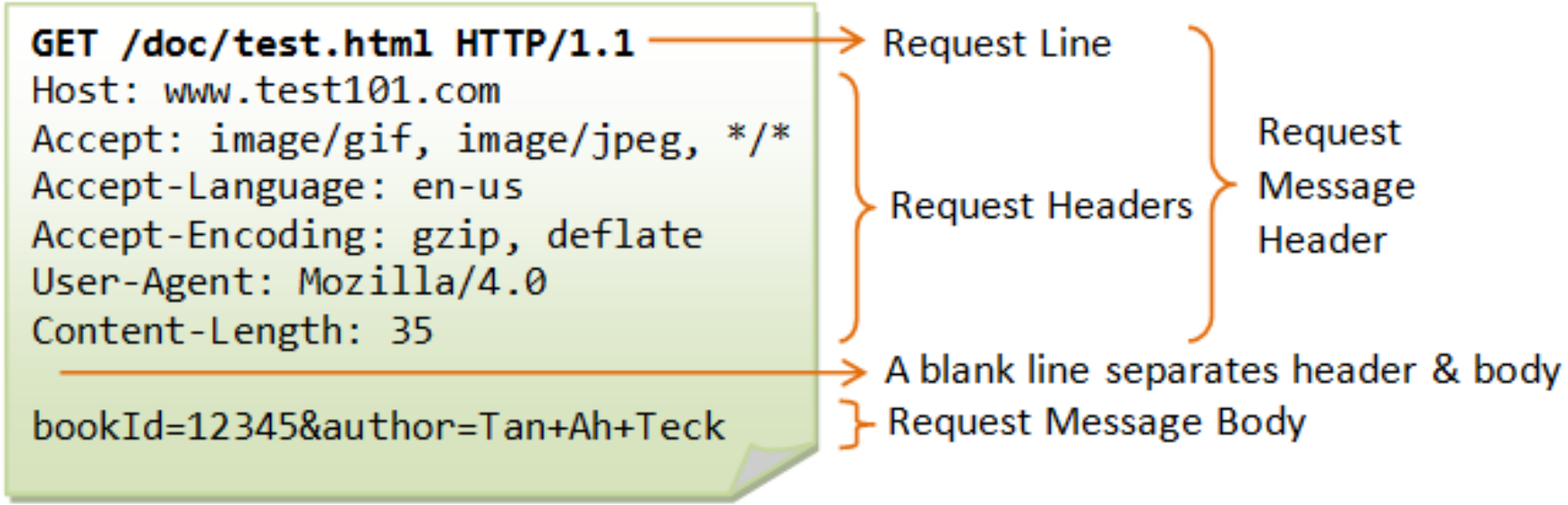
- Jedná se o síťový model, ve kterém klient pošle serveru požadavek a server na oplátku pošle zpět odpověď podle aktuálního stavu stejně jako v modelu Request-Response.
- Server není povinen uchovávat informace o relaci nebo stavu každého komunikačního partnera pro více požadavků.
- Jejich implementace na internetu je velmi snadná.
- Bezstavové protokoly fungují lépe, když dojde k havárii, protože není třeba obnovovat žádný stav, selhávající server lze po havárii jednoduše restartovat.
- Příklady: HTTP (Hypertext Transfer Protocol), UDP (User Datagram Protocol), DNS (Domain Name System).

Stavový protokol

- Pokud klient odešle serveru požadavek a server neodpoví, pak klient pošle serveru požadavek znovu.
- Stavové protokoly jsou zřejmě náročnější na implementaci
- Servery musí uchovávat informace o stavu a další podrobnosti o relaci, takže je možné v případě selhání komunikace přenos obnovit
- Příklady: FTP (File Transfer Protocol), Telnet.

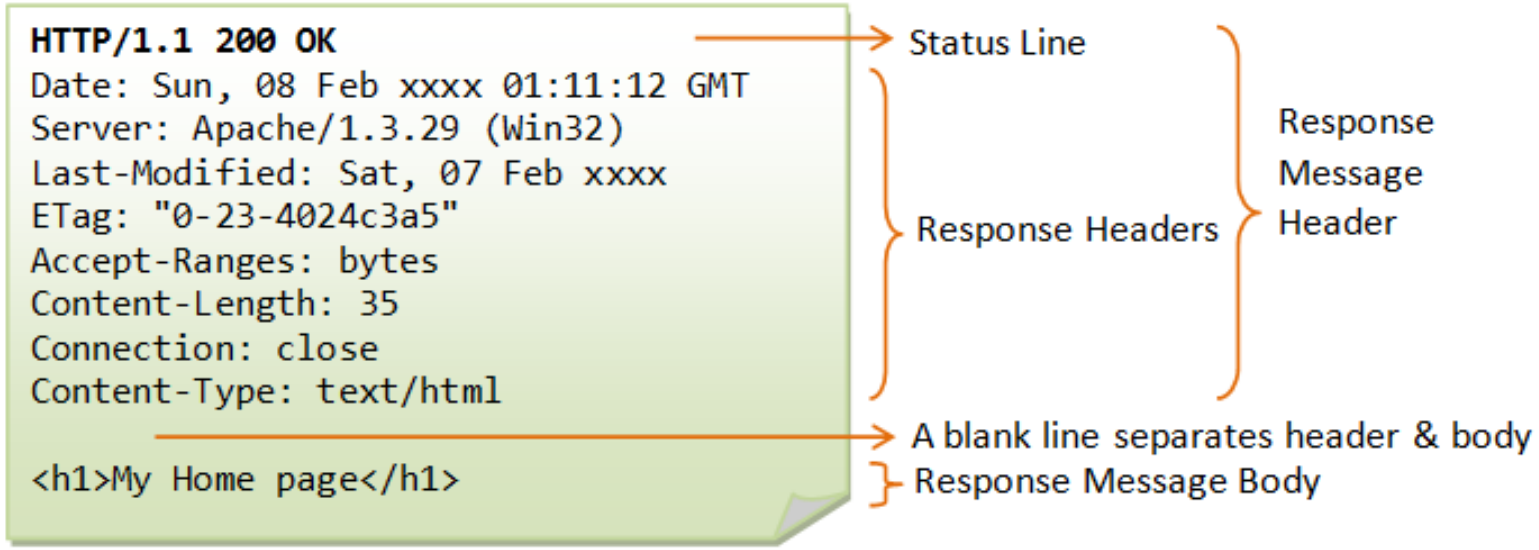
HTTP požadavek

- Hlavička
  - Request line - identifikuje HTTP metodu, URI a verzi protokolu
  - Request headers
- Tělo požadavku



HTTP odpověď

- Hlavička
  - Status line - identifikuje verzi protokolu a kód odpovědi
  - Response headers
- Tělo



HTTP metody

- **GET** Klient může použít požadavek GET k získání webového zdroje ze serveru.
- **HEAD** Klient může použít požadavek HEAD k získání hlavičky, kterou by získal požadavkem GET. Protože hlavička obsahuje datum poslední změny dat, lze ji použít ke kontrole proti místní kopii mezipaměti.
- **POST** Slouží k odeslání dat na webový server.
- **PUT** Požádá server o uložení dat.
- **DELETE** Požádá server o odstranění dat.
- **TRACE** Požádá server, aby vrátil diagnostickou stopu provedených akcí.
- **OPTIONS** Požádejte server, aby vrátil seznam metod požadavků, které podporuje.
- **CONNECT** Slouží k tomu, aby proxy server navázal spojení s jiným hostitelem a jednoduše odpověděl na obsah, aniž by se jej pokusil analyzovat nebo uložit do mezipaměti. Často se používá k navázání spojení SSL prostřednictvím proxy.

Kódy HTTP odpovědí 1/2

- **1xx** - téměř nepoužívaný
- **2xx** - úspěch
  - **200 OK** - requests succeeded, usually contains data
  - **201 Created** - returns a Location header for new resource
  - **202 Accepted** - server received request and started processing
  - **204 No Content** - request succeeded, nothing to return
- **3xx** - přesměrování
  - **304 Not Modifed** - resource not modifed, cached version can be used

Kódy HTTP odpovědí 2/2

- 4xx - client error
  - 400 Bad Request - malformed syntax
  - 401 Unauthorized - authentication required
  - 403 Forbidden - server has understood, but refuses request
  - 404 Not Found - resource not found
  - 405 Method Not Allowed - specified method is not supported
  - 409 Conflict - resource conflict with client data
  - 415 Unsupported Media Type - server does not support media type
- 5xx - server error
  - 500 Internal Server Error - server encountered error and failed to process request

Pozn. 2:

Architektury komunikačních rozhraní

REST - Representational State Transfer

- Zaměřen na zdroje systému (Resources) a na to, jak jsou stavy zdrojů adresovány a přenášeny. Je to styl **architektury**, ne **standard**.
- Zdroje jsou vytvářeny (Create), čteny (Read), aktualizovány (Update) a mazány (Delete) - CRUD - pomocí bezestavového protokolu HTTP
- Klient přistupuje ke zdrojům prostřednitvím koncového bodu URI a využívá HTTP metody
  - **GET** - získání existujího zdroje
  - **POST** - vytvoření nového zdroje
  - **PUT** - aktualizace existujícího zdroje
  - **PATCH** - částečná aktualizace existujícího zdroje
  - **DELETE** - smazání zdroje

REST - hlavní principy

- **Bezestavový** - server neudržuje žádný stav mezi požadavky od klienta.
- **Klient-server architektura** - klient a server musí být od sebe odděleny, aby se mohly vyvíjet nezávisle.
- **Možnost ukládání do mezipaměti** - data získaná ze serveru by měla být kešovatelná buď klientem, nebo serverem.
- **Jednotné rozhraní** - server poskytuje jednotné rozhraní pro přístup ke zdrojům bez definování jejich reprezentace.
- **Vrstvený systém** - klient může přistupovat ke zdrojům na serveru nepřímo prostřednictvím dalších vrstev, jako je proxy nebo load balancer.
- **Kód na vyžádání** (volitelný) - server může klientovi předat kód, který může spustit, například JavaScript pro jednostránkovou aplikaci.

RPC - Remote Procedure Call

- RPC umožňuje lepší definici sémantiky
- Napodobuje volání lokálních procedur přes síťové rozhraní
- Je velmi častou volbou v případě architektury mikroslužeb
- Volba protokolu je libovolná (protocol agnostic), lze přizpůsobit aplikaci
  - Při použití HTTP se používají pouze metody GET a POST
- Významnou implementací je gRPC od Google
- Rozmazává hranici mezi serverem a klientem - server může být klientem a klient serverem (protože prostě jen volají procedury)

Protokoly vyšších vrstev vs. IoT

- Použití HTTP protokolu je pro IoT problematické
  - Omezené možnosti IoT prostředí (zejména konektivita)
  - HTTP hlavička reprezentuje nejméně 71B
- Problémy s firewally
  - Skrytý zdroj zprávy příchozí z NAT sítí
  - v IoT sítích ale potřebujeme vědět, kdo data posílá
- Dokumenty založenné na XML jsou příliš objemné

COAP - Constrained Application Protocol

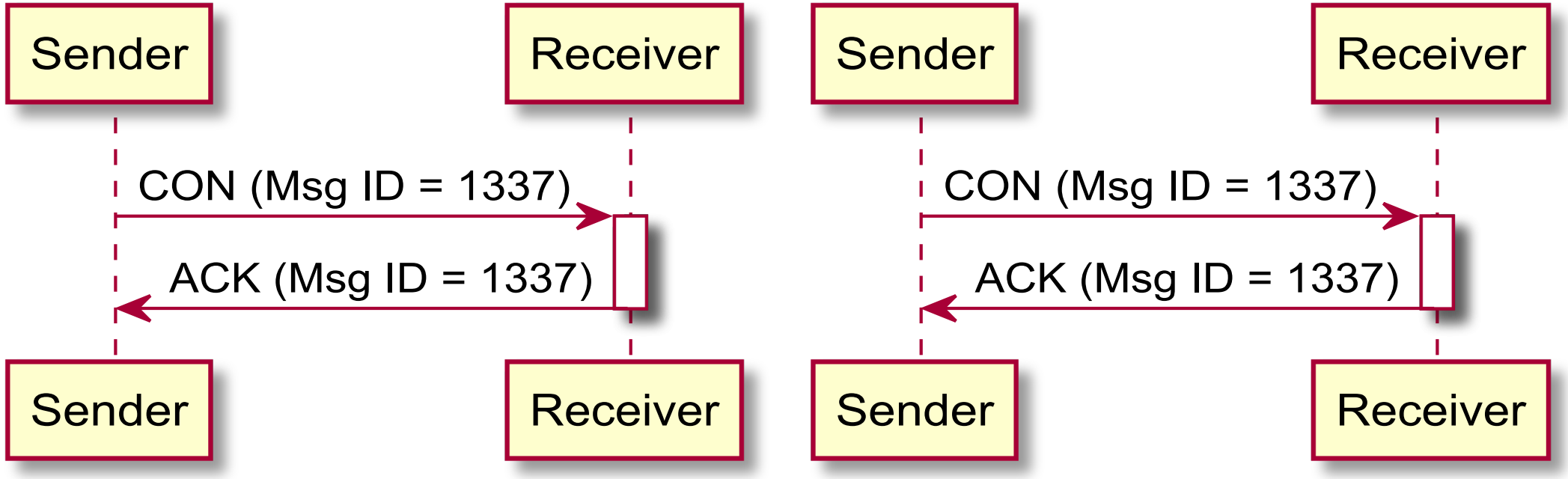
- Protokol založený na modelu REST
- Manipuluje se zdroji pomocí stejných metod jako HTTP
- Využívá UDP transportní protokol
- Režie protokolu TCP je příliš vysoká a jeho řízení toku není vhodné pro krátkodobé transakce.
- UDP má nižší režii a podporuje vícesměrové vysílání, ale datagramy
  - se mohou ztratit
  - mohou být duplikovány
  - mohou dorazit v nesprávném pořadí

COAP protokol

- Rozdělen do dvou dílčích vrstev
  - request/response
    - GET, PUT, POST a DELETE
  - zprávy
    - SUCCESS, CLIENT ERROR, SERVER ERROR
- Čtyři typy zpráv:
  - S potvrzením (confirmable) - vyžaduje ACK
  - Bez potvrzení (non-confirmable) - není třeba ACK
  - Potvrzení (ACK)
    - Reset - indikuje, že byla přijata zpráva, ale chybí kontext pro zpracování
    - Prázdná - pouze hlavička o velikosti 4B

COAP protokol

Příklad komunikace



COAP protokol

- Každý dotaz má unikátní URI
- Specifikace a detaily: https://coap.me/
- Rozdíly proti HTTP:
  - Založen na protokolu UDP (ale lze použít volitelné mechanismy pro zvýšení spolehlivosti, tj. potvrzování zpráv + zpětné přenosy).
  - Asynchronní paradigma požadavek/odpověď
  - Jiná (kratší) hlavička paketu (viz další slide)
  - Mechanismy zjišťování služeb (service discovery) a proxy

COAP protokol

2 bit	2 bit	4 bit	4 bit	16 bit
Ver	T	TKL	Code	Message ID
Token (if any, TKL bytes) ...				
Options (if any) ...				
1 1 1 1 1 1 1 1			Payload (if any) ...	

- Token - hodnota pro zjištění korelace mezi dotazem a odpovědí
- Message ID - páruje CON a ACK zprávy

COAP protokol

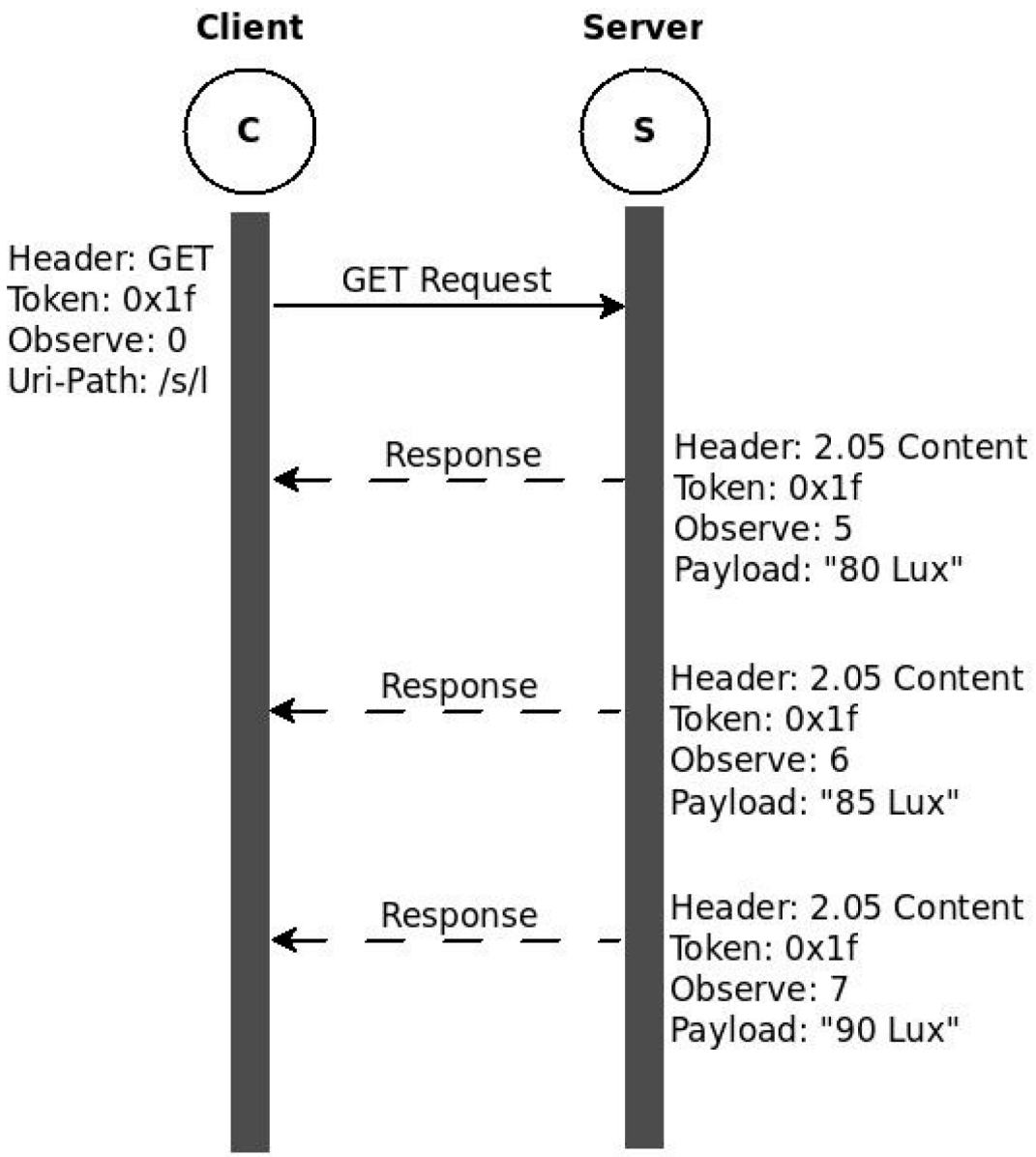
CoAP implementuje některé jednoduché mechanismy spolehlivosti

1. Detekce duplicit pro potvrzené (CON) i nepotvrzené (NON) zprávy
2. Jednoduchá spolehlivost opakovaného přenosu typu stop-and-wait s exponenciálním zpětným přenosem pro potvrzitelné zprávy
    - Odesílatel přeposílá zprávu Confirmable v exponenciálně rostoucích intervalech, dokud neobdrží ACK (nebo zprávu RST) nebo dokud nevyčerpá všechny pokusy.

COAP protokol

Mechanismus OBSERVE umožňuje implementovat mechanismus odběru dat

1. Klient požádá o prostředek (GET) s polem Observe Option.
2. Server přidá klienta do seznamu pozorovatelů daného prostředku.
3. Při každé změně cílového prostředku server informuje všechny jeho pozorovatele.

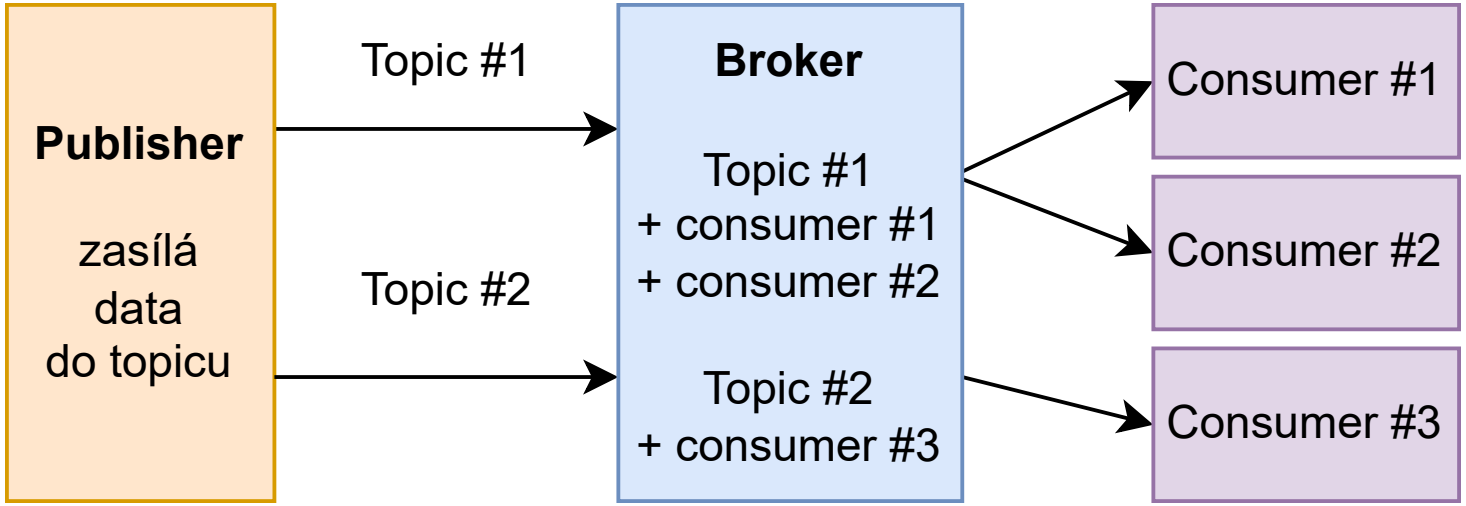


COAP protokol

- CoAP se při zabezpečení komunikace mezi klientem a serverem spoléhá na protokoly nižší vrstvy.
- Šifrování zpráv na vrstvě TSP (DTLS - Datagram Transport Layer Security) nebo na síťové vrstvě (IPSec).
- Vzhledem k tomu, že protokol CoAP realizuje podmnožinu funkcí protokolu HTTP/1.1, vztahují se bezpečnostní hlediska protokolu HTTP i na protokol CoAP. Kromě toho CoAP představuje některé jedinečné zranitelnosti
    - Proxy servery jsou ze své podstaty man-in-the-middle.
    - Riziko zesílení zpráv a útoků DDoS.
    - Podvržení IP kvůli absenci handshake v UDP.

Komunikační model Publish-Subscribe

- Zahrnuje tři účastníky:
  - vydavatel (publisher) - posílá data v rámci téma (topic)
  - spotřebitel (consumer) - přihlašuje (subscribe) k odběru téma, o zdroji nic neví
  - zprostředkovatele (mediator, broker) - posílá spotřebiteli data v rámci téma
- Role vydavatel/spotřebitel jsou čistě logické



Vlastnosti modelu Publish-Subscribe

1. Interakce many-to-many

- Stejná informace může být doručena ve stejný okamžik různým spotřebitelům.
- Každý spotřebitel dostává informace od různých producentů.

3. Oddělení v prostoru

- Interagující strany se nemusí navzájem znát.
- Adresování zpráv je na základě jejich obsahu.

3. Oddělení v časové

- Interagující strany se nemusejí aktivně účastnit komunikace ve stejnou dobu.

MQTT

Message Queue Telemetry Transport Protocol

- Lehký protokol pro zasílání zpráv určený pro M2M
- Navržen pro zařízení s omezenou šířkou pásma, nespolehlivou konektivitou a sítě s vysokou latencí
- Původně jej navrhli Andy Stanford-Clark (IBM) a Arlen Nipper v roce 1999 pro propojení telemetrických systémů ropovodů přes satelit.
- Vydán v roce 2010 (bez licenčních poplatků)
- Od roku 2014 jako standard OASIS.
- aktuální specifikace 3.1/3.1.1

MQTT

- Je určen pro sítě TCP/IP (používá TCP)
- Pro sítě bez TCP existuje implementace MQTT-SN (NS - sensor network)
- ISO/OSI: skutečnosti úroveň 5-7 - aplikační, prezentační a relační vrstva
- Používá datově agnostický textový protokol
- Zajišťuje spolehlivost
- Poskytuje některé mechanismy pro zajištění doručení

MQTT broker

- Broker je vlastně server pro obě části (vydavatele a odběratele).
- Zajišťuje QoS
- Může uchovávat zprávy (data)
  - Vydavatel rozhoduje o tom, zda si broker zprávu ponechá.
  - V tomto případě každý odběratel při odběru automaticky obdrží nejnovější hodnotu, takže Broker udržuje jakýsi "stav".
- Předává obsah zájemcům z řad odběratelů (obsah pochází od vydavatelů)

MQTT data

- Zprávy (obsah) jsou uspořádány do témat ve formě stromové struktury
  - jako je adresářová cesta, oddělovačem je / (lomítko)
- Odběratel se může přihlásit k odběru konkrétního tématu nebo může použít zástupný vzor pro odběr různých témat:
  - [#] znamená celou větev
  - [+] znamená jednoúrovňové

Příklad:

- Vydavatel vydává např: **CVUT/FEL/209/Sensor/Temperature**
- Odběratel se přihlásí k odběru: **CVUT/FEL/+/Senzor/#**
- Odběratel bude upozorněn vždy, když zařízení odešle informaci o jakémkoli měření (tj. teploty, ale také vlhkosti) provedeném **/Senzor/** někde v budově **CVUT/FEL** (může to být místnost, ale také třeba chodba)

MQTT QoS

Level 0 - Neuznaná služba

- Doručeno každému odběrateli maximálně jednou
- Stejné garance doručení jako TCP

Level 1 - Uznávaná služba

- Zajišťuje doručení zprávy alespoň jednou.
- Broker očekává potvrzení, jinak zprávu znovu odešle

Level 2 - Zajištěná služba

- Dvoustupňové doručení
- Zajišťuje, že zpráva je každému účastníkovi doručena přesně jednou

Další vlastnosti MQTT

- Příznak čisté relace (volitelný) - trvanlivá připojení:
  - Pokud je **true**, Broker odstraní všechny klientské odběry při odpojení klienta.
  - Pokud je false, spojení zůstane nečinné a všechny zprávy se shromažďují (QoS v závislosti na typu připojení) a doručeny, jakmile je připojení obnoveno.
- Klient může brokerovi nařídit, aby ho nechal odeslat konkrétní téma (nebo témata), když se objeví neočekávané spojení.
  - Zjištění selhání/havárie: vhodné např. pro kritické a bezpečnostní systémy
- Bezpečnost
  - Slabá - uživatelská jména a hesla zasílána v prostém textu
  - Lze využít zabezpečený kanál (SSL/TLS)

MQTT protokol

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

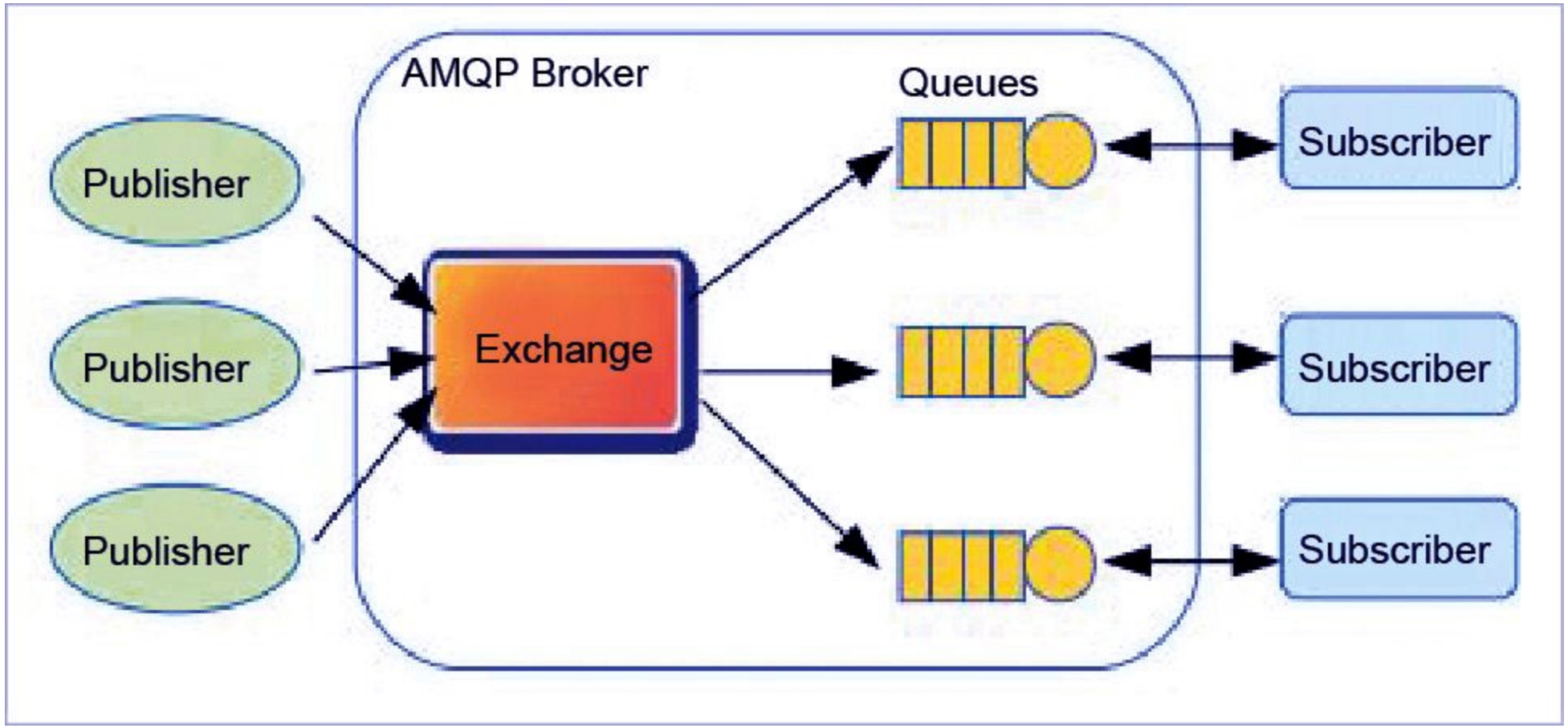
Mnemonic	Enumeration	Description
Reserved	0	Reserved
CONNECT	1	Client request to connect to Server
CONNACK	2	Connect Acknowledgment
PUBLISH	3	Publish message
PUBACK	4	Publish Acknowledgment
PUBREC	5	Publish Received (assured delivery part 1)
PUBREL	6	Publish Release (assured delivery part 2)
PUBCOMP	7	Publish Complete (assured delivery part 3)
SUBSCRIBE	8	Client Subscribe request
SUBACK	9	Subscribe Acknowledgment
UNSUBSCRIBE	10	Client Unsubscribe request
UNSUBACK	11	Unsubscribe Acknowledgment
PINGREQ	12	PING Request
PINGRESP	13	PING Response
DISCONNECT	14	Client is Disconnecting
Reserved	15	Reserved

AMQP - Advanced Message Queuing Protocol

- Otevřený standardní protokol pro aplikace orientované na zprávy.
- Podporuje interoperabilitu systémů v distribuovaných prostředích.
- Založen na protokolu TCP s dalšími mechanismy spolehlivosti (at-most-once, at-least-once nebo once-delivery).
- Podporuje jak komunikaci typu point-to-point, tak komunikační paradigmata publish-subscribe (jako MQTT).
- Programovatelný protokol: některé entity a schémata směrování jsou primárně definovány aplikacemi.

AMQP architektura

- Na rodíl od MQTT vytváří fronty zpráv, kde zpráva čeká, dokud ji odběratel nepřečte

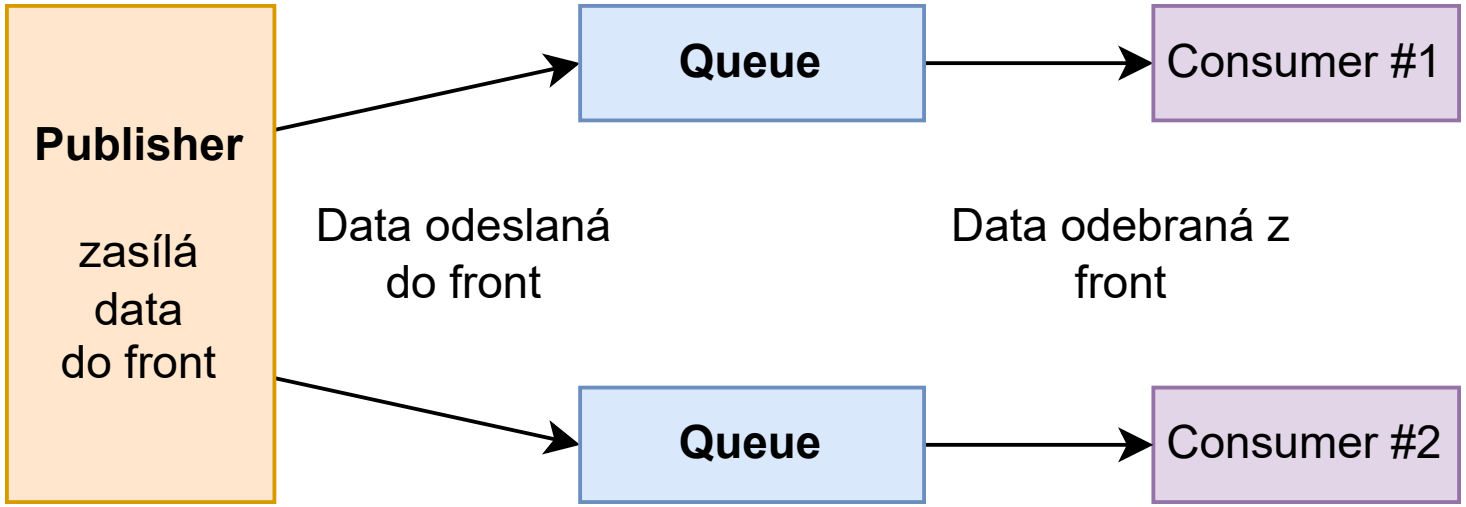


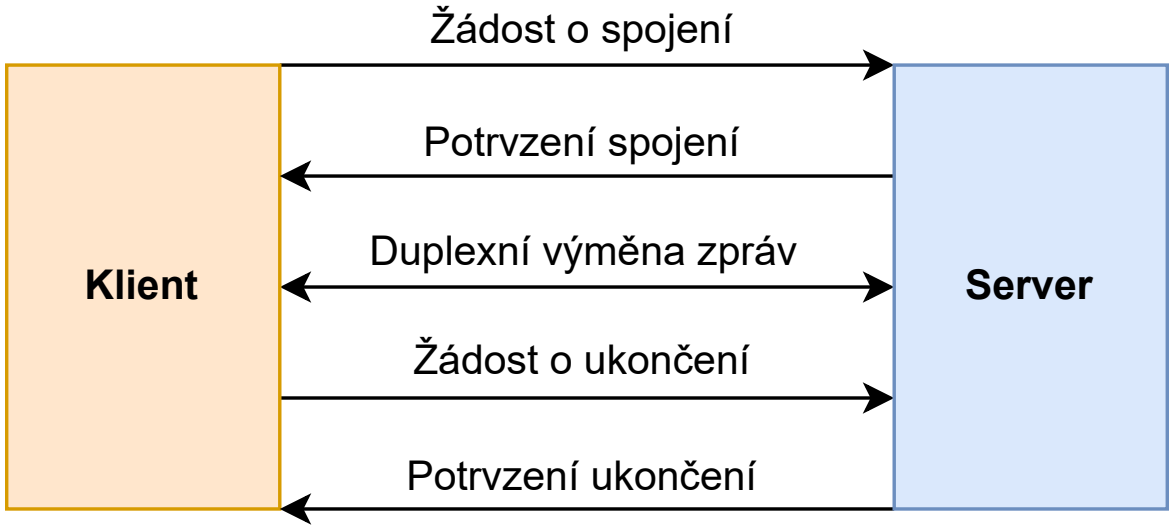
AMQP komunikace

- Pravidla, kterými se výměna řídí při směrování.
  - **Přímá výměna** - doručuje zprávy do front na základě klíče pro směrování
  - **Fanout Exchange** - doručuje zprávy do všech front, které jsou k němu vázány.
  - **Topic Exchange** - doručuje zprávy do jedné nebo více front na základě přiřazení témat.
  - **Výměna hlaviček** - doručuje zprávy na základě více atributů vyjádřených jako hlavičky
- Protokol definuje dva typy zpráv
  - Holé zprávy, které dodává odesílatel.
  - Zprávy s poznámkami, které se zobrazují u příjemce.

Komunikační model Push-Pull

- Producenti dat odesílaji data do front a konzumenti je z front odebírají
- Producenti nemusí vědět o konzumentech
- Fronty fungují jako vyrovnávací paměť, řeší nesoulad rychlostí vkládání a odebírání



Komunikační model Exclusive-Pair

- Obousměrný, plně duplexní, používá trvalé spojení mezi klientem a serverem.
- Spojení zůstává otevřené, dokud klient odešle požadavek na ukončení.
- Klient a server mohou po navázání spojení navzájem posílat zprávy .