VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. Karel Pokorný

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

LABORATORNÍ SCÉNÁŘE OBJASŇUJÍCÍ ZÁKLADY KOMUNIKAČNÍCH PROTOKOLŮ
LABORATORY SCENARIOS EXPLAINING THE BASICS OF COMMUNICATION PROTOCOLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Karel Pokorný**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jan Jeřábek, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Karel Pokorný ***ID:*** 175233
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Laboratorní scénáře objasňující základy komunikačních protokolů

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku komunikačních protokolů a zejména různých režimů přenosu v paketových sítích. Navrhněte a popište dva komplexní scénáře, na kterých bude možné si vyzkoušet tyto režimy přenosu a nastavovat různé atributy v různých situacích. Tyto scénáře budou vhodné jako laboratorní úlohy pro předměty zaměřené na komunikační technologie. Předpokládá se realizace v jazyce Java ve vhodném prostředí. Výstupem práce budou dva kompletní scénáře včetně podrobných návodů pro studenty, prokonzultovaných s vedoucím práce, předpřipravených výchozích situací, jednoduchého grafického prostředí pro nastavení scénáře, komentovaných zdrojových kódů, doplňujících úkolů pro studenty a vzorového řešení. Předpokládá se, že délka realizace jedné úlohy bude pro studenta přibližně 2 hodiny času.

DOPORUČENÁ LITERATURA:

[1] FOROUZAN, Behrouz A. TCP/IP protocol suite. 4th ed. Boston: McGraw-Hill Higher Education, 2010, xxxv, 979 s. ISBN 978-0-07-337604-2.

[2] JEŘÁBEK, J. Komunikační technologie. Skriptum FEKT Vysoké učení technické v Brně, 2018. s. 1-172.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: doc. Ing. Jan Jeřábek, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Cílem této práce bylo navrhnout dva komplexní scénáře se zaměřením na různé režimy přenosu v paketových sítích. První scénář se zaměřuje na ARQ (Automatic Repeat Request) protokoly. Součástí laboratorní úlohy je seznámení se s protokoly Stop-and-Wait, Go Back N, Selective Repeat a jejich vzájemné porovnání. Druhý scénář se věnuje typům přenosu v paketových sítích. Laboratorní úloha obsahuje porovnání jednosměrného, vícesměrového, všesměrového a výběrového přenosu. Oba scénáře jsou podpořeny aplikací simulující jednotlivé metody či protokoly a virtuálním prostředím, pomocí kterých jsou rozdíly v daných technologiích demonstrovány.

KLÍČOVÁ SLOVA

ARQ, Stop-and-Wait, Go Back N, Selective Repeat, Směrování, Unicast, Multicast, Anycast, Broadcast, Paketové sítě, Komutace paketů, Java, Docker, VirtualBox

ABSTRACT

The goal of this thesis was to design two complex scenarios with focus on different kinds of transmission in packet-switched networks. First scenario is about ARQ (Automatic Repeat Request) protocols. It consists of introduction to Stop-and-Wait, Go Back N and Selective Repeat protocols and their comparison. Second scenario compares unicast, multicast, broadcast and anycast transmission methods. Both scenarios use applications which can simulate particular methods or protocols. These applications along with virtual environments are used for demonstration of characteristics of these methods/protocols.

KEYWORDS

ARQ, Stop-and-Wait, Go Back N, Selective Repeat, Routing, Unicast, Multicast, Anycast, Broadcast, Packet-switched network, Packet switching, Java, Docker, VirtualBox

POKORNÝ, Karel. *Laboratorní scénáře objasňující základy komunikačních protokolů*. Brno, Rok, 108 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Jan Jeřábek, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Laboratorní scénáře objasňující základy komunikačních protokolů“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Janu Jeřábkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Obsah

Seznam obrázků

Úvod

Pochopení technik řízení toku dat a vypořádání se s chybovými stavy je nezbytné pro dosažení efektivně pracující sítě.

Princip znovuodeslání datové jednotky je vhodný nástroj pro případ, kdy se nějaký segment po cestě ke svému cíli ztratí. Je však potřeba si uvědomit, že v případě, kdy odesílatel neobdržel potvrzení od příjemce z důvodu jeho zahlcení, stává se znovuodesílaní nepotvrzeného segmentu nežádoucím a potencionálně kritickým prvkem komunikace, kdy se akorát zvyšuje vytížení. Pro zamezení výskytu těchto problémů existují techniky implementující do komunikace mechanismus, který dovoluje přijímací stanici dynamicky regulovat rychlost, kterou k ní odesílá zdroj svá data. Tyto techniky mohou být implementovány na spojové nebo vyšší vrstvě.

Cílem této práce je navrhnutí dvou komplexních laboratorních scénářů, které se budou zaměřovat na režimy přenosu dat v paketových sítích. Teoretický rozbor a samotné scénáře si kladou za cíl seznámit s problematikou řízení toku dat a chybových stavů na spojové úrovni, jež mohou nastat. Hlavní náplní je především představení a následné porovnání ARQ (Automatic Repeat reQuest) algoritmů Stop-and-Wait, Go Back N a Selective Repeat. Student příjde do kontaktu s výchozí implementací a zkusí si doimplementovat jak klíčové myšlenky těchto technik, tak i chování pro různé chybové stavy. Druhou část práce tvoří scénáře laboratorní úlohy kladoucí si za cíl objasnění různých typů přenosu dat v paketových sítích a doimplementování programu, který simuluje přenos v režimech unicast, multicast, broadcast a anycast, a poukázat na jejich vazby se specifickými IPv4 i IPv6 adresami.

1 Komutace v sítích

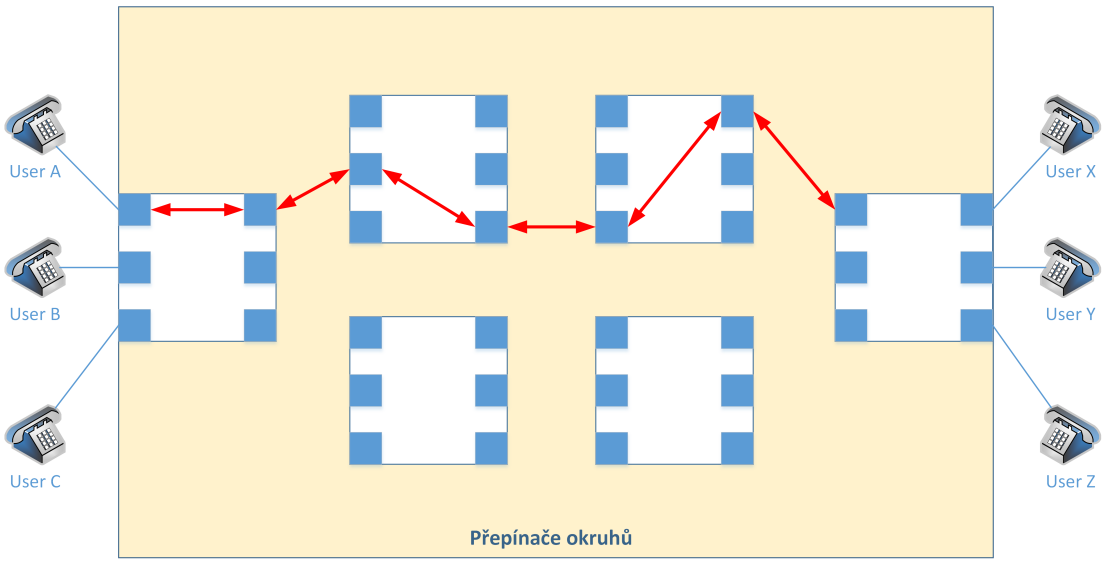
Komunikace v datových sítích je realizována *přepojováním*, což je proces přeposílání dat, které přicházejí z jednoho portu do jiného portu, který vede k cílovému zařízení[18]. Obecně se komunikace dělí do dvou základních kategorií:

- Nespojovaná - data jsou přeposílána na základě tabulek, které ví, kam v daný moment data přeposlat
- Spojovaná - před samotným zahájením přenosu je potřeba sestavit cestu, kterou se datová komunikace uskuteční (cesta může být po dokončení přenosu zrušena, nebo může být ponechána pro budoucí přenos)

1.1 Přepojování okruhů

Přepojování okruhů je starší technologií využívanou od počátků telefonie, a stala se tak základnou pro tradiční telefonní sítě. V režimu přepojování okruhů je dočasně dedikován spoj, který zaručuje, že po dobu trvání datového přenosu bude pro komunikující koncová zařízení k dispozici konstantní šířka pásma, a že tok dat bude přenesen ve správném pořadí. Zprávy jsou typicky odesílány v nedělitelné formě, pokud se však rozloží, stále jsou všechny přenášeny po stejné trase.

Obrázek 1.1 ilustruje způsob, jakým funguje technologie přepínání okruhů na typickém příkladě použití ve veřejné telefonní síti. Jednotlivá koncová zařízení jsou připojena do sítě a v případě, že chce volající *User A* vytočit cílovou stanici *User X*, se v komutační síti sestaví okruh, který tyto dvě zařízení fyzicky propojí (červené spoje napříč jednotlivými přepínači okruhů).



Obr. 1.1: Princip přepojování okruhů

V oblasti datových sítí je zástupcem technologie ISDN (Integrated Services Digital Networks), což je soubor standardů pro přenos různorodých síťových služeb s použitím obvodů veřejné telefonní sítě v digitální formě[18].

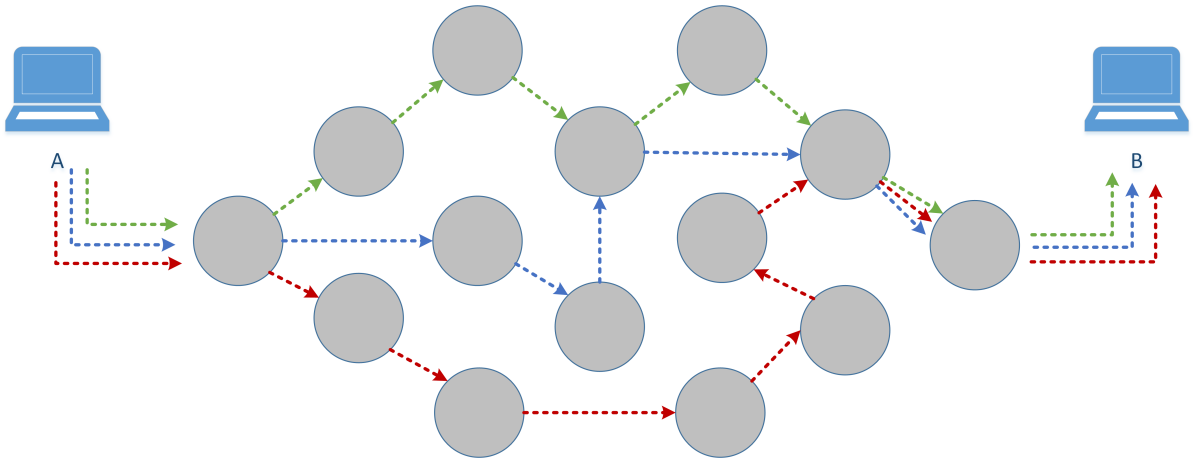
1.2 Přepojování paketů

Přepojování paketů je založeno na principu hledání nejvhodnější dostupné trasy s přihlédnutím na současné možnosti topologie sítě. Pakety jsou přenášeny od vysílajícího uzlu k cílovému uzlu tak, že každé přepojovací zařízení po cestě přeposílá paket dalšímu zařízení, které je podle něj v danou chvíli nejvýhodnější a probíhá nezávisle. Základní myšlenkou paketových sítí tedy je, že odeslání jednotlivých paketů probíhá bez předběžného zjišťování, zda cílový uzel stále existuje a nebo jestli k cíli dorazil.

Oproti komutačním sítím zde není žádná garantovaná cesta, po které bude komunikace probíhat. Výhodou u dynamického směrování paketů je, že pokud nějaký uzel po cestě vypadne, přepínací zařízení pošle paket k cíli alternativní cestou. S tím úzce souvisí skutečnost, že se jednotlivé pakety dostávají ke svému cíli v jiném pořadí, než byly původně odeslány, nebo se po cestě ztratí. Z toho důvodu je potřeba zavést mechanismy, které dokáží pakety seřadit do původního pořadí a zajistit znovuposlání všech paketů, které se do cíle nedostaly tak, aby bylo na cílovém zařízení možno znovusestavit data přenášená pakety.

Aby tyto mechanismy byly realizovatelné, musí každý paket obsahovat poměrně hodně informací potřebných k jeho správné manipulaci a nakonec rekonstrukci přenášené informace. Tato potřeba ústí ve výrazně vyšší režii, ale na druhou stranu nám paketové síťě dovolují větší volnost – nevyžadují stejnou přenosovou rychlost, stejný způsob přenosu a stejné kódování u obou účastníků komunikace. Zároveň také pakety cestují po kratších úsecích s nižší chybovostí, což síťovým zařízením dovoluje lépe využít přenosové pásmo[13]. Spolehlivost a lepší využití sítě je tedy vykoupeno vyšší režií a potencionálně pomalejším přenosem než v komutačních sítích (odesílá se celá zpráva, takže odpadává proces rekonstrukce a nehrozí ztráta nebo zpoždění dílčích zpráv)[18].

Obrázek 1.2 popisuje na příkladu poslání tří paketů techniku přepojování paketů. Uživatel *A* se rozhodne odeslat zprávu uživateli *B*. Protože je zpráva příliš velká, je rozdělena do tří paketů. Jednotlivé přepojovací zařízení (šedá kolečka) v paketové síti se pak chovají ke každému paketu nezávisle a provádějí přepojování podle aktuálních možností sítě, což v našem případě znamená, že sice každý paket dorazí do cíle k uživateli *X*, ale na rozdíl od přepojování okruhů dorazí každý z nich jinou cestou.



Obr. 1.2: Princip přepojování paketů

Používané technologie

Následující rozbor vychází ze studie [10]. Prvním zastupitelem paketových sítí je *X.25*, jehož nevýhodou je podporovaná rychlost přenosu pouze do 64 kbit/s a přílišné zatěžování uzlů kvůli spoustě funkcí na kontrolu a opravu chyb, které byly za svého času hojné vzhledem k nespolehlivosti sítí WAN. Dnes je X.25 nahrazen především technologií *Frame Relay*.

Frame Relay narozdíl od protokolu X.25 pracuje na spojové vrstvě, takže se zbavila nadbytečné režie, což vedlo ke zrychlení přenosu dat. Kontrolu a opravu chyb nechala Frame Relay na protokolech vyšších vrstev, takže není zajištěno, že se data dostanou až ke svému cíli. Další výhodou je rychlost v řádech jednotek Mbit/s.

Následovala *buňková komunikace*, která byla odpovědí na stále se zvyšující požadavky na kvalitu a šířku pásma. Buňka se od paketů a rámců liší tím, že má pevně danou délku a formát, což usnadňuje zpracování na hardwarové úrovni přepojovacích zařízení.

Další technologií v paketových sítích byl ATM (Asynchronous Transfer Mode), který se podobá telefonním sítím, protože přepojuje pakety za použití virtuálních okruhů. ATM se snaží vyřešit problém mezi paketovými a komutačními sítěmi tím, že převádí bitová i paketová data do jediného buňkového toku, které nesou označení virtuálního okruhu. ATM však nedokáže vyhovět stále rostoucím požadavkům na kvality služeb (QoS) a rychlost.

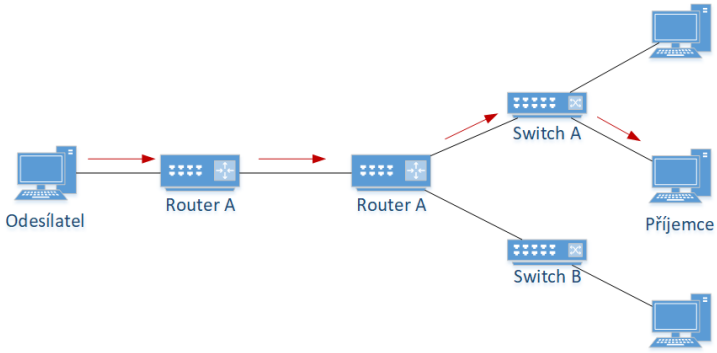
V současnosti tedy vítezí *Ethernet*, který realizuje fyzickou a linkovou vrstvu modelu ISO/OSI (3.2), což umožňuje provozovat více protokolů síťové vrstvy. Ethernet ve velkém nahrazuje ATM především pro zajímavé přenosové rychlosti a schopnosti vyhovět požadavkům na prioritizaci datových toků.

2 Typy přenosu

Při výměně informací po paketových sítích rozlišujeme mezi třemi typy přenosu podle toho, kolika cílovým zařízením je informaca posílána – neboli kolik má odeslaná zpráva příjemců.

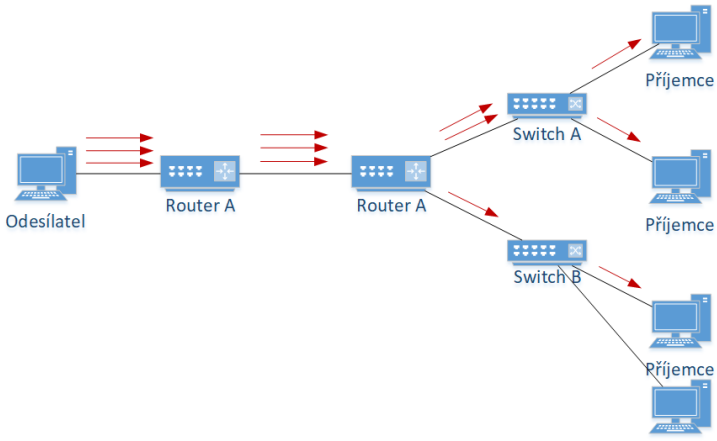
2.1 Jednosměrový přenos

První typem je jednosměrový přenos (též známý jako *unicast*), což znamená, že informace je odeslána jedním odesílatelem k jednomu příjemci (1:1), viz. obrázek 2.1.



Obr. 2.1: Princip jednosměrového přenosu

Tento typ přenosu by sice mohl být použitý i pro přenos k více příjemcům, ale byl by značně neefektivní (jak na straně přepojovacích zařízení, tak pro samotný vysílač), jelikož by musel zdroj odeslat data pro každého příjemce zvlášť, jak je ukázáno na obrázku 2.1. Unicast je pravděpodobně nejpoužívanějším typem přenosu, jelikož jej využívá většina dnešních internetových aplikací[4].

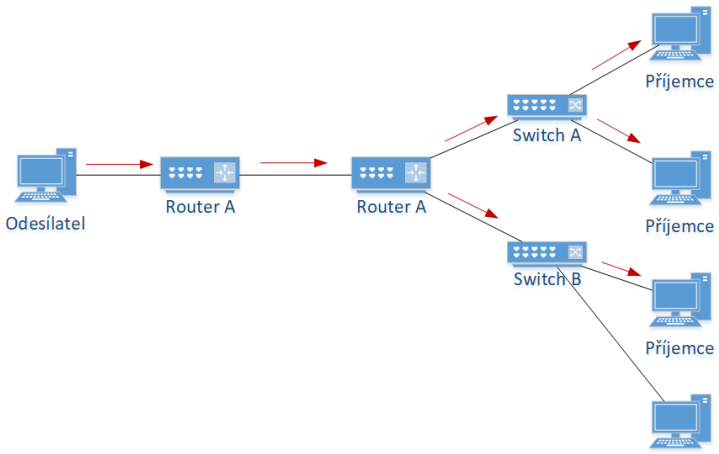


Obr. 2.2: Použití unicastového přenosu pro více příjemců

2.2 Vícesměrový přenos

Dalším typem je vícesměrový přenos (známý jako *multicast*), který se používá v případě, kdy chceme typicky informaci poslat skupině příjemců (1:`mnoho`). Tento model je výhodný pro přenos multimediálních dat, kde se uživatelé dle libosti přihlašují k odběru nějakého toku dat (např. streamovaná videa). Dále mohou nastat i případy, kdy se v komunikaci nachází i více zdrojů (`mnoho`:`mnoho`), kde narozdíl od předešlého typu může vysílat prakticky kdokoliv ze skupiny[4]. Zprvu se multicast používal v lokálních sítích, později však začal být podporován i v sítích rozlehlých.

Díky multicastovému přenosu se může zpráva snadno dostat k mnoha příjemcům a přitom nebýt odeslána na stejnou linku více než jednou, jak by to bylo s použitím unicastu. S použitím multicastu, odesílatel odešle jedinou kopii dat a přepojovací zařízení v síti pak duplikují tuto zprávu kdykoliv je potřeba, aby se zpráva dostala ke všem příjemcům z cílové skupiny, jak ukazuje obrázek 2.3[2].



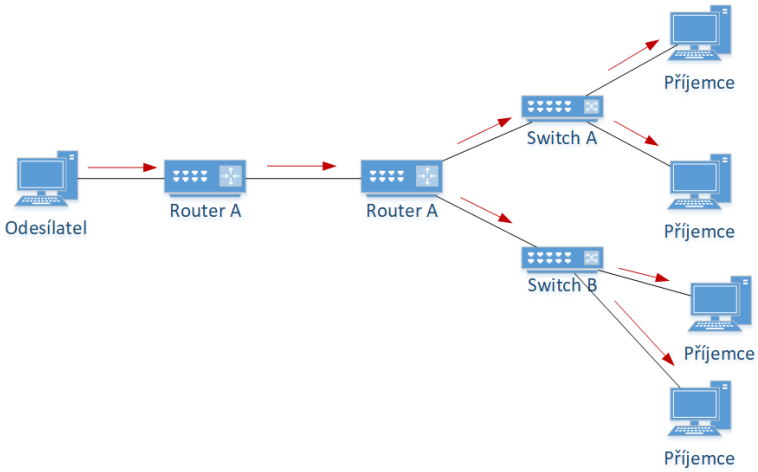
Obr. 2.3: Použití skupinového přenosu pro více příjemců

2.3 Všesměrový přenos

Všesměrový přenos, neboli *broadcast*, je typ přenosu, kdy je jeden zdroj dat a příjemci jsou všechna ostatní zařízení v lokální síti nebo podsíti (1:`všichni`). Používá se k tomu speciální všesměrová adresa, která se utvoří tak, že všechny bity vyjadřující adresu uzlu v IP adrese se nahradí binární jedničkou – což je vždy poslední adresa v daném rozsahu. Všechny stanice v síti pak obdrží tuto zprávu a musí ji zpracovat (viz. obrázek 2.4).

Hlavní výhodou všesměrového typu přenosu je šetření přenosového pásma a snadná dostupnost všech stanic v síti. Nevýhodou je naopak zbytečné zatěžování stanic,

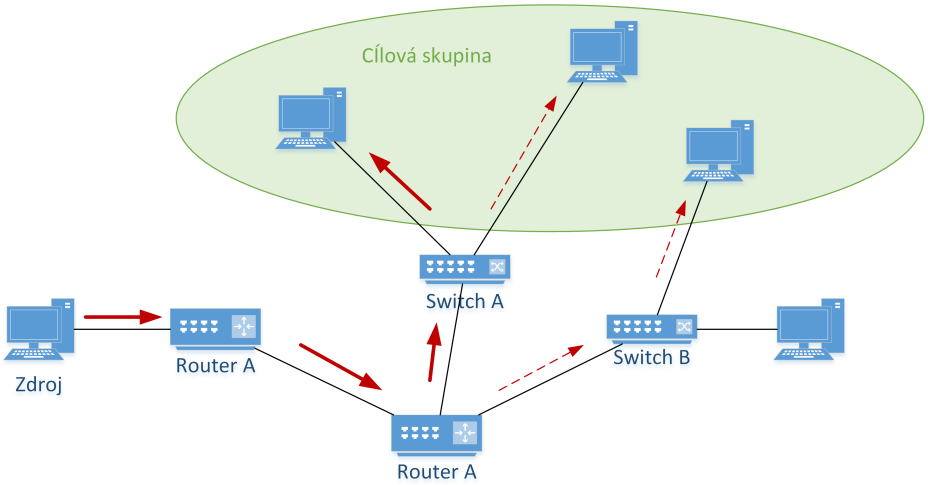
které zpráva nezajímá nebo se jich netýká. Aby se zahlcování nevyžádaným síťovým provozem omezilo, je vhodné sítě rozdělit na menší podsítě[4].



Obr. 2.4: Princip všesměrového přenosu

2.4 Výběrový přenos

Posledním typem přenosu je výběrový (známý jako *anycast*), který pracuje se skupinou příjemců, kteří vystupují pod stejnou IP adresou. Když odesílatel pošle zprávu k této skupině, síť se postará o to, aby se tato zpráva dostala k jednomu příjemci ze skupiny – typicky nejbližší nebo nejvhodnější stanice (1:`kdokoli`), jak je ukázáno na obrázku 2.5. Tento typ přenosu je vhodný k eliminaci redundantních zpráv (když příjemce vypadne ze sítě, zpráva je poslána jinému příjemci ze skupiny). Podpora anycastového přenosu může být v praxi složitá[2].



Obr. 2.5: Princip výběrového typu přenosu

3 Komunikační protokoly

Text v této kapitole se zabývá problematikou komunikačních protokolů s využitím v oblasti informatiky a internetu.

Komunikační protokol je soubor pravidel, které umožňují dvěma a více stanicím si mezi sebou vyměnovat informace. Komunikační protokol definuje syntaxi, sémantiku, způsob navazování a ukončování komunikace a způsoby zotavení z chybových nebo neočekávaných stavů[6]. Protokol musí být dohodnut všemi komunikujícími stanicemi a každá zpráva daného protokolu má exaktní význam. Protokol přesně definuje sled jednotlivých zpráv, takže vysílající stanice předem zná množinu možných odpovědí na odeslanou zprávu. Komunikační protokol obecně neřeší implementační detaily a tedy není svázaný s implementací.

Stejně jako máme mnoho způsobů komunikace v reálném světě, můžeme rozlišovat i internetové protokoly podle jejich použití. Nelze tedy říci, že všechny protokoly slouží pouze k odesílání dat, existují např. i protokoly k řízení datové komunikace, navazování zabezpečených kanálů a mnoho dalších[1].

Protokoly rozdělujeme na [6]:

- `Symetrické` - obě komunikující stanice mají stejná práva a funkci
- `Asymetrické` - komunikace klient-server, kdy jedna strana má vyšší pravomoci

3.1 Hierarchie protokolů

Počítačové komunikační protokoly se starají o realizaci dané komunikační funkce na všech úrovních. Pro zjednodušení je tedy většina sítí postavena na principu hierarchické posloupnosti vrstev. Tyto vrstvy i jejich počet je pro různé síťové modely jiný. Hlavním přínosem tohoto rozdělení je to, že každá vrstva modelu nabízí vyšší vrstvě určité služby, ale podrobnosti implementace těchto služeb jsou skryty[6]. Tato konzumace rozhraní služeb z nižších vrstvev je velice výhodná, jelikož se díky tomu daná vrstva může zabývat pouze implementaci vlastní aplikační logiky či protokolu a nemusí rozumět všemu, co se děje na nižších vrstvách.

Konverzace následně probíhá mezi jednotlivými síťovými prvky na úrovni stejných vrstev x. Pravidlům takové komunikace se říká protokol vrstvy x. Mezi každými dvěma vrstvami existuje rozhraní, které definuje, jaké operace a služby nabízí nižší

[1] Typickým příkladem by mohl být Real-time Transport Protocol (RTP), který standardizuje doručování zvukových a obrazových dat po internetu a na druhé straně jeho sesterský protokol RTP Control Protocol (RTCP), který poskytuje statistiky a řídící informace, ale sám nepřenáší žádná multimediální data[16].

vrstva vyšší. Souhrn vrstev a prokolů nazýváme *síťovou architekturou*. Soubor protokolů, kde se každý protokol stará o komunikaci na jedné vrstvě se nazývá *síťový zásobník*, o kterém pojednává sekce 3.4.

3.2 Rozdělení služeb

Jak už bylo naznačeno v předchozí kapitole 3.1, každá vrstva síťového modelu poskytuje vyšší vrstvě konkrétní služby. Implementace těchto služeb může být zabezpečena službami z bezprostředně nižší vrstvy. Služby jednotlivých vrstev můžeme dále dělit podle společných kritérií.

Jedním z možných dělení je na základě toho, zda je služba *spojovaná* nebo *nespojovaná*. U spojovaných služeb je před zahájením komunikace navíc potřeba navázat spojení. Při navazování spojení je nutné, aby protistrana se spojením souhlasila. Toto spojení je opět rozvázáno po skončení komunikace. Opakem jsou nespojované služby, kdy se žádné spojení nenavazuje a komunikace se zahájí okamžitě. U nespojovaných služeb se nezjišťuje ani to, zda protistrana se zasíláním dat souhlasí[6].

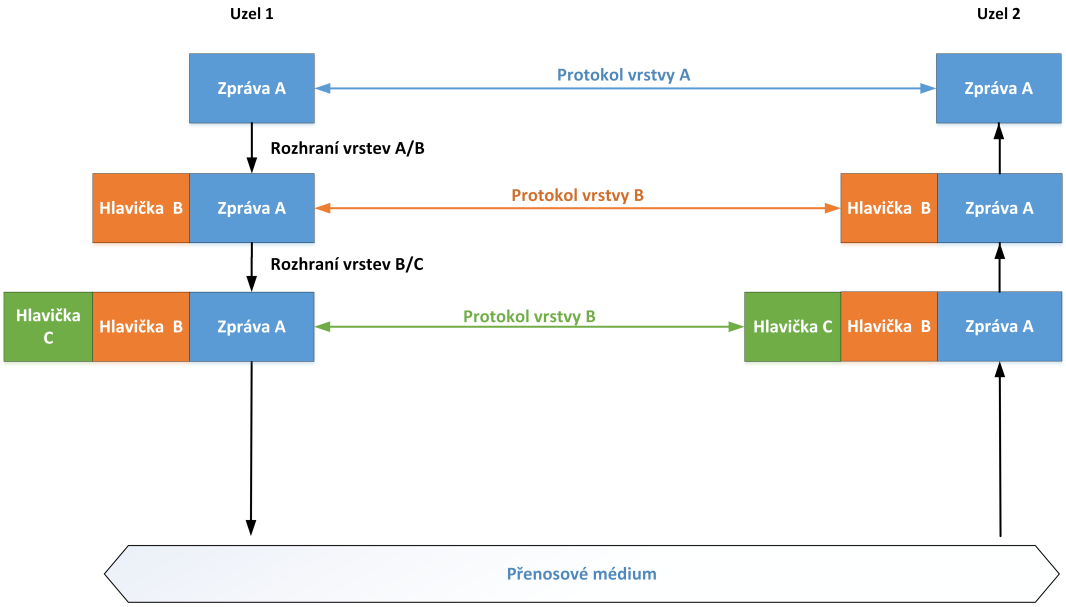
Další typické kritérium pro dělení služeb je jejich spolehlivost. *Spolehlivé* služby se vyznačují tím, že při přenosu neztratí žádná data. Toho je docíleno tak, že příjemce odesílateli potvrzuje každou přijatou zprávu. Spolehlivé služby jsou používány pro přenos zpráv, u kterých by chybějící bloky dat kriticky omezovaly účel přenosu, například při přenosu souborů. S potvrzováním zpráv však přichází i režie potřebná k udržování této synchronizace a tedy i zpoždění. Zpoždění způsobené potvrzováním zpráv může být přirozeně pro některé služby nežádoucí, z toho důvodu existují i služby *nespolehlivé* (typicky aplikace založené na real-time multimediálním přenosu, kdy je k příjemci soustavně posílán velký objem dat, jehož potvrzování by mohlo způsobit např. sledování již neaktuálního dění). U nespolehlivých služeb typicky není vyžadován precizní přenos všech dat a aplikace používající tyto služby dokáží pracovat i s místy nekompletními daty a docílit uspokojivého uživatelského zážitku[2].

Výše popsaná rozdělení – spojovaná/nespojovaná a spolehlivá/nespolehlivá – na sobě nejsou nijak závislá, vyskytují se všechny jejich vzájemné kombinace[6].

3.3 Zapouzdřování dat

Jak už bylo naznačeno, jednotlivé vrstvy předávají svá data dalším vrstvám, které jsou bezprostředně pod nimi. Aby předávání dat mělo nějaký smysluplný řád, byl zaveden pojem zapouzdřování, který je naznačen na obrázku 3.1. Pojem zapouzdřování

v počítačových sítích znamená připojování metadat (data popisující obsah odesílaných dat a jejich správné použití a význam). Každá vrstva síťového modelu musí před odesláním dat připojit k existující zprávě svá metadata, kterými se pak bude řídit protivrstva na straně příjemce po „odpouzdření". Mechanismus zapouzdřování zabezpečuje modulární přístup při navrhování komunikačních protokolů. Použitím abstrakce nám totiž dovoluje se při popisu jednotlivé služby dané vrstvy logicky odprostit od struktur používaných jejich bezprostředně nižšími vrstvami[18].



Obr. 3.1: Ilustrace principu zapouzdřování.

Zapouzdřování probíhá tak, že vrstva A vyprodukuje nějaká data v podobě zprávy A. Tato zpráva je následně v souladu s pravidly služeb na rozhraní vrstev A/B předána bezprostředně nižší vrstvě, tedy vrstvě B. Vrstva B převzatou zprávu obohatí o řídící a kontrolní data[1] ve formě hlavičky B. Tímto je původní zpráva A transformována na novou zprávu, která je opět předána v souladu s pravidly rozhraní vrstev B/C. Výsledná zpráva je konečně předána fyzickému přenosovému médiu. Tyto nově přidané informace jsou důležité při zpětné transformaci několikrát přebalené zprávy na původní zprávu A. Zpětná transformace probíhá na cílovém zařízení[4]. Každá vrstva se v celém procesu zapouzdření stará pouze o svoji část zprávy, které rozumí.

Zároveň je potřeba, aby dokázala každá z vrstev identifikovat, že je přijatá zpráva určena právě tomuto uzlu/procesu (zde záleží na kompetenci dané vrstvy), jelikož se síť skládá typicky z více než pouze dvou uzlů.

[1]Řídící a kontrolní data jsou v tomto případě informace specifické pro daný protokol a jsou mimo jiné nositelem informace o uskutečněné transformaci zprávy A.

3.4 Síťový zásobník

Tato podkapitola se bude zabývat popisem dvou nejdůležitějších síťových modelů dnešní doby – modelů ISO/OSI a TCP/IP. Každý model definuje rozdělení typů služeb, síťových zařízení a softwaru do souboru vrstev.
Síťový zásobník (nebo anglicky *protocol stack*) je model a architektura, které popisují síťové transakce mezi dvěma síťovými prvky. Důvodem existence je standardizace zařízení a služeb v síti. Takto odsouhlasené standardy umožňují komunikaci mezi síťovými prvky pracujícími na různých vrstvách v síti[18].

3.4.1 Model ISO/OSI

Referenční model OSI (Open System Interconnection Reference Model) popisuje vrstvy a jejich funkce, nezabývá se konkrétně žádnými protokoly ani implementacemi služeb. Slouží tedy pouze jako základna pro normy. Ačkoliv je model OSI nejčastěji používán k popisu síťových technologií, jen velmi zřídka bychom narazili na síť realizovanou na těchto sedmi vrstvách. V současném světě jeho vliv ustupuje, jelikož byl potlačen modelem TCP/IP[18].

Obr. 3.2: Open System Interconnection Reference Model

Obrázek 3.2 uvádí všech sedm vrstev modelu OSI a jejich hierarchické uspořádání. První čtyři vrstvy se vztahují k hardwaru, zbylé tři jsou naopak doménou softwaru. Následující popis jednotlivých vrstev vychází ze zdrojů [18] a [4]:

Fyzická vrstva

Fyzická vrstva je nejnižší úrovní modelu OSI a je odpovědná za přenos bitů dat mezi jednotlivými síťovými zařízeními. Definuje přenosové médium a způsob jakým se má užívat (např. optická vlákna, rádiové vlny, ..).

Linková vrstva

Úkolem linkové vrstvy je detekce chyb vzniklých na fyzické vrstvě a postarání se o adresaci hardwaru – spojení mezi sousedícími prvky sítě v kontextu síťové trasy.

Data z fyzické vrstvy jsou přenášena v celcích, které nazýváme `rámce` (pokud je zpráva příliš velká, data jsou rozdělena na segmenty o typické délce stovek až tisíců bajtů). Mimo obsah zprávy je rámec obohacen o fyzické adresy vysílajícího a přijímacího systému. Na linkové vrstvě pracuje důležitý síťový prvek *přepínač*.

Síťová vrstva

Síťová vrstva se stará o adresaci přímo nesousedících komunikujících systémů. Manipulovanou datovou jednotkou je zde paket. Nabízí metody řízení a směrování za účelem určení trasy, po které budou přenášeny datové pakety mezi jednotlivými sítěmi. Mimo jiné se stará i o prevenci zahlcení podsítě pakety.

V přepínaných sítích hraje směrování klíčovou roli, jelikož se díky němu dokáže trasa přizpůsobit dynamicky se měnící topologii. Mapu spojení si směrovače uchovávají ve směrovacích tabulkách.

Transportní vrstva

Transportní vrstva poskytuje abstrakci pro zbylé horní vrstvy modelu, které jsou na rozdíl od spodních hardwarových vrstev, typicky softwarové a provádí správu a řízení multiplexních datových spojení na základě současných potřeb. Transportní vrstva se stará o rozčlenění dat souvisejících s nějakou relací a předání ve správné velikosti a formátu vrstvě síťové. Naopak při vstupu dat ze síťové vrstvy do transportní, je potom její prací zajistit správné seřazení přijatých paketů, rekonstrukci relační informace a potvrzení přijetí.

Na úrovni transportní vrstvy rozlišujeme spojovanou a nespojovanou komunikaci.

Relační vrstva

Relační vrstva disponuje především bezpečnostními mechanismy (přihlašování/odhlašování) a funkcemi potřebnými k vytváření a udržování relací. K datům v paketech se přidávají oddělovače nebo kontrolní body, díky kterým jsme schopni obnovit přerušenou relaci, aniž bychom museli znovu přeposílat všechna předchozí data.

Provoz na relační vrstvě může být jednosměrný (half-duplex) nebo obousměrný (full-duplex). V jednosměrném režimu se předává identifikátor nazývaný token – pouze strana vlastnící token může v daný moment vysílat data.

Prezentační vrstva

Úroveň prezentační vrstvy poskytuje formátování a případnou kompresi/šifrování dat z aplikační vrstvy. Na opačné straně zase prezentační vrstva, pokud je to nutné, dekomprimuje či dešifruje data příchozí z relační vrstvy, aby byla srozumitelná pro aplikaci v aplikační vrstvě. Používají se zde protokoly, pomocí kterých lze překlenout rozdíly mezi aplikacemi či operačními systémy, což umožňuje komunikaci počítačům s různými znakovými sadami.

Aplikační vrstva

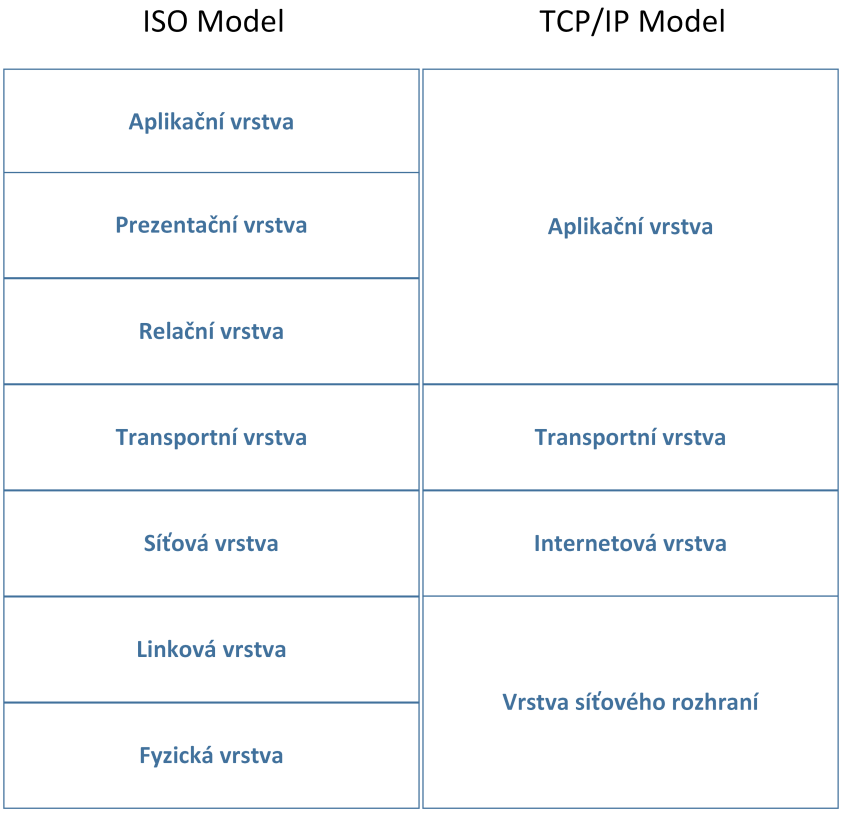
Aplikační vrstva zajišťuje síťové spojení mezi aplikací a sítí a hostuje software, se kterým už přímo komunikuje koncový uživatel. Nejrozsáhlejší sadu síťových protokolů nacházíme právě na aplikační vrstvě, mezi typické zástupce řadíme protokol HTTP (Hypertext Transfer Protocol), který používají webové prohlížeče, dále pak třeba FTP (File Transfer Protocol), který slouží k nahrávání a stahování souborů.

3.4.2 Model TCP/IP

Největším konkurentem referenčního modelu ISO/OSI je soustava protokolů TCP/IP. Pojem TCP/IP nezahrnuje pouze protokoly TCP (Transmission Control Protocol) a IP (Internet Protocol), nýbrž celou sadu protokolů spolu s názory o fungování počítačových sítí – přináší totiž i představu toho, jak by mělo vypadat členění síťového vybavení do jednotlivých vrstev, spolu se způsoby plnění vyplývajících úkolů[4].

TCP/IP se od ISO/OSI liší hlavně tím, že na rozdíl od modelu OSI tvůrci předpokládali, že zajištění spolehlivosti je problémem až koncových uživatelů a měla by tedy být řešena až na úrovni transportní vrstvy. Tím se eliminuje ztráta přenosové kapacity komunikační sítě způsobená režií vynaloženou na udržení spolehlivosti a místo toho může být tato energie věnována vlastnímu přenosu dat. TCP/IP tedy předpokládá, že se budou hostitelské počítače připojovat do jednoduché a rychlé posdítě.

Zatímco model OSI se těší hojnému uplatnění v teoretických diskuzích, TCP/IP je zase často aplikován na skutečných projektech. Na rozdíl od mnohavrstvého modelu OSI předpokládá TCP/IP nespojovaný přenos v komunikační vrstvě s pouze čtyřmi vrstvami[6]. Obrázek 3.3 toto srovnání ilustruje, popis jednotlivých vrstev vychází z textu [4].



Obr. 3.3: Porovnání referenčního modelu ISO/OSI s TCP/IP.

Vrstva síťového rozhraní

Vrstva síťového rozhraní je nejnižší vrstvou TCP/IP architektury, která zahrnuje fyzickou i linkovou vrstvu referenčního modelu OSI. Stará se o přímé vysílání a přijímání dat a ovládání konkrétní přenosové trasy. Bližší specifikace se v jednotlivých případech liší použitou přenosovou technologií.

Internetová

Internetová vrstva odpovídá síťové vrstvě modelu ISO/OSI a již není závislá na přenosové technologii. Odpovědností internetové vrstvy je přenést pakety od odesílatele až ke svému skutečnému adresátovi, k čemuž se typicky využívají směrovače. Na

této úrovni je z důvodu nespojovaných přenosů zajišťována nespolehlivá datagramová služba. Jednotlivé sítě po cestě od odesílatele k příjemci mohou být navíc v některých věcech odlišné (např. charakter adres, maximální velikost paketů, atd.), vyrovnání se s těmito skutečnostmi je také odpovědností internetové vrstvy.

Transportní

Transportní vrstva je často označována jako TCP vrstva, protože je její realizací nejčastěji protokol TCP. Úkolem vrstvy je přenášet data mezi dvěma koncovými aplikacemi. Mimo přenosu je potom důležitá pro regulaci toku dat, zajištění spolehlivosti a měnit nespojovaný přenos v síťové vrstvě na spojovaný.

Alternativou k protokolu TCP je typicky protokol UDP (User Datagram Protocol). UDP je charakteristický tím, že se nestará o spolehlivost přenosu. Protokol bez garance spolehlivosti je využíván např. aplikačním protokolem SNMP[2].

Aplikační

Entitami aplikační vrstvy jsou jednotlivé uživatelské aplikace komunikující přímo s transportní vrstvou. Oproti referenčnímu modelu ISO/OSI si aplikace v TCP/IP musí zajistit případné prezentační a relační služby samy. Pokud aplikace nevyžaduje služby relační nebo prezentační vrstvy, zaniká nutnost jejich implementace.

[2]SNMP (Simple Network Management Protocol) je vysoce používaný protokol pro monitorování stavu a správu síťových zařízení, jako jsou směrovače, na IP sítích[11].

4 Řízení toku dat

Při návrhu datových sítí je potřeba vyvinout značné úsilí na zajištění dostatečné propustnosti sítě. Jedním z hlavních rizik těchto sítí je totiž možné zahlcení jednotlivých síťových zařízení a schopnost zotavení z chybových stavů. *Zahlcení* je stav, kdy určitý síťový prvek nestíhá včas odbavovat přicházející pakety, což vede k tomu, že mu neustále narůstá fronta požadavků na zpracování až do chvíle, kdy má zaplněnou všechnu vyrovnávací paměť[1]. V tu chvíli existují dva způsoby jak se se zahlcením vyrovnat[7]:

- Nově příchozí pakety jsou zahozeny – systémy zahazování paketů jsou důležité pro udržení efektivního využití přenosového pásma v případě, že dochází k většímu zatížení sítě, nebo je už zahlcená. Za těchto okolností musejí postižené síťové prvky zahazovat pakety, přičemž způsobů zahazování existuje hned několik. Pokud není použitý žádný speciální algoritmus na zahazování paketů, předpokládá se, že jsou zahazovány nově příchozí pakety[9].
- Zahlcené zařízení začne regulovat příchozí objem dat tím, že požádá sousední zařízení o zpomalení vysílaní. To však může mít za důsledek přeplňování front v předchozích přepojovacích zařízení. Postupem času byly vymyšleny algoritmy, jejichž implementací jsme schopni předvídat možné blížící se zahlcení a vykonat tak určité kroky, které jim pomohou tomuto stavu předejít[7] (např. Zpětný tlak, Tlumící paket, Implicitní a explicitní signalizace zahlcení).

4.1 Řešení chybových stavů

Během přenosu dat přirozeně čas od času dochází k poškození nebo ztrátě přenášené informace, ať už kvůli možným výpadkům v síti nebo z důvodu dříve uvedeného zahlcení sítě, při kterém jsou pakety systematicky zahazovány. K vypořádání se s těmito problémy jsou zavedeny techniky, které řídí komunikace a reakce na vzniklé chybové situace.

K samotnému zjištění, že stanice přijala poškozený paket, slouží tzv. kontrolní součet FCS (Frame Check Sequence), jehož hodnota je součástí přenosového rámce. FCS se na straně vysílače spočítá pomocí CRC (Cyclical Redu-nancy Check), což je kód, který se používá k detekci chyb. Poté se na straně příjemce z přijatého

[1] Přepínače a směrovače mají buď společnou paměť pro všechny porty, nebo může mít každý port svoji vlastní. Do této paměti jsou ukládány příchozí datové jednotky (kde čekají na zpracování) a pakety připravené k odeslání na další přepojovací/koncové zařízení. Velikost paměti je v daném zařízení neměnná[7].

rámce vypočítá nová hodnota FCS, která se porovná s původní FCS z daného rámce a pokud se hodnoty liší, předpokládá se, že byl přijatý rámec poškozen.[5].

ARQ

[8]Problémy poškozených a ztracených rámců řeší ARQ (Automatic Repeat reQuest) protokoly. ARQ zavádí principy potvrzování přijatých dat. V případě, že vysílač neobdrží potvrzení na zaslaný rámec po dobu stanoveného časového úseku, vyšle tento rámec znovu, přičemž má vysílač předdefinováno, kolikrát se nepotvrzený rámec může pokusit znovu odeslat. ARQ protokoly slouží i k řešení situace, kdy je doručen poškozený rámce. V tomto případě příjemce zareaguje odesláním rámce negativního potvrzení a očekává, že mu vysílač odešle poškozený rámec znovu. Na úrovni spojové vrstvy máme tři ARQ protokoly, kterými se typicky řeší zmíněné chybové situace: Stop-and-wait, Go-Back-N, Selective-repeat.

4.1.1 Stop-and-Wait

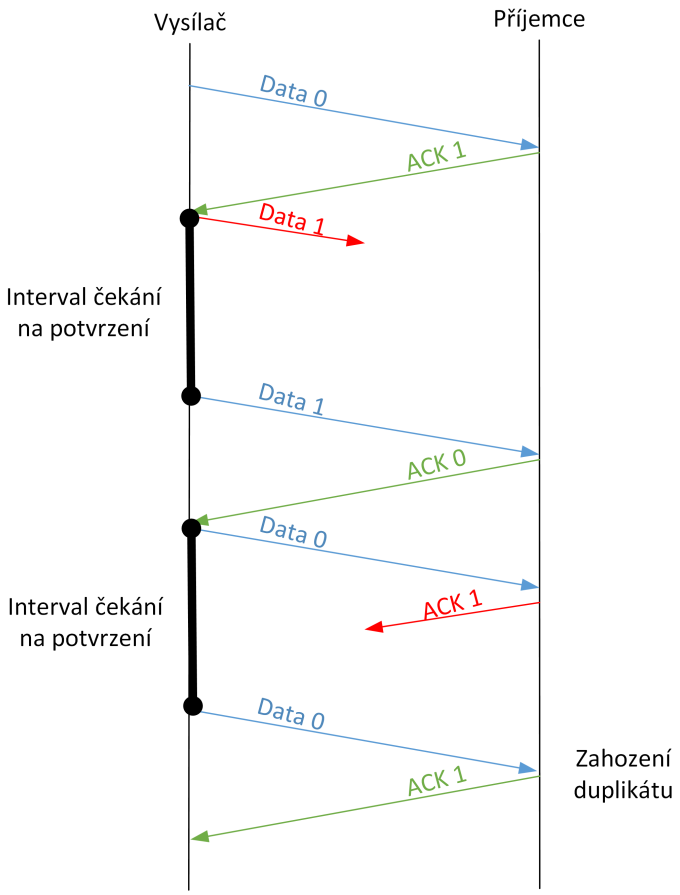
Nejjednodušší realizací je protokol stop-and-wait, jehož popis vychází z textů [7] a [8]. Vysílací strana odešle rámec obsahující kontrolní součet a čeká na potvrzení, že byl přijat. Pokud je rámec doručen v pořádku až k příjemci, vyšle odesílateli zprávu potvrzující bezchybné obdržení rámce, čímž přijímací strana zároveň deklaruje, že je připravena na obdržení další zprávy. Jakmile vysílač obdrží potvrzení, pokračuje stejným způsobem odesláním dalšího rámce. Kromě schopnosti vypořádat se s poškozenými či ztracenými rámci může přijímač do jisté míry regulovat rychlost zasílání dat jednoduše tím, že vyčká s odesláním potvrzení.

Mimo standardní chybové případy s poškozením nebo ztracením vysílaného rámce, které jsou řešeny způsoby uvedenými v předchozí sekci **ARQ**, může nastat situace, kdy dojde ke ztrátě nebo poškození potvrzovacího rámce. V takovém případě vysílač čeká do vypršení stanoveného časového úseku a opakuje odeslání původního rámce, jako kdyby došlo k jeho ztrátě. Přijímač tedy znovu obdrží stejný rámec a aby se zabránilo opakovanému zpracování již zpracovaného rámce, jsou datové i potvrzující rámce číslovány jako 0 a 1. Zpráva ACK1 znamená, že přijímač obdržel rámec s označním 0 a naopak.

Příklad komunikace s využitím protokolu stop-and-wait je znázorněn na obrázku 4.1. Každá hrana mezi časovými osami komunikujících stran vyjadřuje odeslání jednoho rámce. První dvojice zpráv mezi vysílačem a příjemcem ilustrují bezchybnou komunikaci na obou stranách. Třetí rámec byl ztracen, takže vysílač musel počkat do vypršení časovače a následně poslal rámec znovu, tentokrát už úspěšně.

Později je na obrázku popsán i případ, kdy se ztratí potvrzení o přijetí *ACK1*. Vysílač znovu čeká na skončení časového intervalu, po jehož dobu čeká na potvrzení,

a po té znovu odešle rámec *Data 0*. Příjemce detekuje duplicitní rámec, který zahodí, a pošle zpět potrvzení o přijetí.



Obr. 4.1: Ukázka fungování protokolu stop-and-wait a jeho schopností vypořádat se se ztracenými datovými rámci odesílatele i potvrzovacími rámci příjemce.

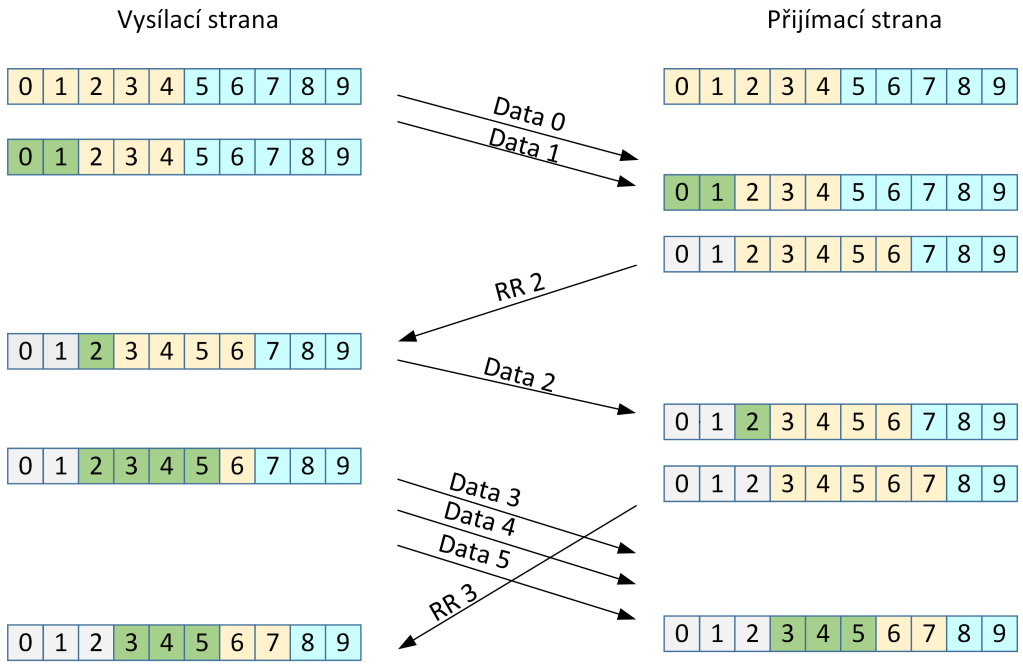
4.1.2 Posuvné okno

Slabina stop-and-wait mechanismu je, že dokáže najednou přenášet pouze jeden rámec, což velice omezuje efektivitu přenosu. Logickým krokem při řešení tohoto problému by bylo odesílat více rámců bez nutnosti čekání na potvrzení mezi nimy. S touto modifikací vzniká potřeba jednotlivé rámce číslovat, aby vysílací stanice měla přehled o potvrzených rámcích.

V praxi to funguje tak, že přijímací stanice prohlásí, že je schopna najednou přijmout x rámců[7]. V tu chvíli může vysílací stanice odeslat x rámců, přičemž příjímač odesílá potvrzení obsahující číslo dalšího očekávaného rámce (podobně jak u stop-and-wait). Jakmile vysílač obdrží potvrzení s číslem očekávaného rámce a, je to pro něj signálem, že stanice na druhé straně může přijmout dalších x rámců, počínaje rámcem s číslem a. Aby tento mechanismus fungoval, je potřeba, aby si obě strany

udržovaly v paměti rámce, které mohou v daný okamžik odeslat (vysílač) a příjmout (přijímač). Z uvedeného popisu vyplývá, že s využitím techniky posuvného okna může přijímač přijmout např. rámce *a, b, c, d* s tím, že potvrzení odešle až po přijetí rámce posledního rámce *d*, čímž značně zefektivní průtok dat sítí.

Obrázek 4.2 ilustruje tento mechanismus na příkladu s maximální velikostí okna 5. Žluté čtverečky symbolizují rámce, které jsou aktuálně uvnitř okna, zelené symbolizují odeslané/přijaté rámce, šedé jsou potvrzené a modré jsou rámce, které budou postupně naskakovat do okna, jakmile se uvolní místo. Zpráva RR (Receive Ready) je významově podobná zprávám ACK u algoritmu stop-and-wait.



Obr. 4.2: Mechanismus posuvného okna s velikostí okna 5, přičemž odesílatel chce odeslat 9 rámců.

4.1.3 Go-Back-N

Go-Back-N (GBN) je jednodušším typem implementace mechanismu posuvného okna. K potvrzování přijatých rámců se typicky používá zpráva RR (Receive Ready). V případě, že přijímací strana obdrží poškozený rámec, odešle vysílači záporné potvrzení – typicky se jedná o zprávu REJ (Reject) nebo NAK (Negative ACK) – a zahodí tento i další po něm následující pakety bez ohledu na to, jestli byly přijaté bezchybně. Z toho důvodu je vysílač po přijetí záporného potvrzení nucen odeslat znovu jak poškozený rámec, tak i všechny po něm následující. Jakmile přijímač obdrží neporušenou verzi dříve poškozeného rámce, přestane zahazovat rámce a pokračuje v klasickém chování.

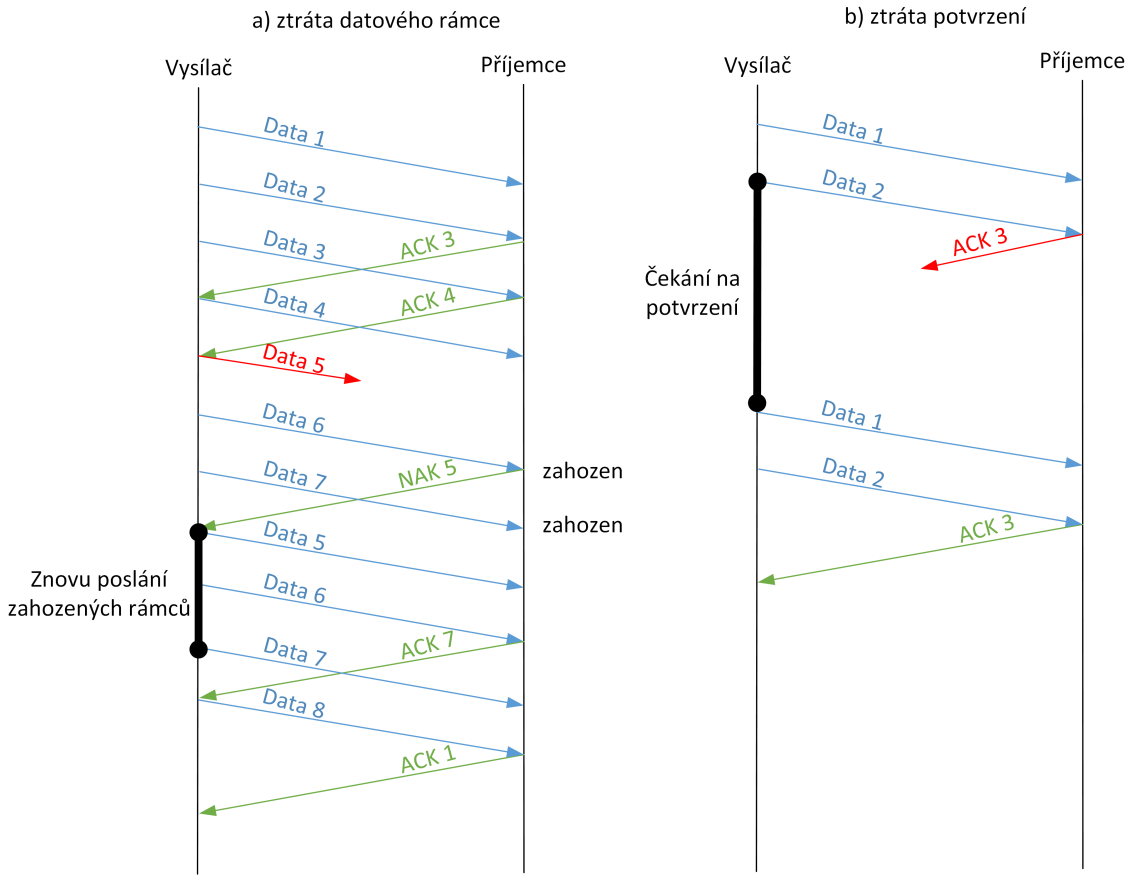
V případě hustého výskytu chyb se nejedná o optimální řešení, jelikož dochází k masivnímu plýtvání přenosového pásma. Správně nastavený protokol GBN (včetně časovačů) nicméně zvládne vyřešit následující chybové stavy (dále uvažujeme, že stanice A je vysílač a strana B je přijímač)[8][19]:

1. **Ztracení odeslaného rámce**
   - stanice A odesílá za sebou rámce 1 a 2. První z rámců, 1, byl po cestě ztracen, a tak doputoval do cíle pouze rámec 2. Stanice B nyní přijala rámec 2, ale jelikož očekává rámec číslo 1, tak jej zahodí a pošle stanici A zprávu *NAK 1* (stanice B zahazuje další rámce do té doby, dokud neobdrží chybějící rámec). Stanice A obdrží zprávu *NAK 1* a zjistí tak, že se rámec číslo 1 po cestě ztratil, tak jej odešle znovu (opakovaně odeslaný rámec 1 je následován rámcem 2, jelikož stanice A ví, že byl zahozen).
   - pokud by byl stanicí A odeslán pouze rámec 1 bez následníka, stanice B by neměla jak zjistit, že došlo ke ztrátě rámce. V takovém případě stanici A pomůže časový interval, po jehož dobu stanice čeká na potvrzení od stanice B. Pokud po skončení intervalu potvrzení nedorazí, stanice A pošle rámec 1 znovu.
2. **Poškození odeslaného rámce** - pokud stanice B obdrží poškozený rámec, chová se velmi podobně jako při ztrátě očekávaného rámce – stanici A je odeslána zpráva NAK s číslem rámce, který je očekáván. Následující rámce (kromě očekávaného) jsou zahazovány a nepotvrzují se.
3. **Ztráta potvrzení** - pokud přenos rámců ze stanice A do stanice B proběhne v pořádku, ale potvrzení je poškozeno nebo bylo ztraceno po cestě, stanice B čeká do uplynutí časového intervalu, po jehož dobu očekává, že dojde potvrzení. Jakmile překročí zmíněný práh časovače, odešle znovu všechny rámce, pro které neobdržel potvrzení. Podobně by se řešilo chování při ztrátě zprávy negativního potvrzení.

Obrázek 4.3 popisuje schopnost algoritmu GBN vypořádat se s chybovými stavy. V levém diagramu je popsán případ, kdy dojde ke ztrátě datového rámce při přenosu k příjemci. Ve chvíli kdy přijímač zjistí, že místo očekávaného rámce 5 dorazil rámec 6, odešle vysílači zprávu NAK 5, a zahazuje další přijaté rámce. Vysílač obdrží tuto zprávu a zareaguje opětovným vysláním ztraceného rámce 5 a všech následujících rámců.

Na pravé straně je popsán naopak stav, kdy dojde ke ztrátě potvrzovací zprávy. Ve scénáři se předpokládá, že chce vysílací stanice odeslat pouze dva datové rámce. Ty jsou k příjemci přeneseny bezchybně, avšak potvrzovací zpráva je po cestě ztracena. V tomto případě se vysílač bez potvrzení nemá jak dozvědět, že data doputovala k příjemci, a tak je odkázán na čekání. Jakmile je překročen práh intervalu, po

jehož dobu se čeká na potvrzení, vysílač opakovaně odešle oba datové rámce a znovu čeká na potvrzení. To bylo nyní bezchybně doručeno vysílači.



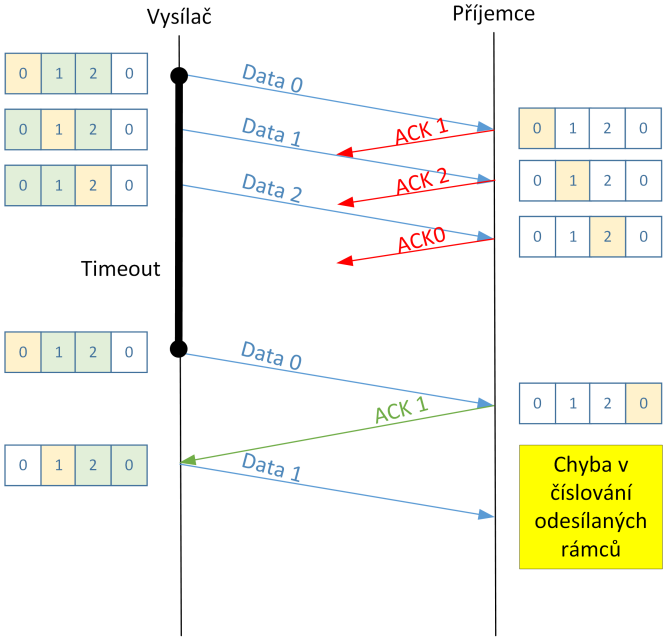
Obr. 4.3: Ukázka schnoposti algoritmu Go-Back-N zotavit se ze stavu ztraceného datového rámce v levé části a ze ztráty potvrzovacího rámce v pravé části obrázku.

Velikost okna u Go-back-N

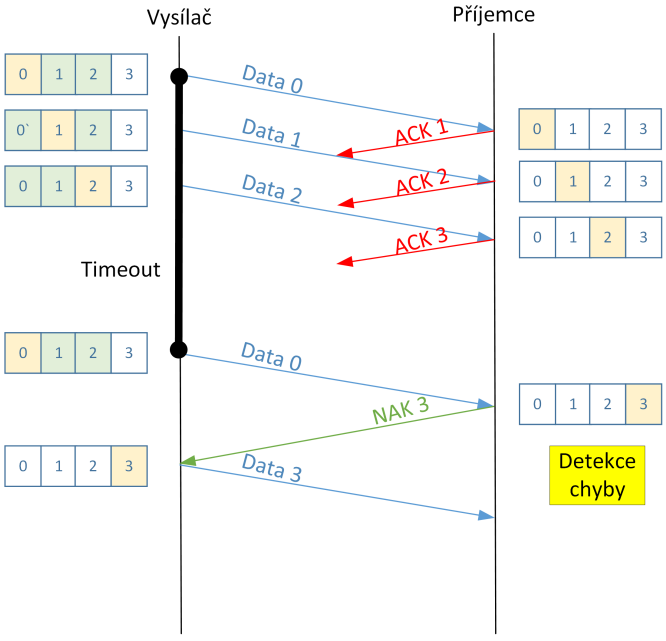
Pro správné fungování protokolu GBN je potřeba zvolit vhodnou velikosti posuvného okna. Okno příjemce má velikost W=1, jelikož GBN nepřipouští přijem jakéhokoliv rámce mimo očekávé pořadí. Na straně vysílače je situace složitější, protože by nevhodně nastavená velikost mohla způsobit nesprávné chování protokolu[1].

Situace je vysvětlena na obrázku 4.4, kde je v horní polovině obrázku ukázána situace pro stejnou velikost okna, jako je počet sekvenčních čísel (W=3). Vysílač odesílá tři rámce, které přijímač v pořádku obdrží, ale potvrzení jsou po cestě ztracena. Po skončení časovače vysílač opětovně pošle první rámec. Příjemce očekává rámec se stejným sekvenčním číslem, a proto nezjistí, že daný rámec už jednou přijal a chová se, jako by se nic nestalo. Nedošlo tedy ke správné reakci na chybovou situaci. Ve spodní polovině obrázku je situace jiná, jelikož velikost okna je menší

než počet sekvenčních čísel, takže příjemce v podobné situaci zjistí, že došlo ke ztrátě potvrzení a zareaguje příslušnou zprávou *NAK 3*. Synchronizace stran tedy není narušena.



a) W = počet sekvenčních čísel



b) W < počet sekvenčních čísel

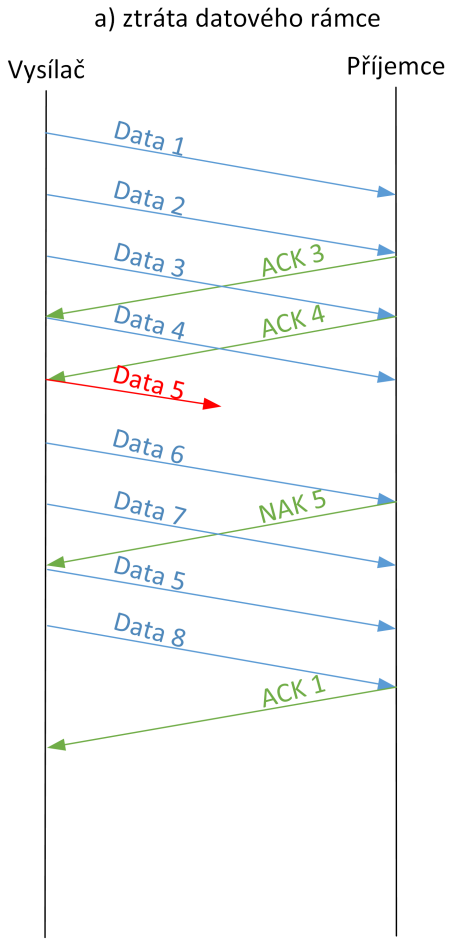
Obr. 4.4: Go-back-N při velikostech posuvného okna: a) velikost okna je stejná jako počet sekvenčních čísel, b) velikost okna je menší než počet sekvenčních čísel.

4.1.4 Selective-repeat

Protokol Selective-repeat (SR) je alternativní implementací posuvného okna. Na rozdíl od GBN se opakovaně posílá pouze rámec, jehož potvrzení nedošlo v očekávaném časovém intervalu, nebo byl odmítnut zprávou NAK. Výhodou mechanismu SR je, že eliminuje opětovné posílání rámců, které jinak doputovaly k cíli bezchybně, akorát ve špatném pořadí. Na druhou stranu je řízení složitější a celkově náročnější na paměť, jelikož si příjemce musí uchovávat neseřazené zprávy před samotným předáním kompletní informace vyšší vrstvě [7][19].

Obrázek 4.5 ukazuje způsob vypořádání se ze situace, kdy byl po cestě ztracen datový rámec. Přijímač upozorní zprávou *NAK 5* vysílací stanici, že došlo ke ztrátě rámce a dokud není rámec 5 v pořádku doručen k příjemci, neposílá přijímací stanice žádné potvrzovací zprávy.

V případě ztráty potvrzení se chování nemění, vysílací stanice za použití SR protokolu opakovaně odešle po skončení časovače všechny nepotvrzené zprávy.

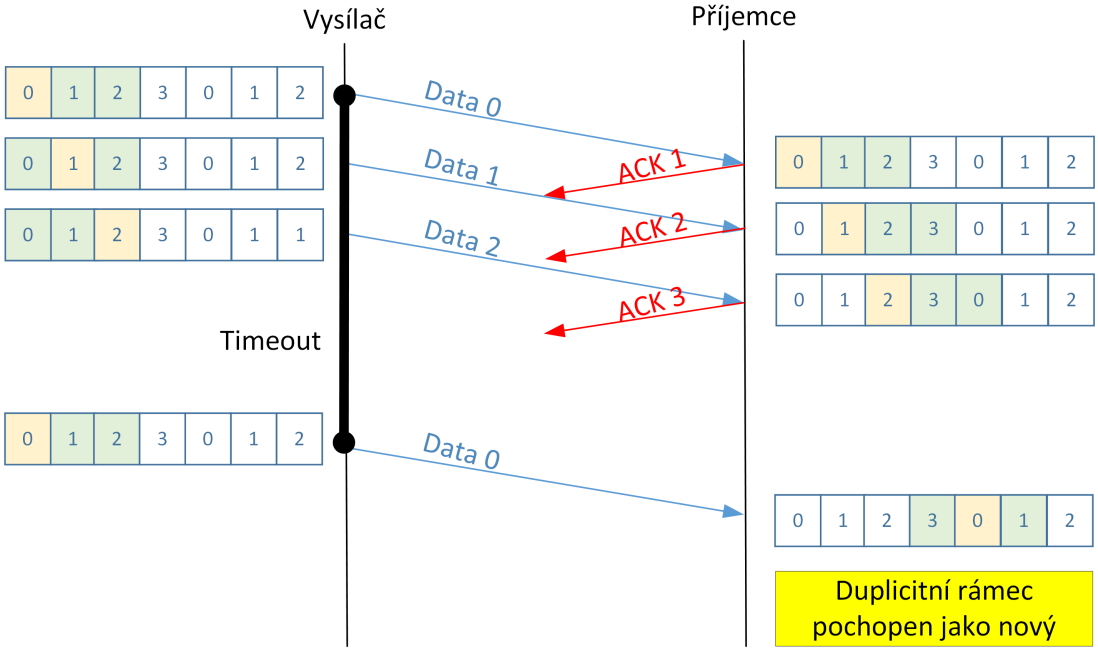


Obr. 4.5: Způsob reakce protokolu Selective-repeat na situaci, kdy se ztratí odeslaný rámec. Odesílatel po upozornění opětovně odeslal pouze ztracenou zprávu.

Velikost okna u Selective-repeat

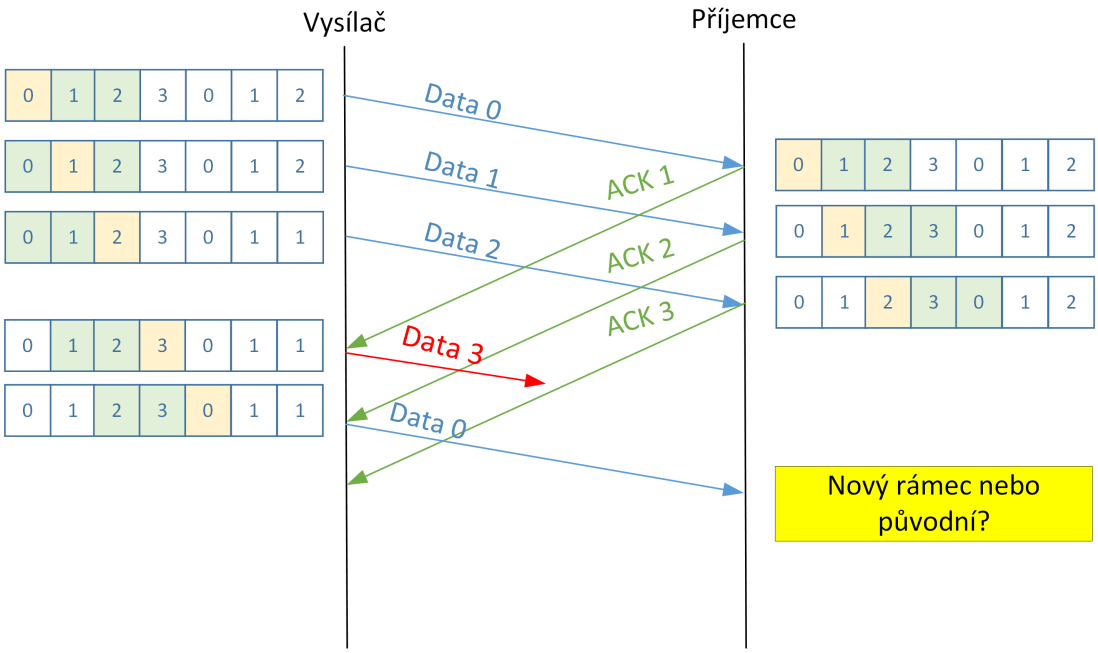
U protokolu Selective-repeat čelíme podobné potřebě zvolit vhodnou velikost posuvného okna, jako u GBN. Na rozdíl od GBN je potřeba udržovat vlastní okno pro obě strany, a to o stejné velikosti[17]. Pro správné fungování protokolu Selective-repeat musí být velikost okna menší nebo rovna polovině počtu sekvenčních čísel.

Důvod je znázorněn na obrázcích 4.6 a 4.7. Na obrázku 4.6 je popsán případ ztracení všech tří potvrzovacích zpráv. Okno na straně příjemce nyní očekává rámce se sekvenčními čísly 3, 0 nebo 1. Vysílač po skončení časovače odešle znovu první rámec, který je na straně příjemce přijatý. Problém je, že přijímač neví, zda se jedná o znovuodeslaný rámec nebo o nový.



Obr. 4.6: Znázornění problému chybně zvolené kombinace číslování s velikostí okna u protokolu Selective-repeat, kdy dochází k opětovnému vyslání původního rámce kvůli ztrátě potvrzovacích rámců.

Obrázek 4.7 naopak znázorňuje scénář, kdy se potvrzení po cestě neztratí. Vysílač po obdržení prvního potvrzení *ACK 1* zareaguje odesláním následujícího rámce v pořadí, *Data 3*, který se však po cestě ztratí. Mezi tím dojde i potvrzení *ACK 2* a je odeslán další rámec *Data 0*, který v pořádku dorazí k příjemci. Z pohledu přijímací stanice je komunikace na obou obrázcích 4.6 i 4.7 identická, ačkoliv pokaždé došlo k jinému chybovému stavu[8]. Z tohoto důvodu je velikost okna N-1, jak tomu bylo u GBN, nedostačující.



Obr. 4.7: Znázornění problému s příliš velkým oknem u protokolu Selective-repeat, kdy příjemce neví, zda je přijatý rámec opakovaně odeslán nebo se jedná o očekávaná data.

5 Použité technologie

Tato kapitola se zaměřuje na technologie, na kterých stojí implementace laboratorních úloh. Nejzásadnějším zástupcem je jazyk Java, ve kterém jsou naprogramovány všechny části aplikací. Součástí bude i představení klíčových knihoven použitých v programu.

5.1 Java

Java je třídní, objektově orientovaný programovací jazyk původně vyvinutý firmou Sun Microsystems. Svoji přenositelností se momentálně řadí mezi nejpoužívanější programovací jazyky na světě. Oproti jiným kompilovaným jazykům lze totiž zkompilovaný program napsaný v Javě spouštět na různých typech platforem bez nutnosti rekompilace. Toho je docíleno tím, že programy napsané v jazyce Java se kompilují do bajtkódu, který se následně interpretuje v JVM.

Java dále využívá automatický *Garbage collector*, což je proces, který sleduje využití paměti a hledá objekty, které už nejsou používány nebo na ně neexistuje žádná reference a tyto objekty následně maže[12]. Implementace obou laboratorních scénářů byla provedena v jazyce Java 8.

Bytecode & JVM

Následující rozbor se opírá o knihu [15]. Na rozdíl od jiných kompilovaných jazyků, výstupem Java kompilátoru není klasický spustitelný kód, ale *bajtkód*, což je vysoce optimalizovaný soubor instrukcí, který je navržen, aby byl spustitelný ve speciálním prostředí, který nazýváme *Java Virtual Machine* (JVM). Po zkompilování je výsledný bajtkód spustitelný na jakékoliv platformě. Stačí, aby pro tuto platformu existovala implementace JVM. Kdyby byly programy kompilovány do nativního jazyka, tak by pro každý typ CPU (Central Processing Unit) musely existovat různé verze programu.

Tento přístup také pomáhá Javě být bezpečnějším jazykem. JVM má vestavěné zabezpečující mechanismy, které dovolují programátorům psát bezpečné programu a zabraňuje škodlivému softwaru ohrozit operační systém, jelikož odstiňuje aplikace od přímé interakce se zdroji operačního systému.

Další důležitou součástí JVM je technologie HotSpot, která poskytuje dynamickou – JIT (Just-in-time) – kompilaci bajtkódu do nativního kódu pro zvýšení efektivity programu. JIT kompilátor kompiluje úseky kódu dle potřeby a pouze když nazná, že by mohl zefektivnit běh programu. Zbytek kódu zůstává interpretován JVM.

5.2 Jackson Databind

Jackson Databind je Java knihovna vyvinutá a stále udržovaná společností FasterXML, která poskytuje vysoce výkonné nástroje pro manipulaci s daty ve formátu JSON[1]. V implementaci laboratorních úloh je využívána především třída `ObjectMapper`, která umožňuje jednoduchou serializaci Java objektu do textového řetězce a inverzně deserializaci textových řetězců do příslušných objekt[3], jak je ukázáno na obrázku 5.1.

```
// An input object
TransmissionUnit o = TransmissionUnit.of( payload: "a",  sequenceNumber: 1, Type.ACK);

// Serialization into the string
String serialized = new ObjectMapper().writeValueAsString(o);

// Deserialization back to the object
TransmissionUnit deserialized = new ObjectMapper().readValue(serialized, TransmissionUnit.class);
```

Obr. 5.1: Ukázka serializace a deserializace s využitím knihovny Jackson Databind.

5.3 Apache Batik

Batik je Java knihovna, která nabízí sadu nástrojů pro aplikace a applety, které chtějí používat obrázky ve formátu SVG pro různé účely, jako je zobrazování, generování nebo manipulace. Například s použitím modulu SVG generátoru může Java aplikace jednoduše exportovat grafiku do SVG formátu, což je konkrétně v implementaci laboratorních úloh použito k vykreslení proběhlé komunikace[20].

SVG

SVG (Scalable Vector Graphics) je značkovací jazyk založený na XML (eXtensible Markup Language), který byl navržen k popisu dvojrozměrné vektorové grafiky. SVG pracuje se třemi typy grafických objektů: tvary ve vektorové grafice, obrázky a text. Souhrn funkcí nabízených SVG zahrnuje i práci s pokročilejšími technikami, jako: průhlednost, stínování, animace, skriptování, vnořené transformace, ořezové cesty a další. Tyto grafické objekty mohou být seskupovány, stylovány, transformovány a skládány do dříve vykreslených objektů. SVG dokumenty mohou být díky zmíněným funkcím interaktivní a dynamické. Animace mohou být definovány buď deklarativně, nebo naskriptováním[20].

[1] JSON (JavaScript Object Notation) je nezávislý formát pro přenos dat, který se umí vypořádat s primitivními datovými typy, objekty i polem hodnot.

5.4 Apache Maven

Maven byl založen s cílem vytvoření standardizovaného sestavování projektů napsaných v jazyce Java tak, aby existovala jasná definice toho, z čeho je projekt složen a jednoduchý způsob sdílení JARů[2] napříč různými projekty[21]. Pravidla, podle kterých se Java projekt sestavuje a kompiluje, případně definice dalších použitých pluginů, jsou popsány v projektovém souboru POM (Project Object Model). Mezi další užitečné funkce patří například i reportování výsledků testů včetně jejich pokrytí kódu nebo generování dokumentace přímo ze zdrojového kódu.

5.5 Docker

Docker je projekt, který cílí na přenositelnost aplikací, které jsou distribuovány ve formě kontejnerů. Tyto kontejnery obsahují všechno, co dockerizovaná aplikace potřebuje ke svému životu, včetně ovladačů a knihoven. S dockerem tedy odpadá potřeba adaptovat hostitelské prostředí za účelem spuštění nějaké aplikace, což velice zefektivňuje vývoj[22].

Technologie Docker je využita ve scénáři Směrování paketů. Navrhnuté řešení potřebuje ukládat záznamy o topologii uzlů do databáze, aby tyto informace mohly být později použity při zjišťování, kterým uzlům má být odeslána zpráva na základě cílové IP adresy. Tato databáze je hostovaná v Dockeru.

5.6 PostgreSQL

V minulé sekci 5.5 je uvedeno, že v laboratorní úloze Směrování paketů je zapotřebí databáze k udržování topologie uzlů. PostgreSQL je objektově relační databázový systém s otevřeným kódem, který je široce užíván v produkci především díky své spolehlivost, dlouholetému vývoji (přes 30 let) a konkurence schopnému výkonu[14].

[2] JAR (Java Archive) je archivační formát pro platformu Java, typicky agregující soubor tříd a přidružených metadat do jednoho souboru, který je vhodný k distribuci.

6 Návrh laboratorních úloh

Tato kapitola se zaměřuje na návrh a způsob vypracování samostatných laboratorních úloh a jejich scénářů. Součástí popisu bude i rozbor implementace a rozhraní vnitřních komponent systému.

6.1 Laboratorní okruh 1 - ARQ protokoly

První laboratorní úloha je zaměřená na protokoly řešící chybové stavy při přenosu dat na úrovni spojové vrstvy. Klade si za cíl především objasnit princip obecně ARQ protokolů a seznámení se s konkrétními implementacemi těchto protokolů. Student bude porovnávat protokol Stop-and-wait s typickými implementacemi mechanismu posuvného okna – Go-Back-N a Selective-repeat.

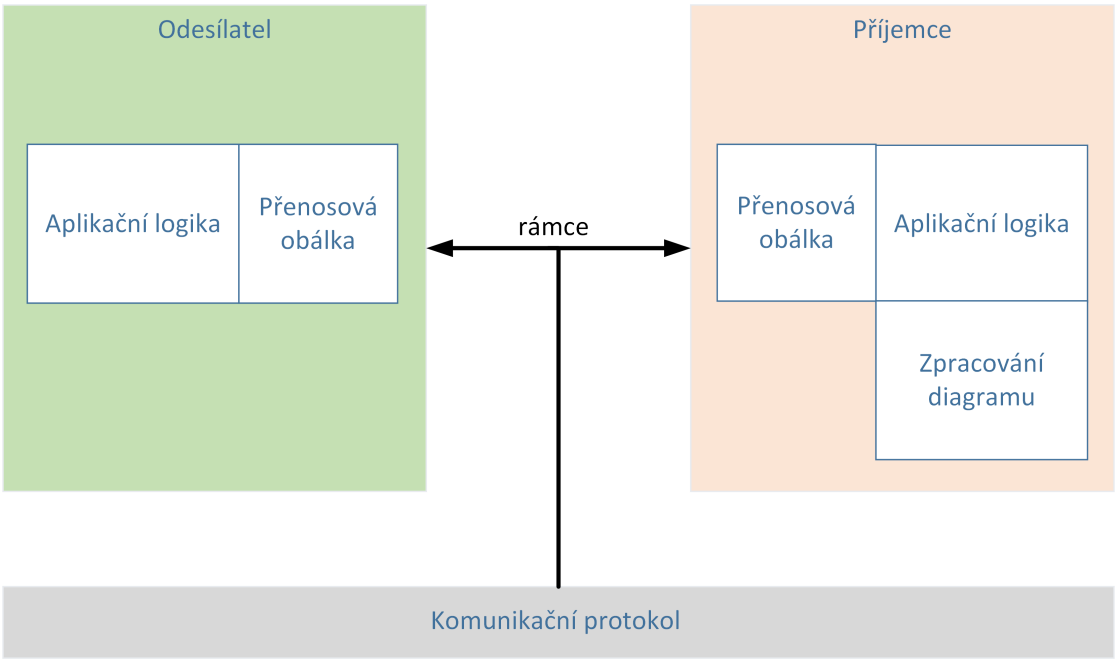
6.1.1 Návrh řešení

Vzhledem k tomu, že je laboratorní úloha zaměřena na mechanismus posuvného okna, k simulaci je potřeba vytvořit dva programy. První, který bude v roli vysílací stanice a druhý program, jehož úkolem bude přijímat a potvrzovat datové rámce. Kostry těchto dvou programů budou modifikovány studentem v průběhu laboratorního cvičení podle přiloženého návodu, s cílem provést studenta možnostmi a typickými znaky implementovaných protokolů a poukázat na rozdíly mezi nimi.

Obrázek 6.1 popisuje vysokoúrovňovou architekturu řešení. Obě implementace (vysílač vlevo a přijímač vpravo) jsou složeny z aplikační logiky podle role dané stanice a obálky, která se stará o příjem a odesílání zpráv. Přijímač pak navíc využívá dalšího modulu na grafické operace nad shromážděnými výsledky.

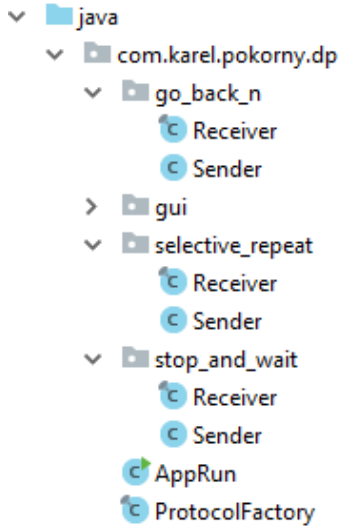
Zmíněná obálka (na obrázku označena jako Přenosová obálka) byla vytvořena za účelem pohodlnějšího a jednotvárného zpracování vstupů a výstupů, aby se student nemusel starat o implementaci těchto nezbytných operací, jejichž realizace je specifická pro daný programovací jazyk a s tématem úlohy nijak nesouvisí. Zároveň obsahuje i mechanismus, který sbírá data o přenosech dané aplikace. Tyto informace jsou důležité pro sestavení výsledného diagramu, který je vyprodukován aplikací ve formátu SVG (o vytvoření tohoto diagramu se stará poslední komponenta *Diagram processor*) po skončení přenosu. Aby bylo minimalizováno riziko zásahu do těchto důležitých (ale v kontextu s laboratorním scénářem nesouvisejících) mechanismů studentem, jsou oba moduly (Přenosová obálka i Zpracování diagramu) vloženy do každé z aplikací formou Maven závislostí. Pozornost studenta se tedy

upírá pouze na jádro aplikace – modul *Aplikační logika* – které implementuje až na pár nezbytností pouze daný ARQ protokol.



Obr. 6.1: Vysokoúrovňová architektura navrhnutého řešení v sestavě jednoho odesílatele a jednoho příjemce. Obě strany mezi sebou komunikují za použití zpráv komunikačního protokolu, které reprezentují přenosové rámce.

Obsahem studentem editovaného projektu jsou tedy pouze třídy implementující jednotlivé protokoly, balíček implementující uživatelské rozhraní a třída `ProtocolFactory` spouštící zvolený protokol. Struktura projektu je ukázána na obrázku 6.2.



Obr. 6.2: Struktura projektu implementujícího aplikační logiku.

Komunikační protokol

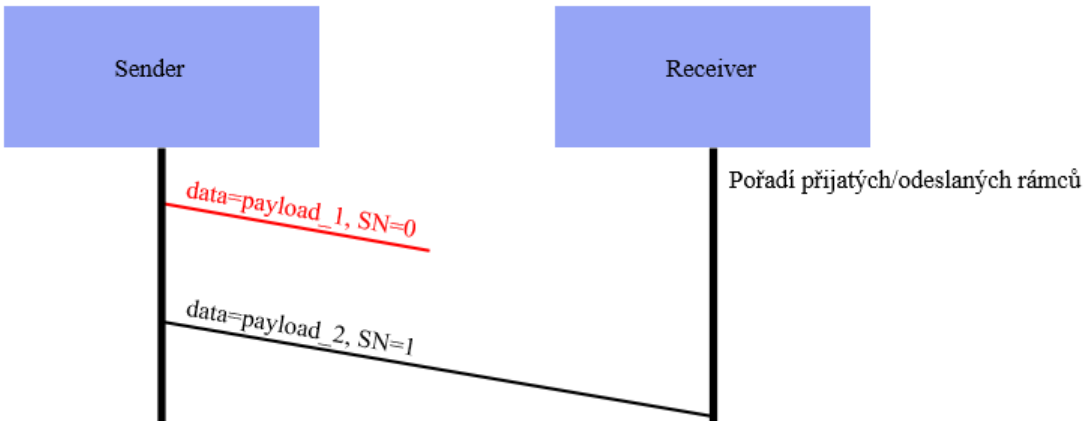
Vysílač s přijímačem si vyměňují zprávy, které jsou definovány vlastním protokolem. Protokol byl implementován tak, aby student nebyl zmaten vícero možnostmi jak korektně vytvářet zprávy a skládá se z následujících položek:

- **payload** - datová část zprávy
- **sequenceNumber** - pořadové číslo zprávy v kontextu s nastavenou velikostí přenosového okna
- **type** - typ zprávy. Mezi přípustné hodnoty patří *DATA*, která znamená, že odesílatel posílá ve zprávě data. Další hodnou je *ACK*. Zprávy obsahující tuto hodnotu mají potvrzovací význam přijetí datového rámce. Posledním typem je *REJ*, která reprezentuje rámec s negativním potvrzením.
- **checksum** - obsahuje kontrolní součet dané zprávy. K výpočtu hodnoty kontrolního součtu je použita standardní třída jazyku Java, `CRC32`. Vzhledem k oddělenému způsobu implementace bylo riziko podvrhnutí hodnoty kontrolního součtu téměř eliminováno.

Tvoření diagramů

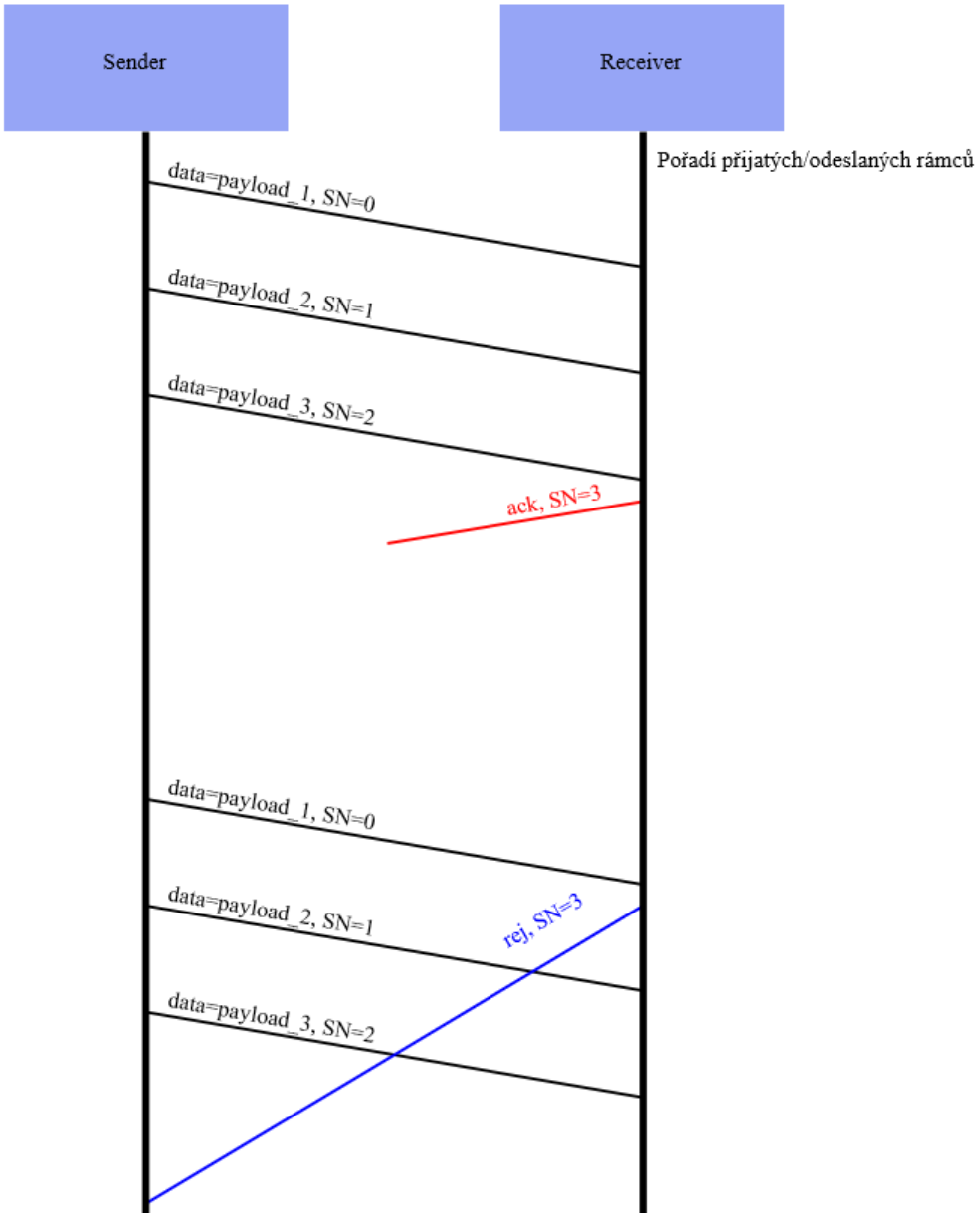
Diagram processor je knihovna, která na základě zalogovaných zpráv obou programů provede v první fázi jejich seřazení. Ve druhé fázi jsou pak seřazené zprávy zpracovány algoritmem, který je s podporou knihovny Apache Batik (viz. sekce 5.3) interpretuje do formy sekvenčního diagramu.

Výsledné sekvenční diagramy se nezaměřují na čas, ale na vzhled proběhlé komunikace. Zachycují především pořadí odeslání/přijetí rámců, spolu se stavem jejich doručení (ukázka na obrázku 6.3).



Obr. 6.3: Ukázka vizuálního odlišení rámce `payload_1`, který byl ztracen po cestě k příjemci, od rámce `payload_2`, který byl doručen úspěšně.

V diagramech je kladen důraz i na vizuální rozlišení situací, kdy je rámec opakovaně odeslán po vypršení časovače, nebo když příjemce odešlé rámec negativního potvrzení.



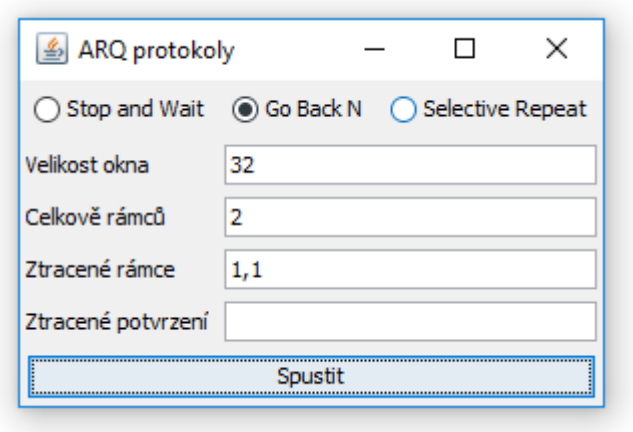
Obr. 6.4: Ukázka vizuálního rozlišení opakovaně odeslaných rámců a rámců negativního potvrzení.

Obrázek 6.4 zachycuje tento případ na protokolu Go Back N. Jelikož se potvrzení o doručení rámce `payload_1` nepodařilo na první pokus doručit, byl odesílatelem tento rámec opakovaně poslán po vypršení časovače – tato skutečnost je na diagramu zachycena odmlkou mezi rámcem `payload_3` a `payload_1`. Přijímač na opakovaně přijatý rámec korektně reaguje zprávou typu *rej*, ta je na diagramu znázorněna

modrou barvou a vidíme, že byla odesílatelem zpracována až po opakovaném odeslání rámce `payload_2`.

Uživatelské rozhraní

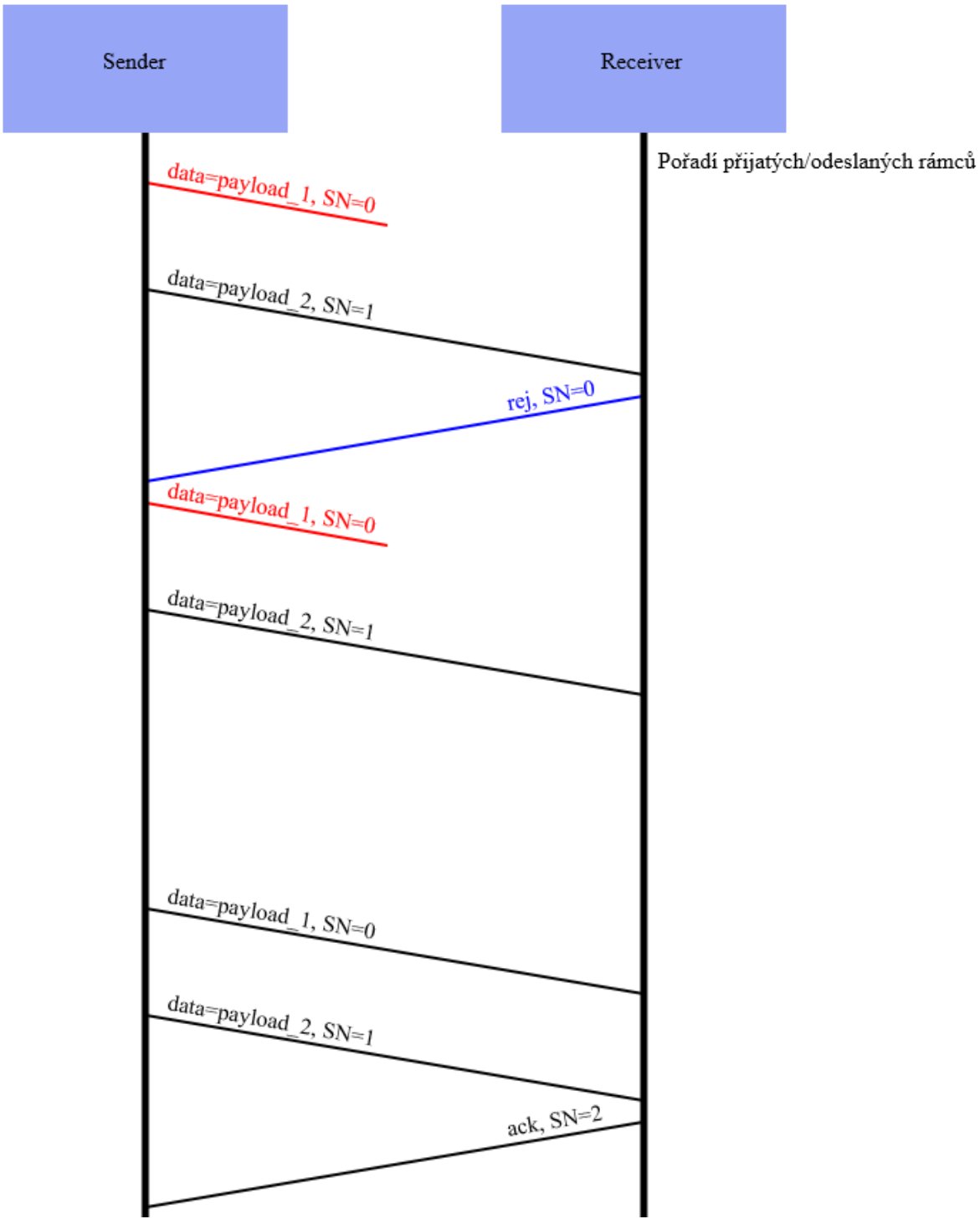
Součástí řešení je i jednoduché uživatelské rozhraní na konfiguraci scénáře. Rozhraní disponuje možností zvolit si typ ARQ protokolu, pro který chce uživatel daný scénář spustit, dále pak velikost přenosového okna, počet datových rámců k odeslání, specifikovat které z těchto rámců mají být po cestě ztraceny a specifikovat která potvrzení mají být po cestě ztracena.



Obr. 6.5: Uživatelské rozhraní ke konfiguraci scénářů.

Uživatelské rozhraní je ukázáno na obrázku 6.5. Nastavení v horní části značí, že scénář bude provedem protokolem Go Back N. Velikost přenosového okna je 32 – hypoteticky může odesílatel odeslat 32 datových rámců bez nutnosti čekání na potvrzení. Celkově odesílatel odešle 2 datové rámce. Po cestě se dvakrát ztratí první datový rámec (rámce, které mají být ztraceny se oddělují čárkou a mohou se opakovat). Stejným způsobem se zapisují ztracená potvrzení, která zde však v tomto případě nejsou.

Obrázek 6.6 obsahuje výstup aplikace po spuštění výše uvedeného scénáře. Na obrázku je vidět, že první rámec `payload_1` byl dvakrát ztracen dle konfigurace. Přijímač správně zareagoval na první přijatý rámec `payload_2` chybou. Odesílatel na přijetí tohoto zamítnutí zareagoval opětovným odesláním obou rámců. Tentokrát přijímač druhý datový rámec automaticky zahodil, jelikož stále čeká na první rámec `payload_1`. Mezitím na straně odesílatele dochází k vypršení časovače a odešlou se napotřetí oba z doposud nepotvrzených rámců. Tentokrát už jsou oba správně doručeny k přijímači, který jejich příjem potvrdí a tím celý scénář končí.



Obr. 6.6: Výstup aplikace ve formě sekvenčního diagramu po spuštění scénáře z obrázku 6.5

.

6.1.2 Scénář laboratorní úlohy

Tato sekce se zaměřuje na popis laboratorní úlohy, která bude řešitelem vykonávána v předpřipraveném virtuálním prostředím a s použitím návodu v příloze A. K dispozici bude referenční aplikace z podkapitoly 6.1.1 a neúplné zdrojové kódy této aplikace, kam bude řešitel doplňovat některé z klíčových vlastností ARQ protokolů.

1. První část je věnována úvodu do problematiky. Student je seznámen s pojmem *ARQ protokoly* a k čemu tyto protokoly slouží.
2. Následující část seznamuje studenta s aplikací, která bude využívána k simulacím jednotlivých scénářů, a jejím uživatelským rozhraním. Součástí je i představení výstupu aplikace ve formě sekvenčního diagramu, který znázorňuje uskutečněnou komunikaci mezi vysílačem a příjemcem.
3. Jakmile je student seznámen se základními principy ARQ protokolů a referenční aplikací, zaměří se na první z těchto protokolů, Stop-and-Wait. První část této kapitoly je zaměřena na popis protokolu Stop-and-Wait s názornou ukázkou komunikace v režii tohoto protokolu.
4. Další část je věnována úkolům týkajících se protokolu Stop-and-Wait:
   (a) Student je naveden ke spuštění aplikace uvnitř virtuálního prostředí.
   (b) Na konkrétním případě je demonstrováno chování protokolu v případě, že nedojde ke ztrátě žádného z rámců. Na tomto jednoduchém příkladu si mimo jiné student vyzkouší práci s referenční aplikací a zobrazení výstupu ve formě diagramu. Scénář je doplněn o dotaz na číslování rámců sekvenčními čísly.
   (c) Na dalším příkladě je ukázána schopnost protokolu Stop-and-Wait zotavit se z chybových stavů. V příkladu dojde ke ztrátě datového i potvrzovacího rámce. Student je dále vyzván ke krátkému vysvětlení událostí zobrazených na výsledném diagramu.
   (d) Jakmile je tato část splněna, je student naveden do vývojového prostředí k pochopení implementace protokolu Stop-and-Wait z komentovaného zdrojového kódu. V kódu je vyznačeno několik klíčových úseků, které má student za úkol doimplementovat.
   (e) Správnost svého řešení si následně student ověří spuštěním konkrétního scénáře, ve kterém dojde ke ztrátě jednoho datového rámce a dvou potvrzovacích.
5. Jakmile student dokončí úkoly týkající se protokolu Stop-and-Wait, přesuneme se k pokročilejším protokolům – prvním z nich je protokol Go Back N:
   (a) Nejprve je vysvětlen přínos techniky posuvného okna oproti jednoduchému protokolu Stop-and-Wait spolu s názornou ukázkou.

(b) V další části je popsán protokol Go Back N (včetně jeho chování v chybových stavech) a část věnovaná správné volbě velikosti posuvného okna pro tento protokol spolu s ukázkami špatné i správné volby.

(c) Po absolvování teoretické části si student na konkrétním příkladě nasimuluje scénář bez ztráty rámců. Tento scénář je následován několika dotazy, převážně týkajících se kumulativního potvrzování.

(d) Následující příklad demonstruje zotavení se protokolu Go Back N ze ztráty datového rámce a je opět následován dotazy na vzniklou situaci.

(e) Poslední příklad této části je věnován ztrátě potvrzovacích rámců. Dotazy ohledně tohoto příkladu jsou směřovány na hodnoty sekvenčních čísel a hypotetických událostí, které by mohly nastat.

(f) Nakonec se student opět zaměří na zdrojový kód, tentokrát příslušícího protokolu Go Back N, seznámí se s komentovanou implementací protokolu a stejně jak v minulé kapitole doimplementuje některé chybějící části klíčové pro Go Back N.

(g) Výsledná implementace je následně ověřena na konkrétním scénáři, ve kterém dojde ke ztrátě datových i potvrzovacích rámců.

6. Pooslední úkol se věnuje protokolu Selective Repeat:

   (a) Na začátku této kapitoly je uvedeno srovnání protokolu Selective Repeat a Go Back N spolu s rozdílemi v implementaci a názornými ukázkami. Pozornost je věnována také volbě velikosti posuvného okna u protokolu Selective Repeat.

   (b) Na konkrétním příkladě je pak ukázán stejný scénář, jako jsme použili u protokolu Go Back N bez ztráty rámců, s tím rozdílem, že je proveden v režii protokolu Selective Repeat. Dotazy ohledně tohoto příkladu si kladou za cíl poukázat na to, že se výstupy procesně nijak neliší.

   (c) Další příklad se zaměřuje na případ, kdy dojde ke ztrátě datového rámce. Dotazy se zaměřují na porovnání výstupu tohoto scénáře s výstupem za použití protokolu Go Back N.

   (d) V posledním příkladě se odehraje ztráta potvrzovacích rámců a je opět kladen důraz na skutečnost, že chování protokolů Selective Repeat a Go Back N se v těchto případech neliší.

   (e) Po vykonání všech vzorových příkladů se student opět přesune do zdrojového kódu, který implementuje protokol Selective Repeat. Jakmile se seznámí s komentovaným kódem, zaměří se opět na některé z klíčových charakteristik tohoto protokolu a doimplementuje tyto chybějící části.

   (f) Správnost implementace je ověřena na konkrétním scénáři, ve kterém dojde ke ztrátě datového rámce i několika potvrzovacích.

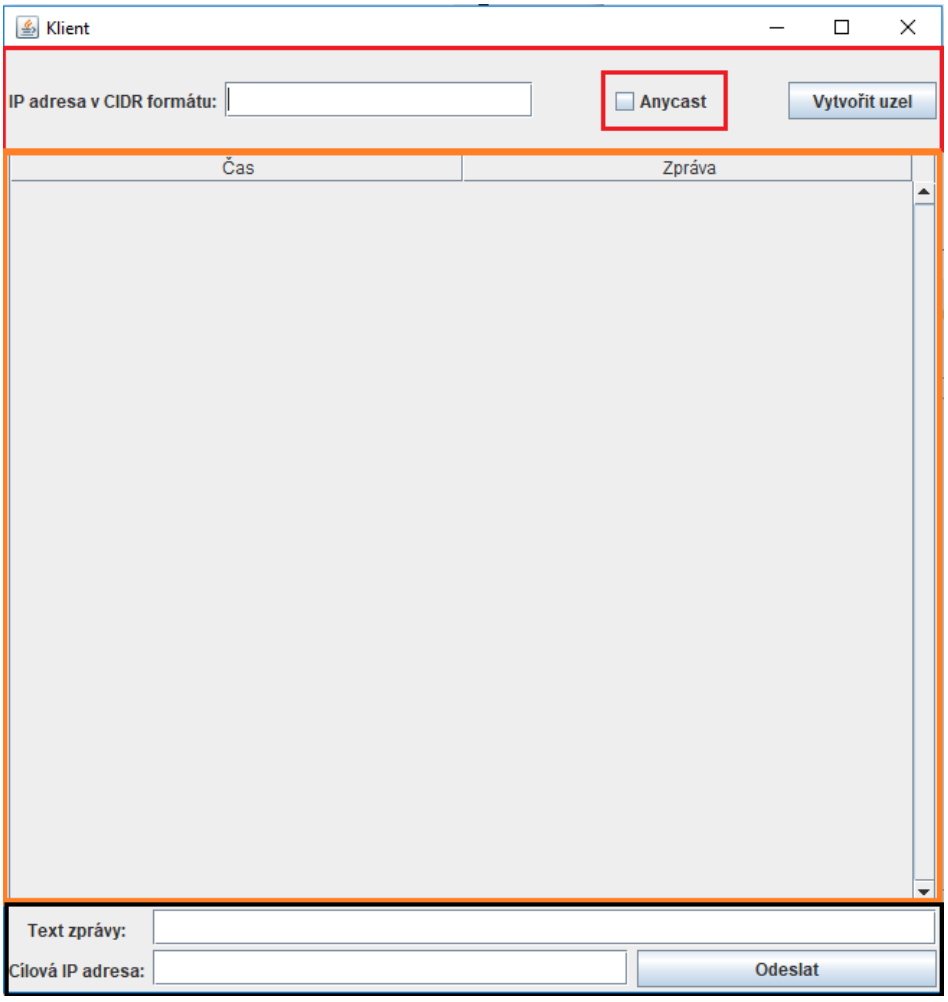
7. Nakonec je student vyzván, aby vypsal nevýhody protokolu Selective Repeat.

6.2 Laboratorní okruh 2 - Typy přenosů v paketových sítích

V této laboratorní úloze si student za pomocí předpřipravených programů objasní různé typy přenosu paketů a následně se jejich chování pokusí doimplementovat.

6.2.1 Návrh řešení

Při řešení byl největší důraz kladen na možnost jednoduché demonstrace klíčových principů každého z typů přenosu. Z tohoto důvodu je aplikace koncipována na způsob vytváření síťových uzlů, které budou tvořit jednoduchou síťovou topologii bez směrovacích síťových prvků. Každý z uzlů má nastavenou IP adresu, má možnost přihlašovat se k multicastovým skupinám, odesílat zprávy ostatním uzlům v topologii a zobrazovat a odlišovat zprávy, které tento uzel přijal. Každý uzel je reprezentován jednou instancí oknové aplikace, která je ukázána na obrázku 6.7.

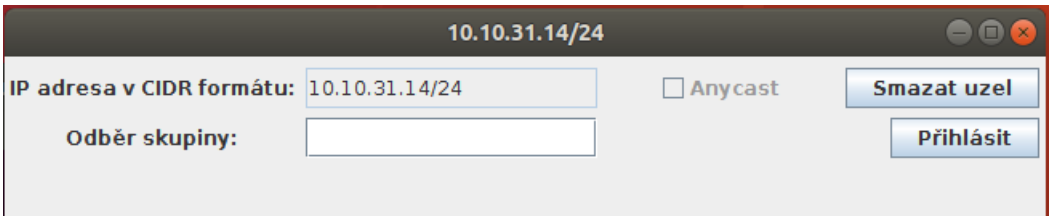


Obr. 6.7: Vzhled uživatelského rozhraní aplikace.

Obrázek 6.7 je barevně rozdělen do tří částí.

- Červené ohraničení nahoře se zaměřuje na konfiguraci daného uzlu. Do prázdného pole patří IP adresa uzlu a je zde i možnost nastavit uzel jako člen anycastové skupiny.
- Oranžový rámec obsahuje historii přijatých zpráv a čas jejich přijetí.
- Konečný černý rámec obsahuje prvky potřebné k odešílaní zpráv ostatním uzlům. První prázdné pole je určeno k zadání obsahu zprávy, druhé pole slouží k zadání cílové IP adresy, na kterou bude daná zpráva odeslána.

Pro případ registrace uzlu k odběru zpráv slouží nové pole (ukázáno na obrázku 6.8), které se automaticky objeví pod polem IP adresy hned po provedení registrace IP adresy uzlu.



Obr. 6.8: Pole pro odběr zpráv z multicastové skupiny.

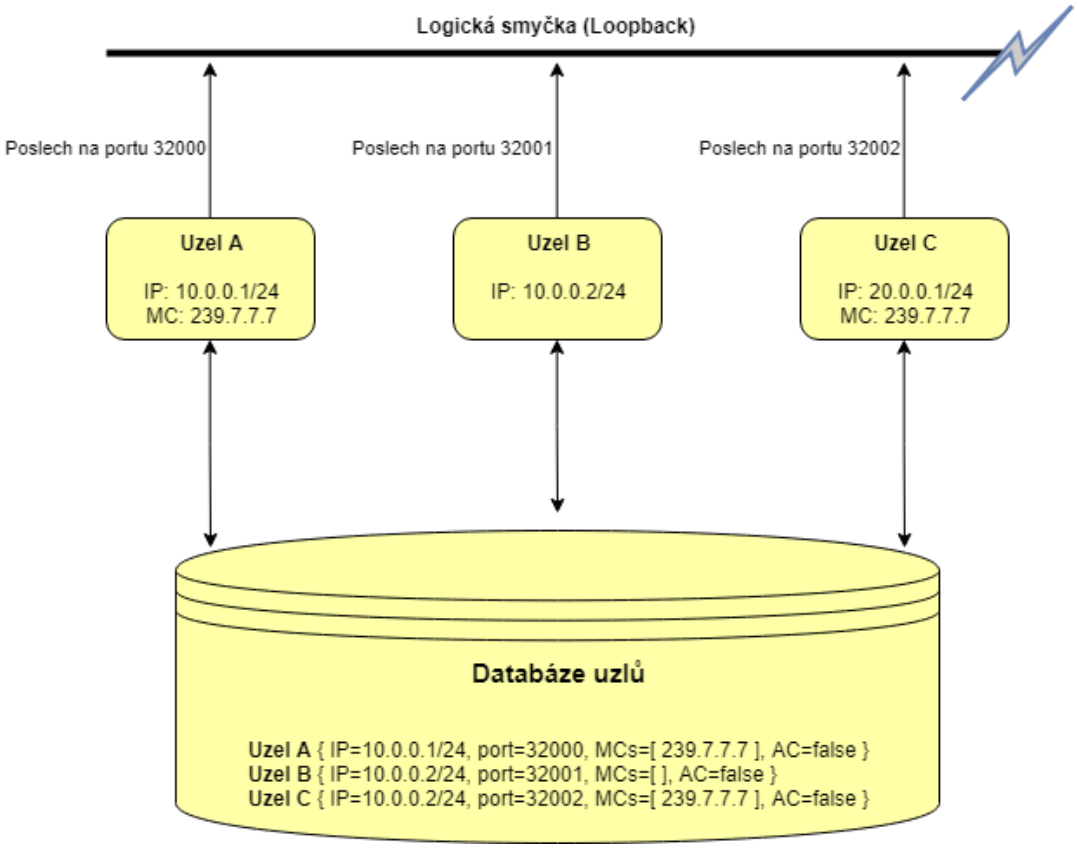
Komunikace na pozadí

Celá komunikace uzlů zastoupených IP adresami je simulována na lokální smyčce. Situace je znázorněna na obrázku 6.9.

Úplně nahoře je znázorněna síťová vrstva virtuálního stroje (konkrétně jen síťové rozhraní lokální smyčky), přes kterou probíhá odesílání a přijímaní zpráv. Po spuštění každé aplikace se čeká na zadání IP adresy. Jakmile je IP adresa stvrzena, proběhne její validace a na pozadí aplikace se odehraje vyjednání dosud neobsazeného portu (již obsazené porty jsou uloženy v databázi) a jeho zabrání uzlem – v kontextu s obrázkem 6.9 si např. uzel A s IP adresou 10.0.0.1 vyjedná port 32000. Následně se spustí v novém vlákně proces, který pouslouchá na síťovém rozhraní lokální smyčky a obdrženém portu, a přijímá zprávy. V databázi se vytvoří záznam s konfigurací uzlu – tento záznam je později použit při odesílání zpráv, kdy je potřeba zajistit, aby se na základě cílové IP adresy odeslala zpráva do správného uzlu (tedy proběhne překlad IP adresy na port).

O inicializaci poslechu, samotný překlad cílové adresy na port i vyjednávání hodnoty portu při registraci nového uzlu se stará samotná aplikace, takže student bude při práci na svém řešení pracovat pouze s IP adresami a řešit pouze to, do

jakých stanic má odesílaná zpráva být doručena. Po zavření okna aplikace je záznám o uzlu automaticky smazán.



Obr. 6.9: Architektura scénáře – směrování paketů. Zkratka *MCs* označuje multicastové skupiny, zkratka *AC* označuje anycast.

Interakce s kódem

Ke vzniku aplikace, kterou je realizován scénář z obrázku 6.9 byly definovány celkem tři datové objekty, každý implementovaný vlastní třídou:

1. **Port** – slouží jako objektová abstrakce portu, které se přiřazují uzlům při jejich vytváření. Na tomto portu uzel poslouchá a čte příchozí zprávy. Tyto objekty jsou následně uchovávány v databázi po dobu životnosti klientského uzlu. Nepředpokládá se, že by student potřebovat jakkoliv manipulovat s instancemi této třídy.
2. **Node** – objekt samotného vytvářeného uzlu. Stejně jako `Port` i instance třídy `Node` jsou uchovávány v databázi po celou dobu své životnosti. Uzel je jediným objektem, který bude řešitele zajímat, jelikož v sobě uchovává IP adresu uzlu, informaci o tom, zda je uzel v režimu anycast a skupinové adresy, ke kterým je uzel přihlášen, což jsou všechny informace potřebné k implementaci

rozhodovacího algoritmu.
`Node` poskytuje také několik metod pro snadnou manipulaci s hodnotami atributů instancí této třídy. Jejich přehled a popis je na obrázku 6.10.

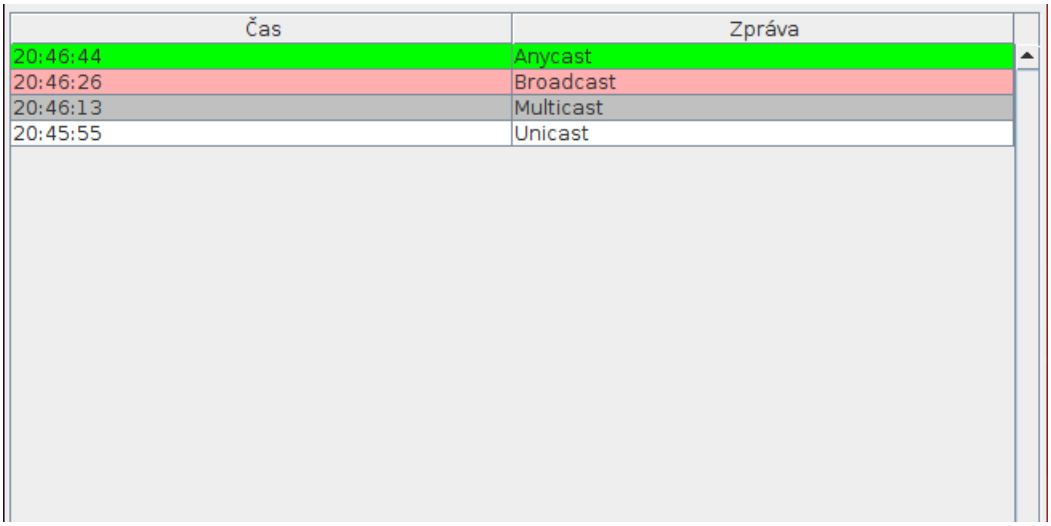
```
/**
 * Vrátí IP adresu tohoto uzlu.
 */
public String getIpAdr()

/**
 * Vrátí informaci, zda je uzel v režimu anycast.
 */
public boolean isAnycastMode()

/**
 * Zjišťuje, zda je uzel přihlášen k odběru zpráv dané skupiny.
 * @param group IP adresa skupiny
 */
public boolean containsMulticastGroupSubscription(String group)
```

Obr. 6.10: Přehled a popis metod třídy Node určených k manipulaci s atributy.

3. **Datagram** – je abstrakce přenášených zpráv mezi jednotlivými uzly. Skládá se z atributů `message`, která obsahuje tělo odeslané zprávy, a `type`, který nese informaci o tom, jakým přenosovým typem (unicast, multicast, ..) je zpráva přenesena. Tato informace je zpracovávána na straně posluchače a slouží k vizuálnímu rozlišení těchto typů v přehledu přijatých zpráv. Obrázek 6.11 ukazuje toto rozlišení:



Obr. 6.11: Barevné rozlišení přijatých zpráv na základě typu přenosu.

Aplikace jako celek je navržena tak, aby se student při řešení laboratorní úlohy nemusel zaobírat ničím jiným, než jsou principy jednotlivých typů předávání paketů a jak jsou svázány s konkrétními hodnotami IP adres. Z toho důvodu byl zdrojový kód programu dekomponován na:

- Balíček *gui* obsahující implementaci uživatelského rozhraní.
- Balíček *model* obsahující všechny datové objekty aplikace (`Port, Node, Datagram`).
- Balíček *repository*, který završuje komunikaci s databází.
- Třídu `Runner`, která spouští aplikaci.
- Třídu `CIDRUtils`, která slouží ke snadné manipulaci s IP adresami.
- Třídu `RoutingService`, která řídí veškeré rozhodování o tom, kterým uzlům má být zpráva odeslána.

```
/**
 * Tuto metodu vytvoreny uzel pouziva k odeslani zprav jinym uzlum.
 * Vstupnimy parametry jsou hodnoty poli ve spodni casti programu.
 *
 * @param message zprava ktera ma byt odeslana
 * @param destAdr cilova IP adresa
 */
public void sendMessage(JTextField message, String destAdr) {

    // Kolekce vsech vytvorenych uzlu
    final List<Node> nodes = NodeRepository.getAll();

    /**********************************************************************************
     *
     *  Do teto metody naimplementujte smerovani pro jednotlive rezimy (UNICAST, MULTICAST, ..)
     *
     **********************************************************************************/

    /**
     * Nasledujici kod ukazuje priklad implementace jednosmerveho typu prenosu pro IPv4 + IPv6:
     *
     * Projdu jeden uzel po druhem a u kazdeho zkontroluji, zda ma stejnou IP adresu, jako je moje
     *  cilova IP adresa (destAdr). Pokud se adresy shoduji, odeslu zpravu tomuto uzlu a pokracuju.
     */
    for (Node n : nodes) { //
        try {
        // Prevedu CIDR adresu do objektu, ve kterz mi usnadni praci s jednotlivymi castmi adresy
            final CIDRUtils potentionalTarget = new CIDRUtils(n.getIpAdr());
            // Je IP adresa uzlu stejna, jako cilova IP adresa?
            if (potentionalTarget.getAddress().equalsIgnoreCase(destAdr)) {
                send(n, message.getText(), Datagram.Type.UNICAST);
            }
        } catch (IllegalArgumentException ex) {
            LOGGER.debug("Node {} is not IPv4/6 complient.", n.getIpAdr());
        }
    }
}
```

Obr. 6.12: Část kódu, do kterého bude student vkládat své řešení.

Třída `RoutingService` obsahuje klíčovou metodu `sendMessage(...)`, která slouží k implementaci směrovacích principů. Obrázek 6.12 popisuje tento úsek kódu s již předvyplněným řešením unicastového směrování.

6.2.2 Scénář laboratorní úlohy

Součástí řešení je i scénář, který je založen na aplikaci popsané v předchozích sekcích a podle kterého budou studenti postupovat za účelem objasnění si jednotlivých režimů přenosu. Kompletní návod je součástí přílohy B.

1. Prvním krokem je seznámení se s problematikou, především připomenutí, na jakých principech fungují paketové sítě a typů komunikace, které se v nich uplatňují. Součástí je i detailnější specifikace jednosměrového, všesměrového, výběrového a vícesměrového přenosu.
2. Po teoretickém úvodu je student seznámen s aplikací, kterou bude pro tuto laboratorní úlohu používat (jak se vzorovým řešením, tak i s kostrou aplikace, kterou bude upravovat) a s virtuálním prostředím, ve kterém se nachází. Mimo jiné je v této části uvedeno i jakým způsobem se aplikace spouští a které části kódu jsou pro studentovo řešení důležité.
3. Jakmile je student seznámen s rozsahem cvičení a prostředím, ve kterém bude pracovat, přecházíme na režimy přenosu v protokolu IPv4. V první podčásti je otevřeno téma jednosměrného přenosu. Na konkrétním příkladě dvou uzlů se student s tímto typem přenosu seznámí za použití referenčního řešení.
4. Student se seznámí i se složitějšími případy použití jednosměrného přenosu, jako je komunikace více uzlů v různých sítích a chování v případě více uzlů se stejnou IP adresou.
5. Jakmile proběhne seznámení s jednosměrným přenosem a je zodpovězena kontrolní otázka, otevře si student úsek kódu dle návodu a seznámí se s konstrukcemi použitými k implementaci jednosměrného přenosu, jelikož tento typ přenosu je v přiložené kostře pro ukázku již naimplementován.
6. Dalším typem přenosu v pořadí bude všesměrový přenos. Za pomoci čtyř uzlů, z nichý jeden leží v jiné síti, a souvisejících otázek se student seznámí se základním principem tohoto typu přenosu.
7. Po experimentech s všesměrovými přenosy se student zaměří na implementaci všesměrového chování do vlastního řešení. Správnost řešení je ověřena na jiném konkrétním příkladu opět doplněném o související otázky.
8. Jakmile student dokončí úkol vztahující se k všesměrovému přenosu, následuje přenos skupinový. K docílení skupinového přenosu je zapotřebí uzly nejprve přihlásit k odběru zpráv ze skupinové adresy. Student se se způsobem této konfigurace seznámí a vyzkouší si na konkrétním příkladě čtyř uzlů, z nichž tři

budou odebírat zprávy stejné skupiny, přenos na základě přihlášených skupin.

9. Až si student osvojí skupinový přenos, přesune se opět k implementaci a doplní do svého řešení potřebný kód, který bude posílat zprávy uzlům přihlášeným k dané skupině.
10. Posledním typem přenosu, se kterým bude student seznámen u protokolu IPv4 je typ výběrový. Student se nejprve obeznámí se způsobem, jakým uzly nakonfigurovat do režimu anycast a následně si na příkladě třech uzlů demonstruje tento typ přenosu. Příklad je opět doplněn o kontrolní dotaz a tipy k implementaci.
11. Následně student doplní implementaci do svého řešení a zkontroluje správnost na příkladě se čtyřmi uzly. Tři uzly sdílí stejnou IP adresu, ale pouze dva z nich jsem v režimu anycast. Zpráva je tedy vždy doručena uzlu ve standarním režimu a zároveň náhodně doručena jednomu ze dvou uzlů sdílející anycastovou skupinu.
12. Jakmile se student seznámí se všemi typy přenosu u protokolu IPv4, přesuneme se ke druhé části, kdy si tyto přenosy ukážeme na protokolu IPv6.
13. První část je úvodem do IPv6 adresace. Nachází se zde rozbor typů adres, které nová verze protokolu IP přináší, s důrazem na globální unikátní adresy, které jsou důležité pro tento úkol.
14. Na konkrétní množině příkladů si student ilustrativně ověří, že jednosměrný a výběrový přenos je u protokolu IPv6 velice podobný jeho starší verzi IPv4. Vyzkouší si opět i jejich kombinaci a upraví vlastní řešení tak, aby podporovalo přenos zpráv s použitím globálních unikátních IPv6 adres.
15. Poslední částí laboratorní práce je skupinový typ přenosu s použitím IPv6 adres. Nejprve je uveden účel těchto adres, jejich skladba a problematika absence všesměrových adres oproti původnímu protokolu IPv4. Dále je vysvětlena podstata *skupinové adresy pro vyzývaný uzel*.
16. Na základě znalostí načerpaných z teoretického základu a referenční aplikace si student na příkladě čtyř uzlů demonstruje, jak tyto jednotlivé typy skupinových adres fungují, a odpoví na související otázky.
17. V posledním úkolu si student podporu těchto adres doimplementuje do svého řešení.

Výstupem laboratorní úlohy tedy bude protokol s odpovědmi na otázky, které byly položené během jednotlivých kroků scénáře, a hotová aplikace, která se bude při odesílání/přijímání zpráv chovat stejně, jako referenční řešení, pomocí kterého jsou demonstrovány jednotlivé příklady v průběhu laboratorní úlohy.

7 Závěr

Cílem této práce bylo seznámit se s problematikou komunikačních protokolů a jejich režimů přenosu v paketových sítích, a na základě těchto znalostí navrhnout dva komplexní scénaře, na kterých bude možno si tyto techniky vyzkoušet. Scénáře a vytvořené aplikace, které jsou jejich součástí, byly vytvářeny pro předměty zaměřené na komunikační technologie a jejich obsah byl průběžně konzultován s vedoucím této práce.

V rámci této práce jsem nastudoval a rozebral problematiku týkající se přenosu paketů v paketových sítích, zvláště pak jednotlivé typy přenosů a jejich vztah ke komunikačním protokolům IPv4 a IPv6. Dále jsem se zaměřil na ARQ protokoly a porovnání jejich schopností zotavit se z různých chybových stavů. Pro oba okruhy jsem navrhl komplexní laboratorní scénáře, přičemž každý ze scénářů je podpořen aplikací disponující jednoduchým uživatelským rozhraním, která umožňuje principy těchto okruhů demonstrovat, a připraveným virtuálním prostředím pro zachování platformové nezávislosti.

Součástí každého scénaře jsou teoretické úvody do probírané problematiky, části věnované seznámení se s referenční aplikací a úkoly související s jednotlivými tématy probírané látky. Postup je provázen kontrolními dotazy, na které student během řešení odpovídá, a které odhalují krajní případy dané technologie. Na závěr každého úkolu si student zkusí doimplementovat klíčové myšlenky algoritmu do upraveného kódu referenční aplikace.

První scénář se zaměřuje na ARQ protokoly. Cílem je seznámit studenta s principy protokolů Stop-and-Wait, Go Back N a Selective Repeat a poukázat na rozdíly mezi nimy. Prvně je rozebrán protokol Stop-and-Wait, na kterém jsou ukázány základní myšlenky pro zotavení se z neočekávaného stavu během přenosu. Druhý krok je zaměřen na rozbor přenosového okna a výhody, které přináší protokolu Go Back N oproti Stop-and-Wait. Poslední část se skládá z představení protokolu Selective Repeat, zavedení přenosového okna na obě komunikující strany a výsledný benefit oproti protokolu Go Back N.

Druhý scénář uvádí a porovnává jednosměrné, skupinové, výběrové a všesměrové typy přenosu. První část je věnována těmto typům přenosu v protokolu IPv4. V této části se student seznámí s každým typem přenosu a obecnými principy adresace. Druhá polovina je zaměřena na rozdíly v typech přenosu, které s sebou přinesl protokol IPv6. Zde je pozornost věnována především různým typům skupinových adres.

V rámci práce byly ke každému scénáři vypracovány podrobné návody, kterými se bude student při řešení laboratorní úlohy řídit. Návod v příloze A popisuje scénář věnovaný ARQ protokolům, návod v příloze B popisuje scénář zaměřený na typy

přenosu v paketových sítích. V poslední části každého z návodů se nacházejí odpovědi na otázky pokládané v průběhu scénáře.

Na práci lze navázat dalšími typy přenosu (např. podpora dalšího typu IPv6 adres) i novými ARQ protokoly, jelikož zdrojové kódy aplikací byly navrženy s ohledem na snadnou rozšiřitelnost a přiložené virtuální prostředí není svázáno se specifickou verzí implementace. Virtuální prostředí bylo testováno na programu VirtualBox ve verzi 6.0.4.

Výstupy této práce si kladou za cíl usnadnit pochopení jednotlivých režimů přenosů, principů ARQ protokolů a především jejich rozdílné chování a benefity v určitých hraničních případech.

Literatura

[1] Ahuja, P.: *Sliding Window Protocol | Set 2 (Receiver Side)*. [Online; navštíveno 08.12.2018].
URL `https://www.geeksforgeeks.org/sliding-window-protocol-set-2-receiver-side/`

[2] Bonaventure, O.: *Computer Networking : Principles, Protocols and Practice*. Lulu.com, 2016, ISBN 978-1365185830.

[3] Jenkov, J.: *Jackson ObjectMapper*. [Online; navštíveno 22.11.2018].
URL `http://tutorials.jenkov.com/java-json/jackson-objectmapper.html`

[4] Jeřábek, J.: *Pokročilé komunikační techniky*. FEKT Vysokého učení technického v Brně, 2015.

[5] Kabelová, A.; Dostálek, L.: *Velký průvodce protokoly TCP/IP a systémem DNS 5. aktualizované vydání*. Computer Press, 2012, ISBN 978-80-251-2236-5.

[6] Kolka, Z.: *Počítačové a komunikační sítě*. Vysoké učení technické v Brně, 2012, ISBN 978-80-214-5171-1.

[7] Koton, J.: *Moderní síťové technologie*. FEKT Vysokého učení technického v Brně, 2014, ISBN 978-80-214-5026-4.

[8] Kurose, J. F.; Ross, K. W.: *Computer Networking 3rd edition*. Addison Wesley, 2005, ISBN 978-0321269768.

[9] Matoušek, P.: *Síťové aplikace a jejich architektura* . Vutium, 2014, ISBN 978-80-2143-766-1.

[10] Mir, S. A.; Sharma, M.: A comparative study of X.25, Frame Relay and ATM in High Speed networks. *International journal for research in applied science and engineering technology (IJRASET)*, Vol.2 Issue IV, April 2014: s. 243–245, ISSN 2321-9653.

[11] Mrázek, O.: *SNMP protokol a jeho využití*. [Online; navštíveno 15.11.2018].
URL `https://vyvoj.hw.cz/produkty/snmp-protokol-a-jeho-vyuziti.html`

[12] Oracle: *Java Garbage Collection Basics*. [Online; navštíveno 22.11.2018].
URL `https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html`

[13] Petrovský, P.: *Multicast v IP sieťach*: bakalárska práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikácií, 2013. 54s. Vedúci práce bol Ing. Radko Krkoš.

[14] PostgreSQL: *New to PostgreSQL?* [Online; navštíveno 15.04.2019].
URL `https://www.postgresql.org/`

[15] Schildt, H.: *Java A Beginner's Guide Sixth Edition*. McGraw-Hill Education, 2014, ISBN 978-0-07-180926-9.

[16] Schulzrinne, H.; Casner, S.; Frederick, R.; aj.: *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550, RFC Editor, July 2003, [Online; navštíveno 12.07.2016].
URL `https://tools.ietf.org/html/rfc3550`

[17] Sharan, A.: *Sliding Window Protocol | Set 3 (Selective Repeat)*. [Online; navštíveno 08.12.2018].
URL `https://www.geeksforgeeks.org/sliding-window-protocol-set-3-selective-repeat/`

[18] Sosinsky, B.: *Mistrovství – počítačové sítě*. Computer Press, a.s., 2011, ISBN 978-80-251-3363-7.

[19] Tech Differences: *Difference Between Go-Back-N and Selective Repeat Protocol*. [Online; navštíveno 26.11.2018].
URL `https://techdifferences.com/difference-between-go-back-n-and-selective-repeat-protocol.html`

[20] The Apache Software Foundation: *Apache^{TM} Batik SVG Toolkit*. [Online; navštíveno 24.11.2018].
URL `https://xmlgraphics.apache.org/batik/`

[21] The Apache Software Foundation: *What is Maven?* [Online; navštíveno 24.11.2018].
URL `https://maven.apache.org/what-is-maven.html`

[22] Torre, C. D. L.: *What is Docker?* [Online; navštíveno 15.04.2019].
URL `https://docs.microsoft.com/cs-cz/dotnet/standard/containerized-lifecycle-architecture/what-is-docker`

Seznam příloh

A Návod k úloze – ARQ protokoly

A.1 Úvod

Během přenosu dat přirozeně čas od času dochází k poškození nebo ztrátě přenášené informace, ať už kvůli možným výpadkům v síti nebo z důvodu zahlcení sítě, při kterém jsou pakety systematicky zahazovány. K vypořádání se s těmito problémy jsou zavedeny techniky, které řídí komunikace a reakce na vzniklé chybové situace.

K samotnému zjištění, že stanice přijala poškozený paket, slouží tzv. kontrolní součet FCS (Frame Check Sequence), jehož hodnota je součástí přenosového rámce. FCS se na straně vysílače spočítá pomocí CRC (Cyclical Redu-nancy Check), což je kód, který se používá k detekci chyb. Poté se na straně příjemce z přijatého rámce vypočítá nová hodnota FCS, která se porovná s původní FCS z daného rámce a pokud se hodnoty liší, předpokládá se, že byl přijatý rámec poškozen.

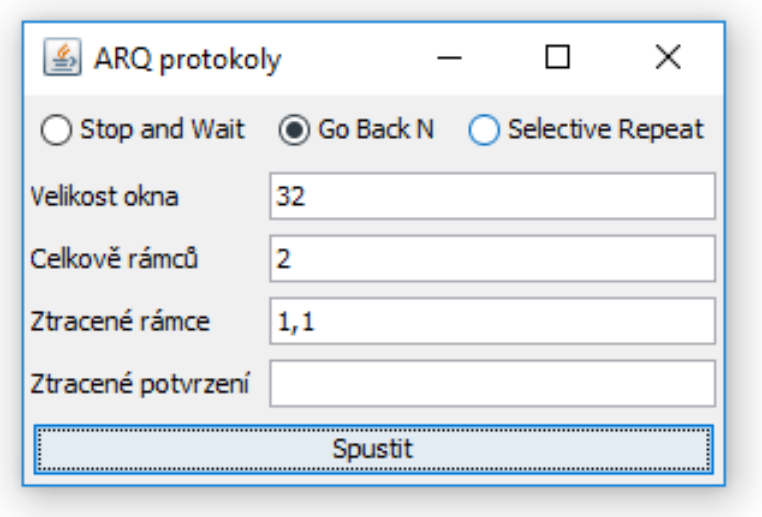
ARQ – Automatic Repeat reQuest

Problémy poškozených a ztracených rámců řeší ARQ (Automatic Repeat reQuest) protokoly. ARQ zavádí principy potvrzování přijatých dat. V případě, že vysílač neobdrží potvrzení na zaslaný rámec po dobu stanoveného časového úseku, vyšle tento rámec znovu, přičemž má vysílač předdefinováno, kolikrát se nepotvrzený rámec může pokusit znovu odeslat. ARQ protokoly slouží i k řešení situace, kdy je doručen poškozený rámce. V tomto případě příjemce zareaguje odesláním rámce negativního potvrzení a očekává, že mu vysílač odešle poškozený rámec znovu. Na úrovni spojové vrstvy máme tři ARQ protokoly, kterými se typicky řeší zmíněné chybové situace: Stop-and-wait, Go-Back-N, Selective-repeat.

A.2 Aplikace znázorňující scénáře

Součástí této laboratorní úlohy je aplikace, ve které lze nastavit popis scénáře a zvolit ARQ protokol, kterým má být tento scénář vykonán. Aplikace daný scénář vykoná a jako výstup vytvoří v adresáři, ve kterém se nachází, obrázek sekvenčního diagramu, který popisuje uskutečněnou komunikaci mezi stranami vysílače a příjemce.

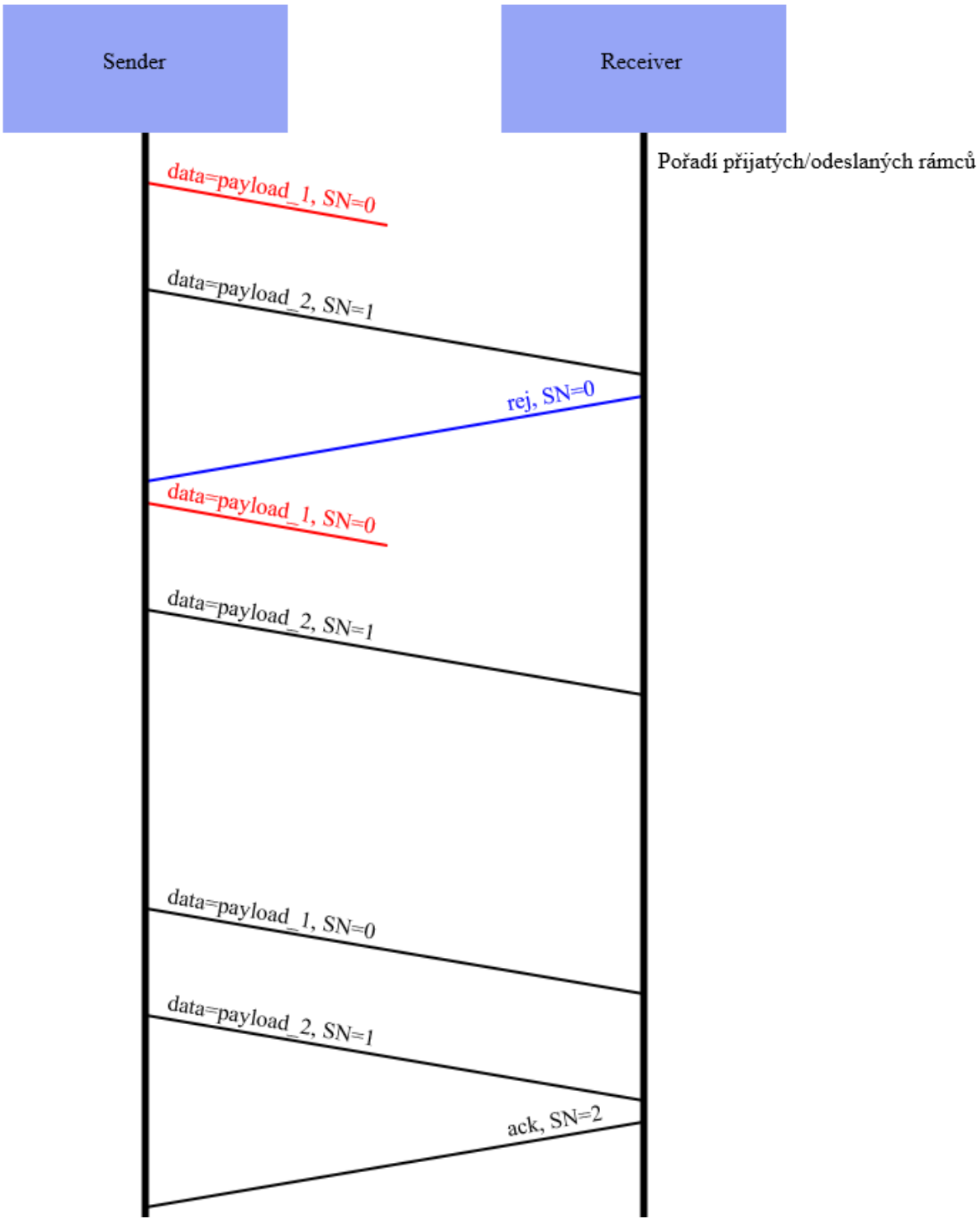
Aplikace je zobrazena na obrázku A.1. Uživatelské rozhraní disponuje možností zvolit si typ ARQ protokolu – tato volba se nastavuje v horní části. Dále lze nastavit velikost přenosového okna (u protokolu Stop-and-Wait je tato volba vypnuta), počet datových rámců k odeslání, specifikovat které z těchto rámců se mají po cestě ztratit a která potvrzení mají být po cestě ztracena.



Obr. A.1: Příklad nastavení scénaře v uživatelském rozhraní aplikace.

Nastavení v horní části obrázku A.1 značí, že scénář bude proveden protokolem Go Back N. Velikost přenosového okna je 32 – hypoteticky může odesílatel odeslat 32 datových rámců bez nutnosti čekání na potvrzení. Celkově odesílatel odešle 2 datové rámce. Po cestě se dvakrát ztratí první datový rámec (rámce, které mají být ztraceny se oddělují čárkou a mohou se opakovat). Stejným způsobem se zapisují ztracená potvrzení, která zde však v tomto případě nejsou.

Obrázek A.2 obsahuje výstup aplikace po spuštění výše uvedeného scénáře. Na obrázku je vidět, že první rámec *payload_1* byl dvakrát ztracen dle konfigurace. Přijímač správně zareagoval na první přijatý rámec *payload_2* chybou. Odesílatel na přijetí tohoto zamítnutí zareagoval opětovným odesláním obou rámců. Tentokrát přijímač druhý datový rámec automaticky zahodil, jelikož stále čeká na první rámec *payload_1*. Mezitím na straně odesílatele dochází k vypršení časovače a odešlou se napotřetí oba z doposud nepotvrzených rámců. Tentokrát už jsou oba správně doručeny k přijímači, který jejich příjem potvrdí a tím celý scénář končí.



Obr. A.2: Výstup aplikace ve formě sekvenčního diagramu po spuštění scénáře z obrázku A.1

.

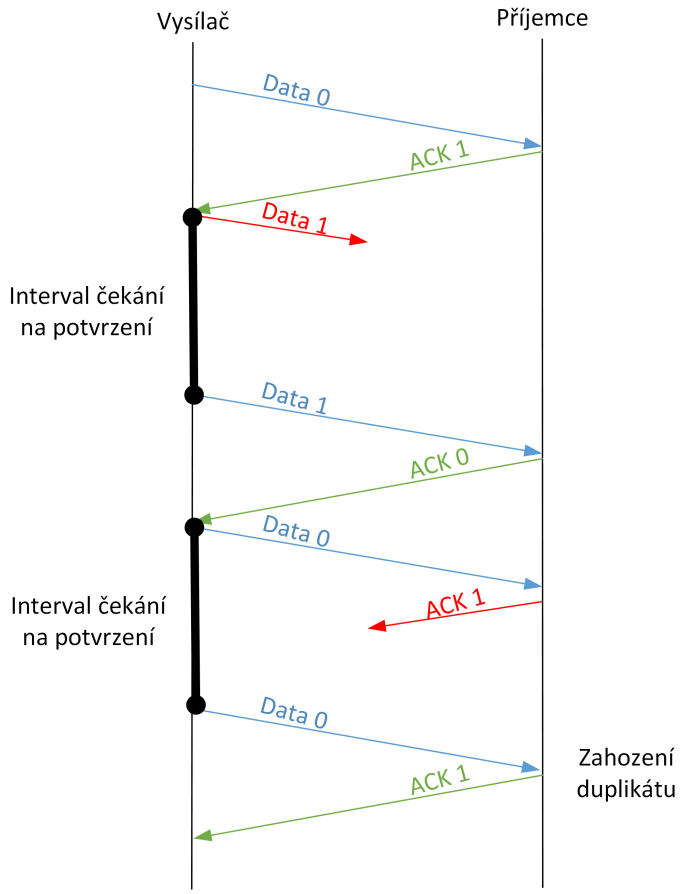
A.3 Stop-and-Wait

Nejjednodušší realizací ARQ je protokol Stop-and-Wait. Vysílací strana odešle rámec obsahující kontrolní součet a čeká na potvrzení, že byl přijat. Pokud je rámec doručen v pořádku až k příjemci, vyšle odesílateli zprávu potvrzující bezchybné obdržení rámce, čímž přijímací strana zároveň deklaruje, že je připravena na obdržení další zprávy. Jakmile vysílač obdrží potvrzení, pokračuje stejným způsobem odesláním dalšího rámce. Kromě schopnosti vypořádat se s poškozenými či ztracenými rámci může přijímač do jisté míry regulovat rychlost zasílání dat jednoduše tím, že vyčká s odesláním potvrzení.

Mimo standardní chybové případy s poškozením nebo ztracením vysílaného rámce může nastat situace, kdy dojde ke ztrátě nebo poškození potvrzovacího rámce. V takovém případě vysílač čeká do vypršení stanoveného časového úseku a opakuje odeslání původního rámce, jako kdyby došlo k jeho ztracení. Přijímač tedy znovu obdrží stejný rámec a aby se zabránilo opakovanému zpracování již zpracovaného rámce, jsou datové i potvrzující rámce číslovány jako 0 a 1. Zpráva `ACK1` znamená, že přijímač obdržel rámec s označním 0 a naopak.

Příklad komunikace s využitím protokolu Stop-and-Wait je znázorněn na obrázku A.3. Každá hrana mezi časovými osami komunikujících stran vyjadřuje odeslání jednoho rámce. První dvojice zpráv mezi vysílačem a příjemcem ilustrují bezchybnou komunikaci na obou stranách. Třetí rámec byl ztracen, takže vysílač musel počkat do vypršení časovače a následně poslal rámec znovu, tentokrát už úspěšně. Později je na obrázku popsán i případ, kdy se ztratí potvrzení o přijetí *ACK 1*. Vysílač znovu čeká na skončení časového intervalu, po jehož dobu čeká na potvrzení, a po té znovu odešle rámec *Data 0*. Příjemce detekuje duplicitní rámec, který zahodí, a pošle zpět potrvzení o přijetí.

Jak u protokolu Stop-and-Wait, tak i u pokročilejších ARQ protokolů je především důležité všimnout si a pochopit, jakým způsobem se v popsaném příkladě pracuje se sekvenčními čísly (čísla za označením typu rámce Data/ACK). Obdržená sekvenční čísla řídí chování dané obou komunikačních stran.

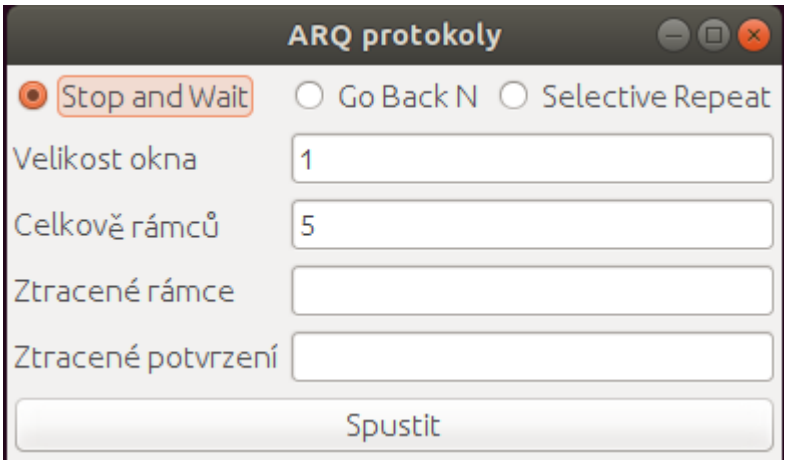


Obr. A.3: Ukázka fungování protokolu Stop-and-Wait a jeho schopností vypořádat se se ztracenými datovými rámci odesílatele i potvrzovacími rámci příjemce.

A.3.1 Laboratorní úkoly k protokolu Stop-and-Wait

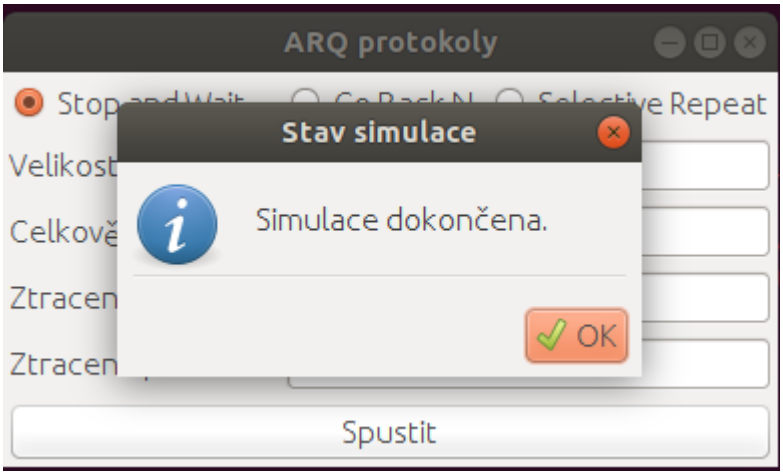
Laboratorní úloha bude realizována uvnitř připraveného virtuálního prostředí **ARQ protokoly**, které lze spustit v programu **VirtualBox**.

1. Nabootujte se do virtuálního prostředí s použitím přihlašovacích údajů *guest/guest*.
2. Otevřete terminál (CTRL + ALT + T) a spusťte referenční aplikaci (*java -jar arq-protokoly.jar*).
3. Po stvrzení příkazu se vám otevře okno aplikace s vyplněnými výchozími hodnotami, jako na obrázku A.4.



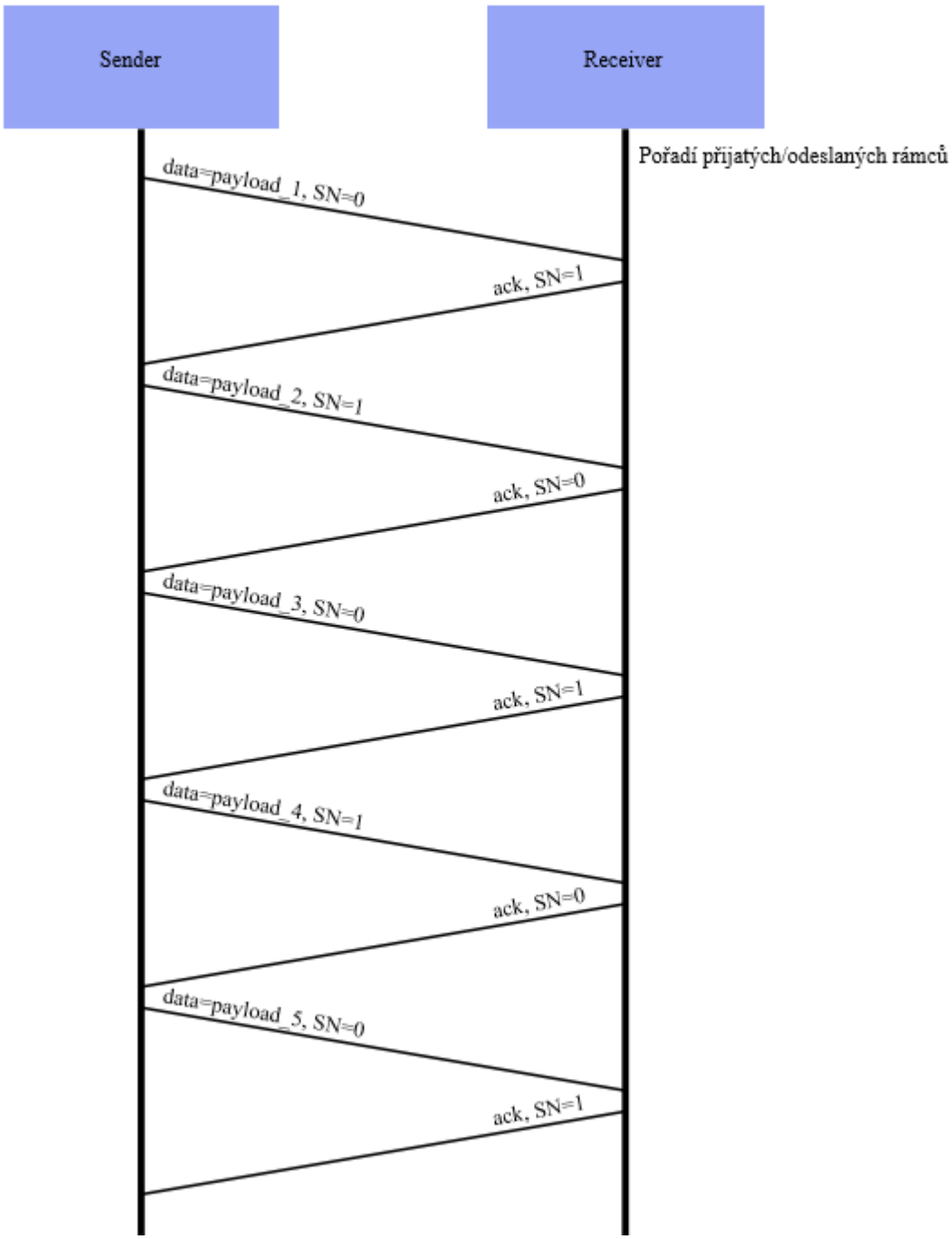
Obr. A.4: Okno aplikace s výchozím nastavením hodnot.

4. Pomocí tlačítka *Spustit* spusťte výchozí scénář. Aplikace zpracuje zadanou situaci a jakmile skončí, vyskočí okno s potvrzením, že scénář byl dokončen (viz. obrázek A.5).



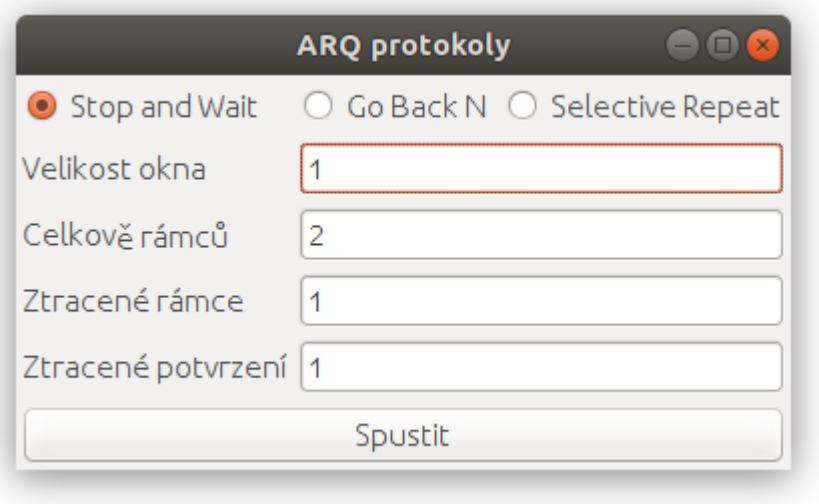
Obr. A.5: Okno potvrzující konec simulace.

5. Otevřete webový prohlížeč **Firefox** a zadejte do něj adresu `file:///home/guest/test.svg`. Po přechodu na stránku se vám vykreslí výsledný sekvenční diagram, jako na obrázku A.6. Všimněte si, že hodnoty sekvenčních čísel jsou pouze 0 a 1. Všimněte si také, jakým způsobem se jejich hodnoty mění. **Otázka 1: Zdůvodněte, proč se sekvenční čísla mezi datovými a potvrzovacími rámci na obrázku A.6 střídají.**



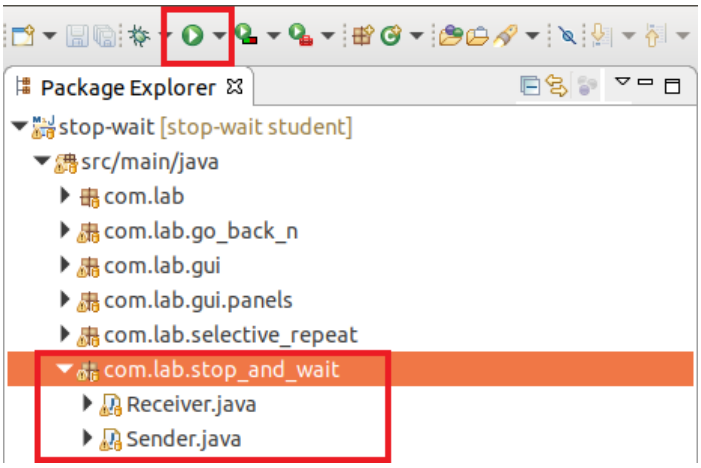
Obr. A.6: Sekvenční diagram jako výstup výchozího scénáře.

6. Na dalším příkladu si ukážeme schopnosti protokolu Stop-and-Wait zotavit se z chybových stavů. Nakonfigurujte si scénář podle obrázku A.7 a spusťte jej. Konfigurace reflektuje stav, kdy má jednou dojít ke ztrátě prvního datového rámce a jednou ke ztrátě potvrzovacího rámce na první datový rámec.



Obr. A.7: Zadání scénáře s protokolem Stop-and-Wait, při kterém dojde ke ztrátě prvního datového a potvrzovacího rámce.

7. Aktualizujte stránku webového prohlížeče klávesou F5 a prohlédněte si výsledný diagram popisující uskutečněnou komunikaci. Všimněte si, jakým způsobem si protokol poradil se ztrátou obout typů rámců. **Otázka 2: Proč vysílač odeslal třikrát za sebou stejný datový rámec 'payload_1'?**
8. Dle potřeby si zkuste nakonfigurovat další scénáře.
9. Otevřete si další terminál a spusťte vývojové prostředí Eclipse příkazem `./eclipse/java-2019-03/eclipse/eclipse`
10. Až se vám otevře vývojové prostředí, otevřete si třídy `Receiver` a `Sender` uvnitř balíčku `com.lab.stop_and_wait` viz. obrázek A.8.



Obr. A.8: Obsah projektu. Červený čtvereček ve vrchní části obrázku ohraničuje tlačítko, kterým se spouští vaše aplikace. Červené ohraničení ve spodní části označuje třídy, které budete modifikovat.

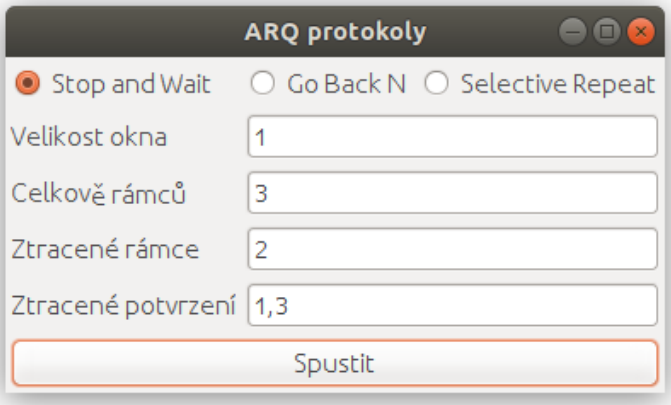
11. Projděte si kód těchto dvou tříd (nejprve `Sender`) a seznamte se s komentovanou implementací každé ze tříd. Úseky, do kterých budete doplňovat implementaci jsou označeny speciálním blokovým komentářem, jako na obrázku A.9.

```
// Rámec byl odeslán -> počkáme si na potvrzení
final TransmissionUnit result = trafficController.readInput(in);

/*******************************    TODO: KOD K DOPLNENI    ************************
 *
 * Zkontrolujte, zda přijatý rámec v proměnné 'result', je potvrzovacím rámcem, na který
 *  čekáme. Do proměnné 'correctAck' přiřaďte výsledek této kontroly.
 *
 ******************************************************************************************/
final boolean correctAck = true;
```

Obr. A.9: Ukázka označení části kódu, který má být doplněn.

12. Jakmile se seznámíte s implementací, doplňte kód do označených částí a spusťte si vaše řešení pomocí tlačítka označeného ve vrchní části obrázku A.8. Nakonfigurujte si ve své aplikaci scénář jako na obrázku A.10 a spusťte simulaci.



Obr. A.10: Konfigurace scénáře k ověření správnosti implementace.

13. Výstup si můžete zobrazit přímo v prostředí Eclipse otevřením vzniklého souboru `test.svg` nebo změnou adresy ve webovém prohlížeči na: `file:///home/guest/lab/stop-wait/test.svg`
14. Konečný diagram přejmenujte na '`output_stop_and_wait.svg`' a vložte do vašeho protokolu s odpověďmi.

A.3.2 Seřazené odpovědi na otázky dle výskytu.

Otázka 1:
Zdůvodněte, proč se sekvenční čísla mezi datovými a potvrzovacími rámci na obrázku A.6 střídají.
Odpověď:
Vysílač zahajuje komunikaci a volí hodnoty sekvenčních čísel svých datových rámců. Jakmile je rámec s tímto sekvenčním číslem přijat na straně přijímače, odpoví zasláním potvrzovacího rámce se sekvenčním číslem rámce, který očekává jako další v pořadí. Jedná se tedy jak o potvrzení přijatého datového rámce, tak i o sdělení připravenosti k příjmu dalšího rámce. Z toho důvodu bude při velikosti přenosového okna 1 a úspěšně přijatém datovém rámci vždy následovat potvrzovací rámec s opačnou hodnotou sekvenčního čísla, než měl původní rámec datový.

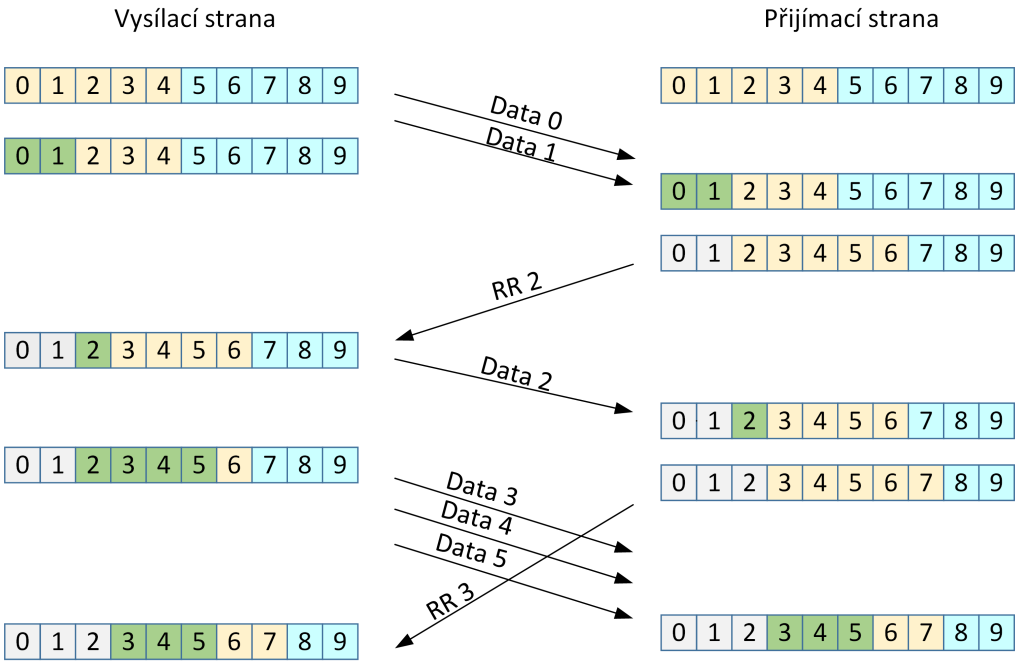
Obecně pak platí, že příjemce díky tomuto mechanismu může snadno rozlišit přijetí duplicitního rámce.

Otázka 2:
Proč vysílač odeslal třikrát za sebou stejný datový rámec 'payload_1'?
Odpověď:
Při prvním pokusu o odeslání tento rámec nedorazil k příjemci. Při druhém pokusu již doručení bylo úspěšné, ale tentokrát se ztratilo potvrzení o přijetí. Vysílač nedostal informaci o tom, že byl rámec 'payload_1' doručen, tak po vypršení časovače přenos tohoto rámce zopakoval.

A.4 Posuvné okno

Slabina Stop-and-Wait mechanismu je, že dokáže najednou přenášet pouze jeden rámec, což ve většině případů omezuje efektivitu přenosu. Logickým krokem při řešení tohoto problému by bylo odesílat více rámců bez nutnosti čekání na potvrzení mezi nimy. S touto modifikací vzniká potřeba jednotlivé rámce číslovat, aby vysílací stanice měla přehled o potvrzených rámcích.

V praxi to funguje tak, že přijímací stanice prohlásí, že je schopna najednou přijmout x rámců. V tu chvíli může vysílací stanice odeslat x rámců, přičemž příjímač odesílá potvrzení obsahující číslo dalšího očekávaného rámce (podobně jak u stop-and-wait). Jakmile vysílač obdrží potvrzení s číslem očekávaného rámce a, je to pro něj signálem, že stanice na druhé straně může přijmout dalších x rámců, počínaje rámcem s číslem a. Aby tento mechanismus fungoval, je potřeba, aby si obě strany udržovaly v paměti rámce, které mohou v daný okamžik odeslat (vysílač) a přijmout (přijímač). Z uvedeného popisu vyplývá, že s využitím techniky posuvného okna může přijímač přijmout např. rámce a, b, c, d s tím, že potvrzení odešle až po přijetí posledního rámce d, čímž značně zefektivní průtok dat sítí.



Obr. A.11: Mechanismus posuvného okna s velikostí okna 5, přičemž odesílatel chce odeslat 9 rámců.

Obrázek A.11 ilustruje tento mechanismus na příkladu s maximální velikostí okna 5. Žluté čtverečky symbolizují rámce, které jsou aktuálně uvnitř okna, zelené

symbolizují odeslané/přijaté rámce, šedé jsou potvrzené a modré jsou rámce, které budou postupně naskakovat do okna, jakmile se uvolní místo. Zpráva RR (Receive Ready) je významově podobná zprávám ACK u algoritmu stop-and-wait.

A.5 Go-Back-N

Go-Back-N (GBN) je jednodušším typem implementace mechanismu posuvného okna. K potvrzování přijatých rámců se typicky používá zpráva ACK. V případě, že přijímací strana obdrží poškozený rámec, odešle vysílači záporné potvrzení – typicky se jedná o zprávu REJ (Reject) nebo NAK (Negative ACK) – a zahodí tento i další po něm následující pakety bez ohledu na to, jestli byly přijaté bezchybně. Z toho důvodu je vysílač po přijetí záporného potvrzení nucen odeslat znovu jak poškozený rámec, tak i všechny po něm následující. Jakmile přijímač obdrží neporušenou verzi dříve poškozeného rámce, přestane zahazovat rámce a pokračuje v klasickém chování.

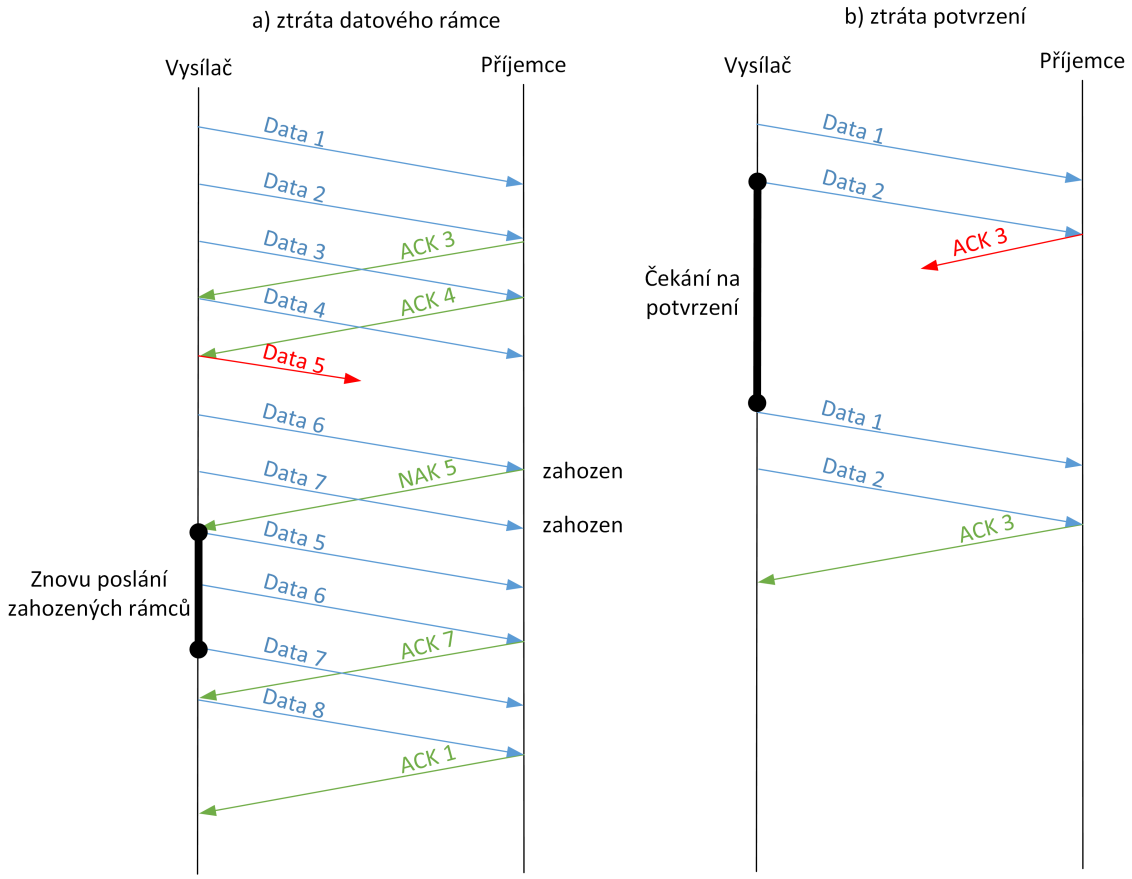
V případě hustého výskytu chyb se nejedná o optimální řešení, jelikož dochází k masivnímu plýtvání přenosového pásma. Správně nastavený protokol GBN, včetně časovačů, nicméně zvládne vyřešit následující chybové stavy (dále uvažujeme, že stanice A je vysílač a strana B je přijímač):

1. **Ztracení odeslaného rámce**
   - stanice A odesílá za sebou rámce 1 a 2. První z rámců, 1, byl po cestě ztracen, a tak doputoval do cíle pouze rámec 2. Stanice B nyní přijala rámec 2, ale jelikož očekává rámec číslo 1, tak jej zahodí a pošle stanici A zprávu *NAK 1* (stanice B zahazuje další rámce do té doby, dokud neobdrží chybějící rámec). Stanice A obdrží zprávu *NAK 1* a zjistí tak, že se rámec číslo 1 po cestě ztratil, tak jej odešle znovu (opakovaně odeslaný rámec 1 je následován rámcem 2, jelikož stanice A ví, že byl zahozen).
   - pokud by byl stanicí A odeslán pouze rámec 1 bez následníka, stanice B by neměla jak zjistit, že došlo ke ztrátě rámce. V takovém případě stanici A pomůže časový interval, po jehož dobu stanice čeká na potvrzení od stanice B. Pokud po skončení intervalu potvrzení nedorazí, stanice A pošle rámec 1 znovu.
2. **Poškození odeslaného rámce** - pokud stanice B obdrží poškozený rámec, chová se velmi podobně jako při ztrátě očekávaného rámce – stanici A je odeslána zpráva NAK s číslem rámce, který je očekáván. Následující rámce (kromě očekávaného) jsou zahazovány a nepotvrzují se.
3. **Ztráta potvrzení** - pokud přenos rámců ze stanice A do stanice B proběhne v pořádku, ale potvrzení je poškozeno nebo bylo ztraceno po cestě, stanice B

čeká do uplynutí časového intervalu, po jehož dobu očekává, že dojde potvrzení. Jakmile překročí zmíněný práh časovače, odešle znovu všechny rámce, pro které neobdržel potvrzení. Podobně by se řešilo chování při ztrátě zprávy negativního potvrzení.

Obrázek A.12 popisuje schopnost algoritmu GBN vypořádat se s chybovými stavy. V levém diagramu je popsán případ, kdy dojde ke ztrátě datového rámce při přenosu k příjemci. Ve chvíli kdy přijímač zjistí, že místo očekávaného rámce 5 dorazil rámec 6, odešle vysílači zprávu NAK 5, a zahazuje další přijaté rámce. Vysílač obdrží tuto zprávu a zareaguje opětovným vysláním ztraceného rámce 5 a všech následujících rámců.

Na pravé straně je popsán naopak stav, kdy dojde ke ztrátě potvrzovací zprávy. Ve scénáři se předpokládá, že chce vysílací stanice odeslat pouze dva datové rámce. Ty jsou k příjemci přeneseny bezchybně, avšak potvrzovací zpráva je po cestě ztracena. V tomto případě se vysílač bez potvrzení nemá jak dozvědět, že data doputovala k příjemci, a tak je odkázán na čekání. Jakmile je překročen práh intervalu, po jehož dobu se čeká na potvrzení, vysílač opakovaně odešle oba datové rámce a znovu čeká na potvrzení. To nyní bezchybně doputovalo k vysílači.

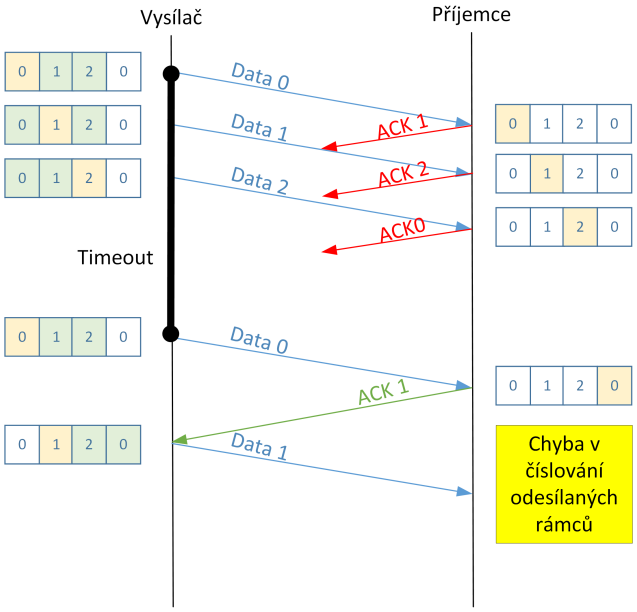


Obr. A.12: Ukázka schnoposti algoritmu Go-Back-N zotavit se ze stavu ztraceného datového rámce v levé části a ze ztráty potvrzovacího rámce v pravé části obrázku.

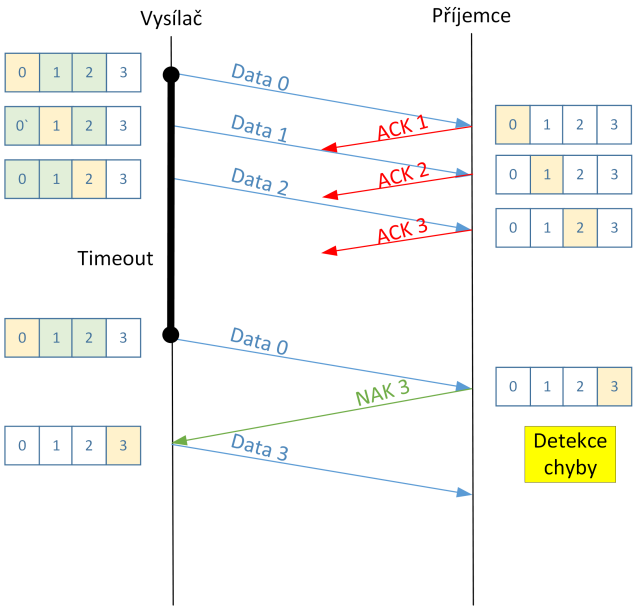
Velikost okna u Go-back-N

Pro správné fungování protokolu GBN je potřeba zvolit vhodnou velikosti posuvného okna. Okno příjemce má velikost $\mathbf{W = 1}$, jelikož GBN nepřipouští přijem jakéhokoliv rámce mimo očekávé pořadí. Na straně vysílače je velikost okna $\mathbf{W < N}$, kde N je počet sekvenčních čísel.

Situace je vysvětlena na obrázku A.13, kde je v horní polovině obrázku ukázána situace pro stejnou velikost okna, jako je počet sekvenčních čísel. Vysílač odesílá tři rámce, které přijímač v pořádku obdrží, ale potvrzení jsou po cestě ztracena. Vysílač tedy opětovně pošle první rámec. Příjemce očekává rámec se stejným sekvenčním číslem, a proto nezjistí, že daný rámec už jednou přijal a chová se, jako by se nic nestalo. Nedošlo tedy ke správné reakci na chybovou situaci. V situaci ve spodní polovině obrázku je velikost okna větší než počet sekvenčních čísel, takže příjemce zareaguje příslušnou zprávou *NAK 3*. Synchronizace stran tedy není narušena.



a) W = počet sekvenčních čísel



b) W < počet sekvenčních čísel

Obr. A.13: Go-back-N při velikostech posuvného okna: a) velikost okna je stejná jako počet sekvenčních čísel, b) velikost okna je menší než počet sekvenčních čísel.

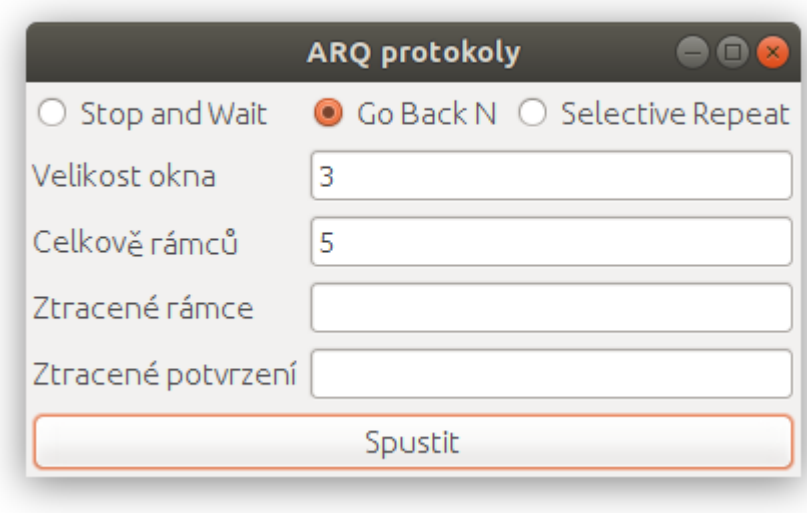
A.5.1 Laboratorní úkoly k protokolu Go Back N

Rozdíl oproti protokolu Stop-and-Wait je zde především v tom, že protokol Go Back N potvrzuje rámce kumulativně. Z toho důvodu se může stát (a u složitějších scénářů

se vám pravděpodobně i stane), že při opakovaném spuštění stejného scénáře se bude diagram na výstupu mírně lišit – vždy by však měl být výstup korektní podle definice Go Back N protokolu.

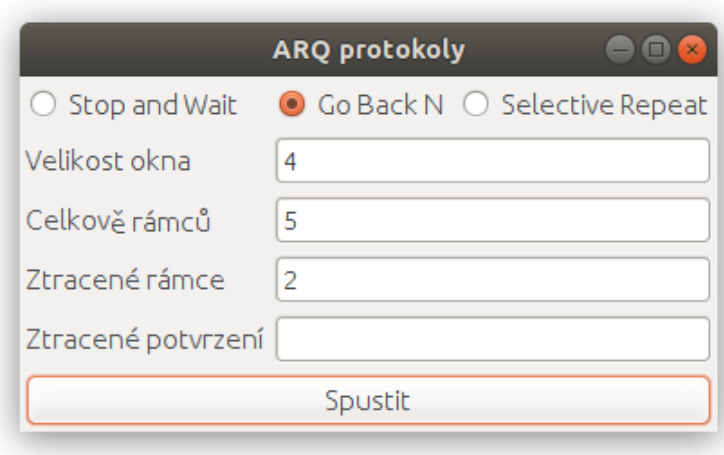
Při realizaci následujících úkolů pokračujte v rozdělaném prostředí z úkolů k protokolu Stop-and-Wait.

1. Pokud nemáte, spusťte si znovu referenční aplikaci (java -jar arq-protokoly.jar) a spusťte si základní scénář z obrázku A.14.



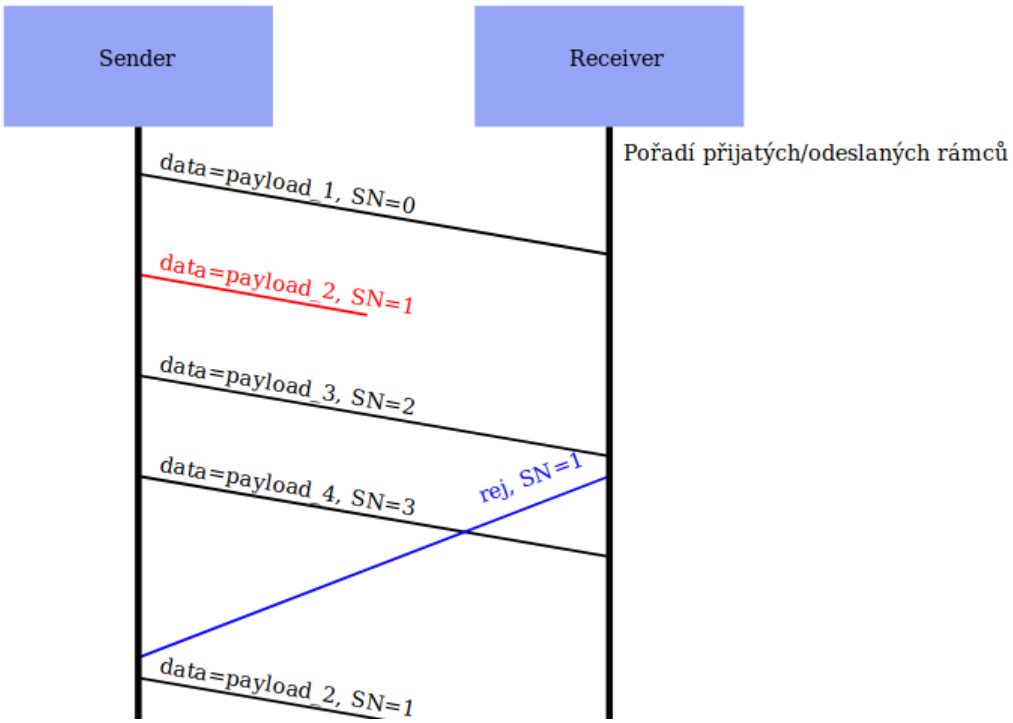
Obr. A.14: Základní scénář protokolu Go Back N.

2. Diagram přejmenujte na 'output_go_back_n_1.svg' a vložte do vašeho protokolu.
   - **Otázka 1: Zdůvodněte, proč v diagramu nemá každý datový rámec své potvrzení, jako při Stop-and-Wait.**
   - **Otázka 2: Jsou hodnoty sekvenčních čísel v souladu s definicí Go Back N? Zdůvodněte.**
3. Číslování rámců a způsob jejich potvrzování si zkuste na dalších scénářích tak, že budete experimentálně měnit velikost okna a počet rámců k odeslání. *Např. velikost okna 5 a celkově rámců 25.*
4. Nyní se podíváme, jak protokol Go Back N vyřeší problém ztraceného datového rámce. Spusťte si scénář z obrázku A.15 a aktualizujte diagram.



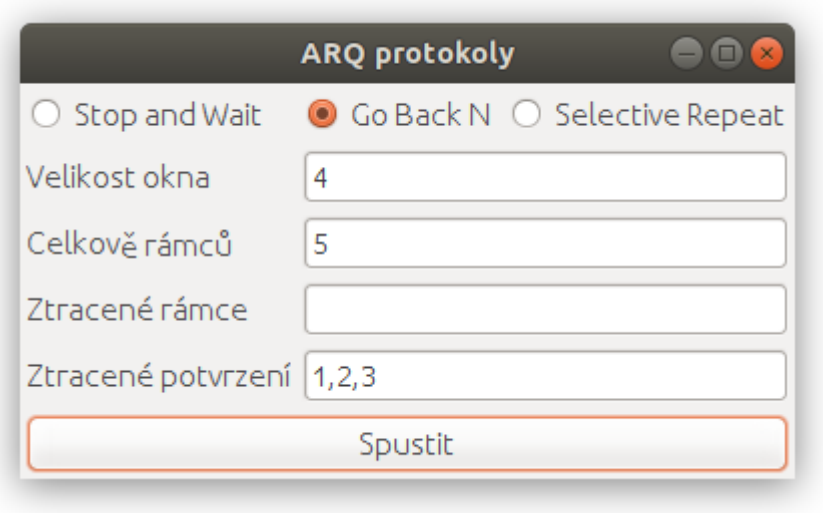
Obr. A.15: Scénář protokolu Go Back N se ztrátou datového rámce.

5. Na vygenerovaném diagramu si všimněte především jeho vrchní části (viz. obrázek A.16), kde došlo ke ztrátě datového rámce a reakci přijímače na tuto událost.
   - **Otázka 3: Proč nedošlo k odeslání potvrzovacího rámce na v pořádku přijatý datový rámec 'payload_3'?**
   - **Otázka 4: Proč má rámec negativního potvrzení hodnotu SN=1?**



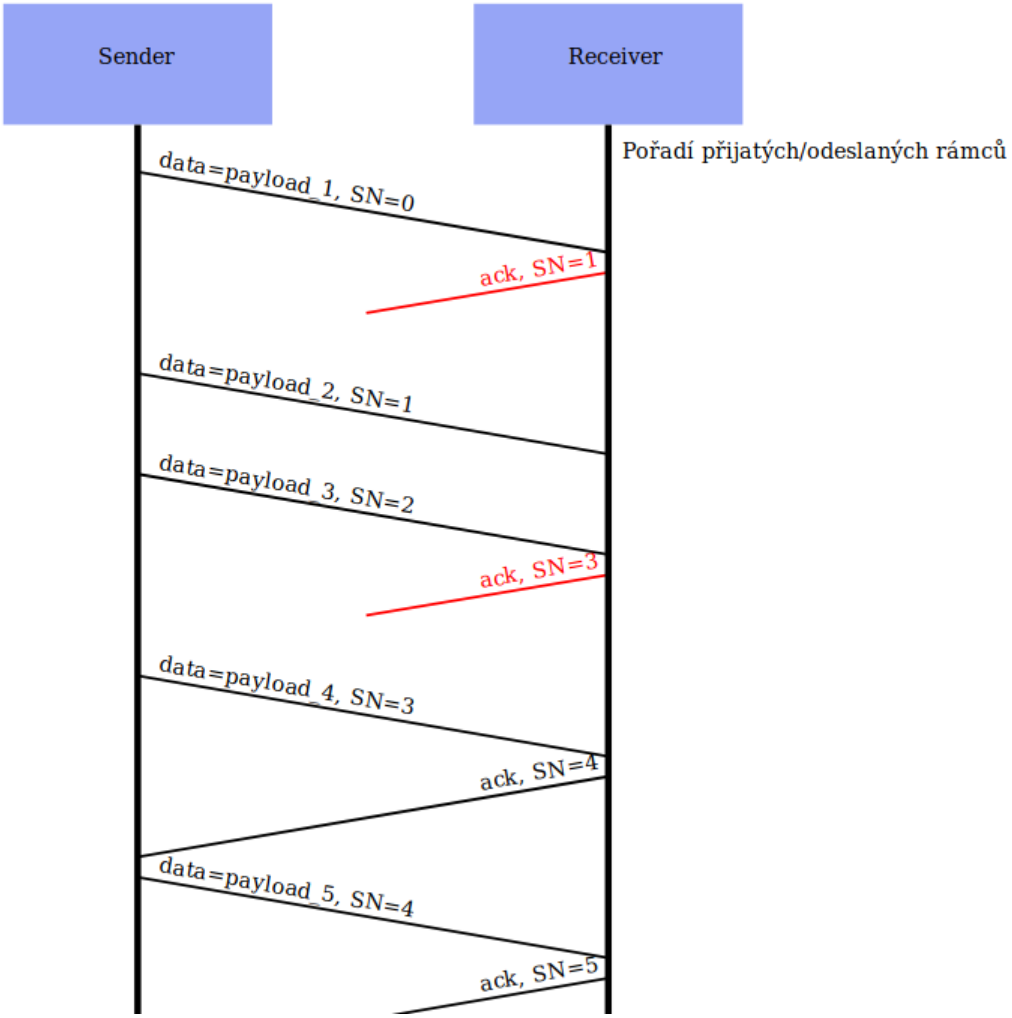
Obr. A.16: Výstup důležité části diagramu ze scénáře s protokolem Go Back N a ztrátou datového rámce.

6. V dalším kroce se podíváme na případy, kdy dojde ke ztrátě potvrzovacího rámce. Příklad takového scénáře je naznačen na obrázku A.17. V nastavení ztracených potvrzení jsou záměrně hned tři rámce, jelikož z důvodu kumulativního potvrzení netušíme, který z potvrzovacích rámců bude stranou příjímače odeslán.



Obr. A.17: Scénář protokolu Go Back N se ztrátou potvrzovacích rámců.

7. Příklad výstupu je ukázán na obrázku A.18. Všimněte si, že jak bylo avizováno, potvrzení na druhý datový rámec nemohlo být ztraceno, jelikož nebylo přijímačem ani odesláno. Stále však došlo ke ztrátě celkově dvou potvrzovacích rámců se sekvenčními čísly SN=1 a SN=3.
   - **Otázka 5: Z jakého důvodu nedošlo v ukázaném výstupu na obrázku A.18 k opakovanému odeslání žádného datového rámce?**
   - **Otázka 6: Vycházejme z obrázku A.18. Co by se stalo v případě, že by potvrzovací rámec se sekvenčním číslem SN=4 byl ztracen?**
   - **Otázka 7: Vycházejme z obrázku A.18. Co by se stalo v případě, že by potvrzovací rámec se sekvenčním číslem SN=5 byl ztracen?**



Obr. A.18: Diagram se ztracenými potvrzeními u protokolu Go Back N.

8. Volitelně si vyzkoušejte další scénáře s různými kombinacemi parametrů.
9. Pokud již nemáte, otevřete si znovu vývojové prostředí Eclipse:
   `./eclipse/java-2019-03/eclipse/eclips`
10. V této části se zaměříme na implementaci protokolu Go Back N. Otevřete si nejprve balíček `com.lab` a v něm umístěnou třídu `ProtocolFactory`, která se stará o spouštění protokolů.
11. Seznamte se s kódem této třídy a následně vyhledejte metodu `initGoBackN`. Na obrázku A.19, je vyznačen úsek kódu, který se stará o vhodnou volbu maximálního sekvenčního čísla pro daný scénář. Dle poznámek doplňte tuto chybějící implementaci.

```
private void initGoBackN(TraceBuilder tracer, Integer numberOfFrames,
        List<Integer> framesToDrop, List<Integer> acksToDrop,
        Integer winSize) throws IOException {

/*******************************    TODO: KOD K DOPLNENI    ************************
 *
 * Do proměnné 'maxSeqNumber' přiřaďte hodnotu maximálního sekvenčního čísla v souladu
 *  s principy protokolu Go Back N.
 *
 ******************************************************************************************/
int maxSeqNumber = 1; // TODO: Zde změňte
```

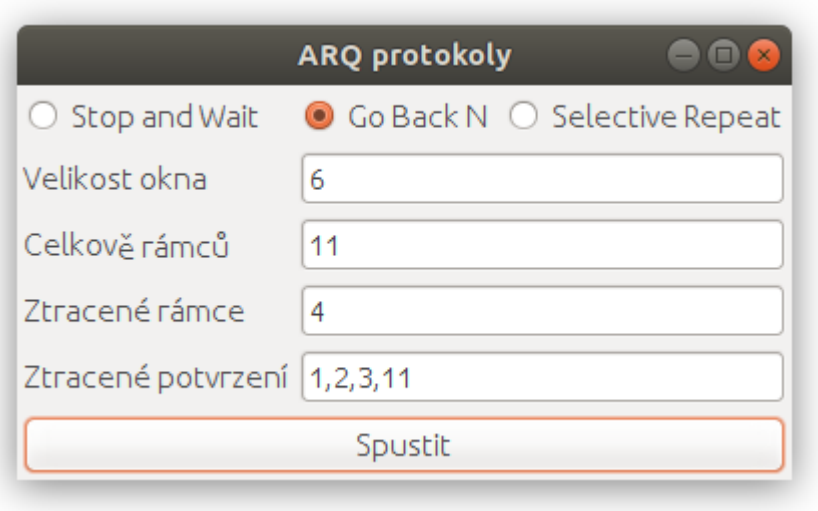
Obr. A.19: Ukázka metody `initBackN`.

12. Dále si otevřete balíček `com.lab.go_back_n` a třídy `Receiver` a `Sender`, jako v předchozím cvičení a projděte si komentovaný kód.
    - Nejprve se zaměřte na implementaci metody `ackSequenceNumber` ve třídě `Receiver` (viz. obrázek A.20). Úkolem této metody je inkrementovat hodnotu předaného sekvenčního čísla s ohledem na jeho maximální povolenou hodnotu a vrátit výsledek operace. Dle poznámek v kódu naimplementujte tělo této metody.

```
private int ackSequenceNumber(int currentSN) {
    /*******************************    TODO: KOD K DOPLNENI    ************************
     *
     * Implementujte tělo této metody, která bude vracet inkrementovanou hodnotu proměnné
     *  'currentSN' s ohledem na strop určený maximální hodnotou sekvenčního čísla.
     *
     * V této části by vám mohla pomoci proměnná 'firstOutOfRangeSN'.
     *
     *  Např:
     *      currentSN = 5
     *      maximální hodnota sekvenčního čísla = 5
     *      výsledek = 6
     ******************************************************************************************/
    return 0; // TODO: zde vraťte požadovanou hodnotu inkrementovaného sekvenčního čísla
}
```

Obr. A.20: Ukázka metody `ackSequenceNumber`.

    - Dalším a posledním krokem implementace bude zařídit, aby správným způsobem docházelo na straně přijímače k zahazování přijatých datových rámců, po přijetí neočekávaného rámce. Vyhledejte další dva bloky indikující místa doplnění implementace na řádcích 190 a 213 ve třídě `Receiver` a doplňte chybějící přiřazení.
13. Jakmile doplníte implementaci na zmíněná místa, spusťte si vaše řešení (viz. obrázek A.8). Ve vašem řešení nastavte scénář z obrázku A.21 a spusťte jej.



Obr. A.21: Scénář protokolu Go Back N pro ověření správnosti implementace.

14. Výsledný diagram pojmenujte '`output_go_back_n_impl.svg`' a vložte do svého protokolu.

A.5.2 Seřazené odpovědi na otázky dle výskytu.

Otázka 1:
Zdůvodněte, proč v diagramu nemá každý datový rámec své potvrzení, jako při Stop-and-Wait.
Odpověď:
Go Back N potvrzuje rámce kumulativně což znamená, že pokud je přijímači doručeno v krátký okamžik více rámců, nebude posílat potvrzení na každý z nich, ale odešle pouze potvrzení na ten poslední přijatý, čímž potvrdí příjem všech předešlých.

Otázka 2:
Jsou hodnoty sekvenčních čísel v souladu s definicí Go Back N? Zdůvodněte.
Odpověď:
Ano jsou v souladu. Velikost přenosového okna je W=3, z definice víme, že velikost okna musí být menší než počet sekvenčních čísel. Poslední datový rámec má sekvenční číslo SN=4, takže tato podmínka byla splněna.

Otázka 3:
Proč nedošlo k odeslání potvrzovacího rámce na v pořádku přijatý datový rámec 'payload_3'?
Odpověď:
Protože jakmile přijímač odešle rámec negativního potvrzení, zahazuje všechny další příchozí rámce až do doby, než je doručen očekávaný datový rámec.

Otázka 4:
Proč má rámec negativního potvrzení hodnotu SN=1?
Odpověď:
Přijímač očekával doručení rámce s hodnotou sekvenčního čísla SN=1, ale místo toho mu byl doručen rámec SN=2, který je až další v pořadí. Přijímač si z tohoto stavu odvodí, že došlo ke ztrátě rámce SN=1, a proto odešle rámec negativního potvrzení s hodnotou sekvenčního čísla SN=1, jelikož to je rámec, který očekává. Mimo jiné tímto přijímač potvrdil přijetí rámce SN=0.

Otázka 5:
Z jakého důvodu nedošlo v ukázaném výstupu na obrázku A.18 k opakovanému odeslání žádného datového rámce?
Odpověď:
Ačkoliv byly ztraceny dva potvrzovací rámce (SN=1 a SN=3), velikost přenosového

okna je W=4 a poslední potvrzovací rámec v rozsahu tohoto okna byl v pořádku doručen vysílači. Z pohledu vysílače to tedy vypadá, jakoby přijímač potvrdil všechny 4 datové rámce na jednou, bez mezipotvrzování.

Otázka 6:
Vycházejme z obrázku A.18. Co by se stalo v případě, že by potvrzovací rámec se sekvenčním číslem SN=4 byl ztracen?
Odpověď:
V takovém případě by vysílač neobdržel potvrzovací rámec na žádný z odeslaných datových rámců ('payload_1' - 'payload_4'), takže by počkal na skončení časovače a po té odeslal všechny rámce znovu.

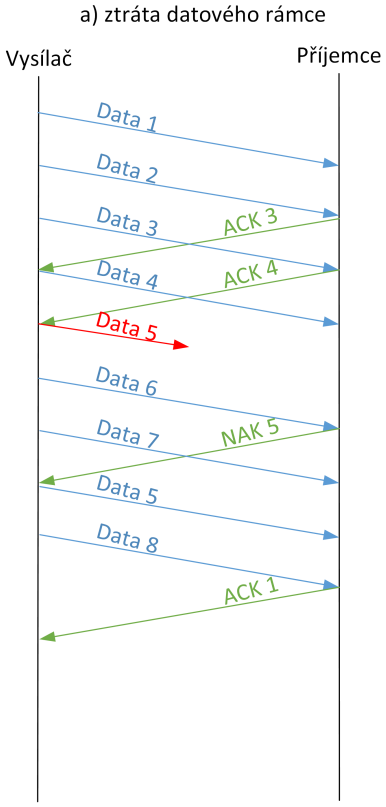
Otázka 7:
Vycházejme z obrázku A.18. Co by se stalo v případě, že by potvrzovací rámec se sekvenčním číslem SN=5 byl ztracen?
Odpověď:
První čtyři datové rámce by byly doručeny a potvrzeny přijímačem. Poslední datový rámec 'payload_5', by však musel být odeslán opakovaně.

A.6 Selective-Repeat

Protokol Selective-repeat (SR) je alternativní implementací posuvného okna. Na rozdíl od GBN se opakovaně posílá pouze rámec, jehož potvrzení nedošlo v očekávaném časovém intervalu, nebo byl odmítnut zprávou NAK/REJ. Výhodou mechanismu SR je, že eliminuje opětovné posílání rámců, které jinak doputovaly k cíli bezchybně, akorát ve špatném pořadí. Na druhou stranu je řízení složitější a celkově náročnější na paměť, jelikož si příjemce musí uchovávat neseřazené zprávy před samotným předáním kompletní informace vyšší vrstvě.

Obrázek A.22 ukazuje způsob vypořádání se ze situace, kdy byl po cestě ztracen datový rámec. Přijímač upozorní zprávou *NAK 5* vysílací stanici, že došlo ke ztrátě rámce a dokud rámec 5 nedorazí bezchybně k příjemci, neposílá přijímací stanice žádné potvrzovací zprávy.

V případě ztráty potvrzení se chování nemění, vysílací stanice za použití SR protokolu opakovaně odešle po skončení časovače všechny nepotvrzené zprávy.

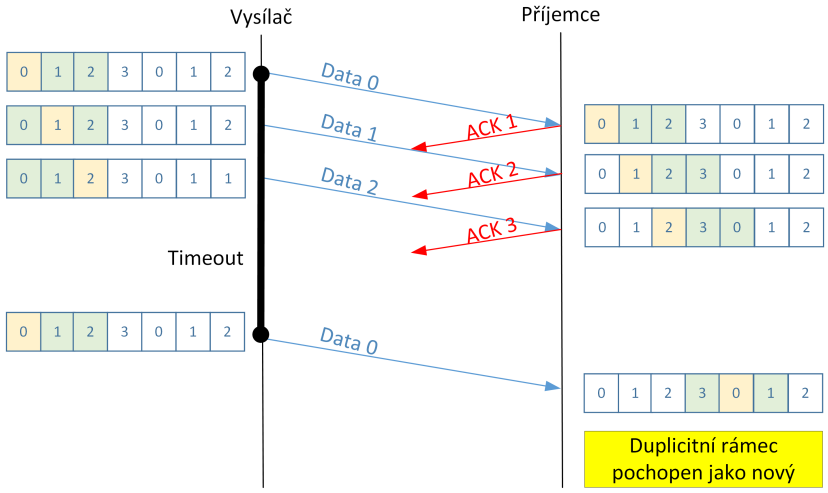


Obr. A.22: Způsob reakce protokolu Selective-repeat na situaci, kdy se ztratí odeslaný rámec. Odesílatel po upozornění opětovně odeslal pouze ztracenou zprávu.

Velikost okna u Selective-repeat

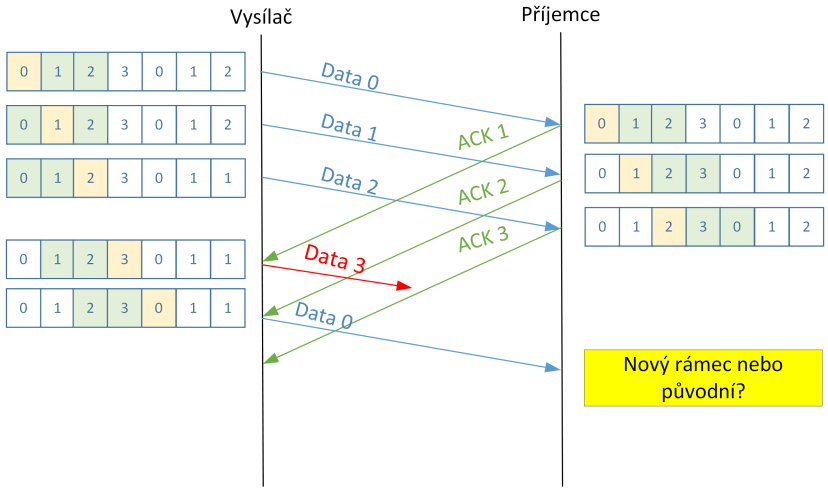
U protokolu Selective-repeat čelíme podobné potřebě zvolit vhodnou velikost posuvného okna, jako u GBN. Na rozdíl od GBN je potřeba udržovat vlastní okno pro obě strany, a to o stejné velikosti. Velikost okna je definována jako **W <= (N + 1) / 2**, tedy menší nebo rovna polovině počtu sekvenčních čísel.

Důvod je znázorněn na obrázcích A.23 a A.24. Na obrázku A.23 je popsán případ ztracení všech tří potvrzovacích zpráv. Okno na straně příjemce nyní očekává rámce se sekvenčními čísly 3, 0 nebo 1. Vysílač po skončení časovače odešle znovu první rámec, který je na straně příjemce přijatý. Problém je, že přijímač neví, zda se jedná o znovuodeslaný rámec nebo o nový.



Obr. A.23: Problém kombinace chybně zvoleného číslování s velikostí okna u protokolu Selective-repeat, kdy dochází k opětovnému vyslání původního rámce kvůli ztrátě potvrzovacích rámců.

Obrázek A.24 naopak znázorňuje scénář, kdy se potvrzení po cestě neztratí. Vysílač po obdržení prvního potvrzení *ACK 1* zareaguje odesláním následujícího rámce v pořadí, *Data 3*, který se však po cestě ztratí. Mezi tím dojde i potvrzení *ACK 2* a je odeslán další rámec *Data 0*, který v pořádku dorazí k příjemci. Z pohledu přijímací stanice je komunikace na obou obrázcích A.23 i A.24 identická, ačkoliv pokaždé došlo k jinému chybovému stavu. Z tohoto důvodu je velikost okna N-1, jak tomu bylo u GBN, nedostačující.



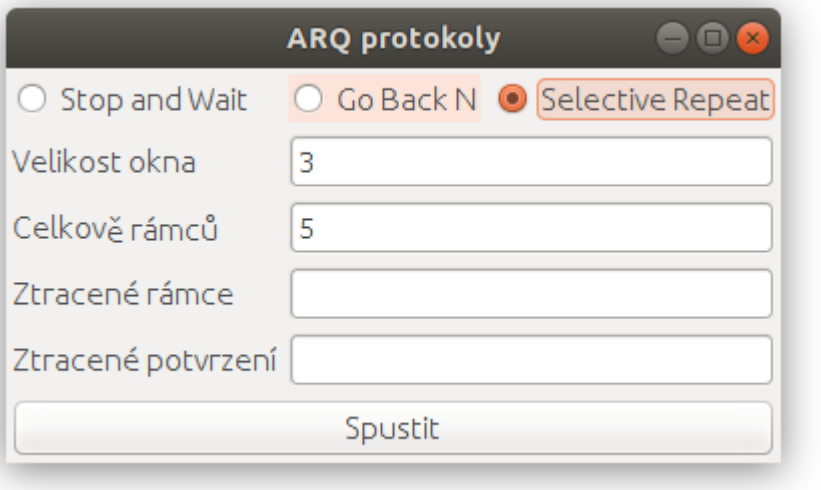
Obr. A.24: Ukázka problému s příliš velkým oknem protokolu Selective-repeat. Příjemce neví, zda je přijatý rámec opakovaně odeslán nebo se jedná o očekávaná data.

A.6.1 Laboratorní úkoly k protokolu Selective Repeat

Protokol Selective Repeat funguje na stejném principu kumulativního potvrzování, jako předchozí protokol Go Back N. Proto se vám opět může stát, že opakování scénáře nebude generovat totožný diagram (vždy by však měly být všechny výsledky korektní s ohledem na definici protokolu).

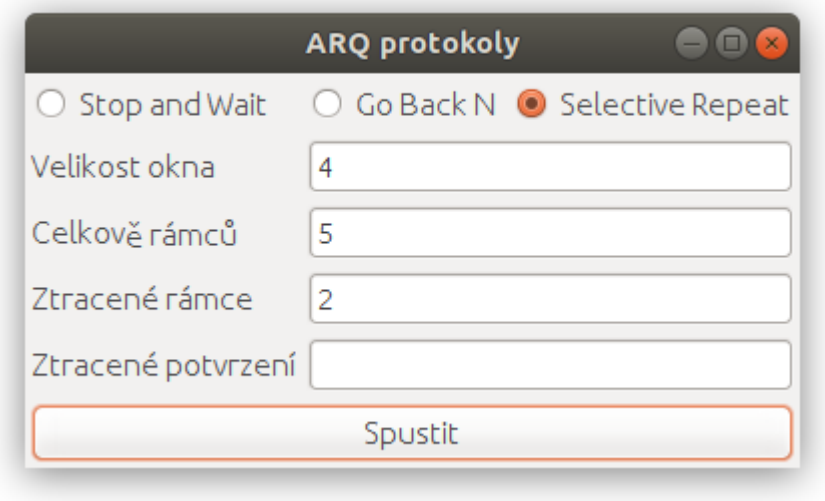
Následující úkoly proveďte obdobně, jako v předchozích úkolech, v přiloženém virtuálním prostředí s referenční aplikací simulátoru ARQ protokolů a zdrojovými kódy tohoto projektu k modifikaci.

1. V referenční aplikaci si spusťte scénář z obrázku A.25.



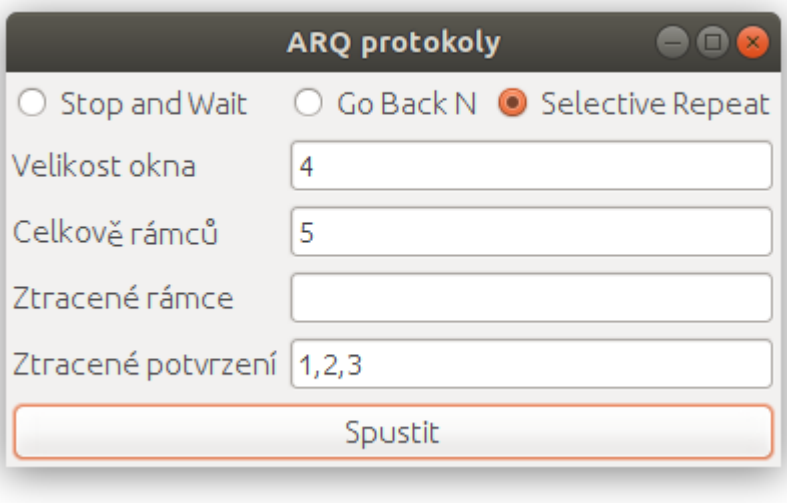
Obr. A.25: Úvodní scénář protokolu Selective Repeat.

2. Výsledný diagram přejmenujte na 'output_selective_repeat_1.svg' a vložte do vašeho protokolu. **Otázka 1: Porovnejte výstup s výstupem scénáře z obrázku A.14. V čem se oba výstupy procesně liší?**
3. Nyní se podíváme, jak si protokol Selective Repeat poradí se ztrátou datového rámce. Spusťte si scénář z obrázku A.26. Výsledný diagram přejmenujte na 'output_selective_repeat_2.svg' a vložte do svého protokolu.
   - **Otázka 2: Popište komunikaci zachycenou ve vašem diagramu.**
   - **Otázka 3: V čem by se výstup lišil, kdyby se jednalo o protokol Go Back N?**
   - **Otázka 4: Který z těchto protokolů je v tomto případě výhodnější použít?**



Obr. A.26: Scénář protokolu Selective Repeat se ztrátou datového rámce.

4. V dalším kroce se podíváme na případ, kdy dojde ke ztrátě potvrzovacího rámce. Spusťte si scénář z obrázku A.27. V seznamu potvrzovacích rámců, které mají být ztraceny opět uvádíme více než jeden, jelikož netušíme, který z nich bude součástí kumulativního potvrzení.



Obr. A.27: Scénář protokolu Selective Repeat se ztrátou potvrzovacího rámce.

5. **Otázka 5: V čem se liší výsledný diagram z předchozího kroku oproti výstupu scénáře z obrázku A.17?**
6. Volitelně si ověřte rozdíly mezi protokoly Selective Repeat a Go Back N na dalších scénářích s různými kombinacemi parametrů.
7. Nyní se vraťte zpět do vývojového prostředí Eclipse. V této části se zaměříme na implementaci protokolu Selective Repeat. Opět si prvně otevřete třídu `ProtocolFactory` a vyhledejte metodu `initSelectiveRepeat`, která se stará o spouštění scénářů s protokolem Selective Repeat.

```
private void initSelectiveRepeat(TraceBuilder tracer, Integer numberOfFrames,
        List<Integer> framesToDrop, List<Integer> acksToDrop, Integer winSize)
                throws IOException {

    /*******************************    TODO: KOD K DOPLNENI    **********************
     *
     * Do proměnné 'maxSeqNumber' přiřaďte hodnotu maximálního sekvenčního čísla
     *  v souladu s principy protokolu Selective Repeat.
     *
     ******************************************************************************/
    int maxSeqNumber = 1; // TODO: Zde změňte
```

Obr. A.28: Ukázka metody `initSelectiveRepeat`.

8. Stejným způsobem, jako u protokolu Go Back N, je zde vyznačen a popsán úsek kódu (viz. obrázek A.28), který se stará o výpočet vhodné hodnoty maximálního sekvenčního čísla s ohledem na nastavenou velikost přenosového okna z uživatelského rozhraní. `Doplňte tuto chybějící implementaci.`

9. Nyní si jako v předchozím cvičení otevřete třídy `Receiver` a `Sender`, které se tentokrát nacházejí v balíčku `com.lab.selective_repeat` a seznamte se s komentovaným kódem.
    - V prvním případě máte ve třídě `Receiver` vypočítat hodnotu sekvenčního čísla rámce negativního potvrzení, který bude odeslán jako reakce na přijetí neočekávaného rámce. Tento úsek je ukázán na obrázku A.29, dle poznámek v kódu naimplementujte přiřazení tohoto výpočtu.

```
/*********************************    TODO: KOD K DOPLNENI    **********************
 * Přijali jsme rámec mimo očekávané pořadí, takže okamžitě odešleme rámec
 *  negativního potvrzení, který vysílač uvědomí o této události.
 *
 * Do proměnné 'sequenceNumber' přiřaďte hodnotu sekvenčního čísla,
 *  kterou ponese rámec REJ.
 ******************************************************************************/
int sequenceNumber = 0; // TODO: Zde přiřaďte hodnotu sekvenčního čísla dle zadání
```

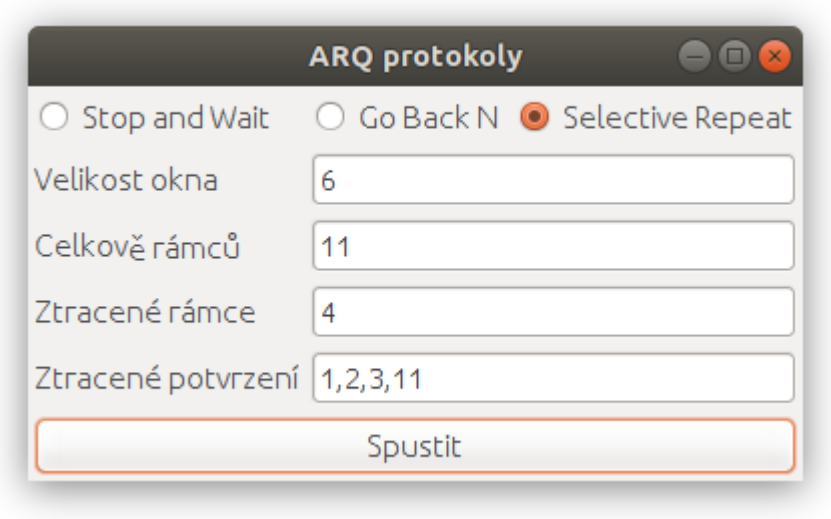
Obr. A.29: Ukázka části kódu určené k výpočtu hodnoty sekvenčního čísla

    - Dalším krokem bude ve třídě `Sender` naimplementovat dle pokynů reakci na přijetí rámce negativního potvrzení REJ (tedy navázání na problém řešený v předchozím bodě). Vašim úkolem je vytvořit/vyhledat datový rámec, který si strana příjemce přijatým rámcem REJ vyžádala. Úsek je ukázán na obrázku A.30.

```
/*********************************    TODO: KOD K DOPLNENI    **************************
 * V protokolu Go Back N se v této situaci musely zahodit všechny nepotvrzené rámce
 *  následující za sekvenčním číslem rámce negativního potvrzení. V protokolu
 *  Selective Repeat však tyto rámce přijímač v pořádku zkonzumoval a je potřeba
 *  mu zaslat pouze jediný rámec, který mu chybí.
 *
 * Vytvořte tento rámec, o který si přijímač řekl a přiřaďte jej do proměnné 'data'.
 *
 * Nápověda:
 * - Přijatý rámec REJ se nachází v proměnné 'result'.
 * - Proměnná 'unackFrames' obsahuje paylod odeslaných nepotvrzených rámců, který získáte
 *    následujícím voláním: unackFrames.get(Sekvencni_cislo_ramce_jehoz_payload_chcete)
 ******************************************************************************/
TransmissionUnit data = null; // TODO: Zde vytvořte datový rámec, který má být odeslán
```

Obr. A.30: Ukázka místa v kódu pro vytvoření rámce negativního potvrzení.

10. Po skončení implementace spusťte ve vašem řešení scénář z obrázku A.31. Výsledný diagram pojmenujte '`output_selective_repeat_impl.svg`' a vložte do svého protokolu.



Obr. A.31: Scénář protokolu Selective Repeat pro ověření správnosti implementace.

11. **Otázka 6: Uveďte nevýhody protokolu Selective Repeat oproti Go Back N.**

A.6.2 Seřazené odpovědi na otázky dle výskytu.

Otázka 1:
Porovnejte výstup s výstupem scénáře z obrázku A.14. V čem se oba výstupy procesně liší?
Odpověď:
Výstup je identický, pro scénáře bez ztracených rámců je chování protokolů Selective Repeat i Go Back N stejné.

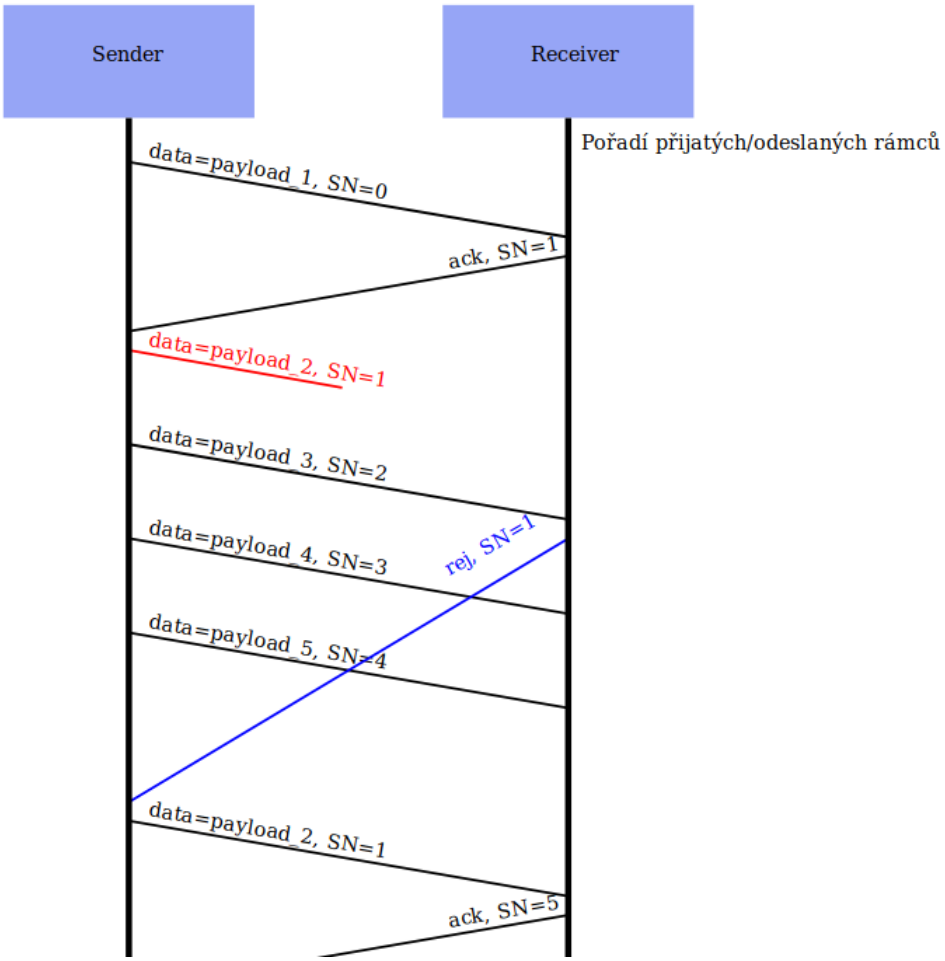
Otázka 2-4:
Popište komunikaci zachycenou ve vašem diagramu.
V čem by se výstup lišil, kdyby se jednalo o protokol Go Back N?
Který z těchto protokolů je v tomto případě výhodnější použít?

Odpověď:
Výstup na obrázku A.32.

Vysílač odešle první datový rámec *payload_1*, na který okamžitě obdrží potvrzení. Díky tomu se mu uvolní místo v přenosovém okně a může odeslat dávkově všechny zbylé rámce *paylod_2-5*. Rámec *payload_2* je po cestě ztracen, což však přijímači nebrání zkonzumovat a uložit si zbytek příchozích rámců. Přijímač si uvědomí, že došlo ke ztrátě rámce ve chvíli, kdy je neočekávaně doručen rámec *payload_3* a odesílá vysílači rámec negativního potvrzení se sekvenčním číslem očekávaného rámce. Vysílač po tomto zjištění opakovaně odešle ztracný rámec *payload_2*, který přijímač potvrdí spolu s ostatními rámci, které byly doručeny dříve.

Kdyby se jednalo o protokol Go Back N, tak by přijímač po přijetí rámce *payload_3* tento rámec zahodil (s ním i všechny následující) a vysílač by po přijetí rámce negativního potvrzení musel znovu odeslat všechny rámce *payload_2-5*.

Výhodnější je v tomto případě protokol Selective Repeat, jelikož ušetřil opakovaný přenos třech rámců, které již jednou k přijímači byly doručeny.



Obr. A.32: Diagram vygenerovaný scénářem protokolu Selective Repeat se ztrátou datového rámce.

Otázka 5:
V čem se liší výsledný diagram z předchozího kroku oproti výstupu scénáře z obrázku A.17?
Odpověď:
V ničem, proces zotavení se ze ztráty potvrzovacích rámců se u protokolu Selective Repeat neliší od protokolu Go Back N.

Otázka 6:
Uveďte nevýhody protokolu Selective Repeat.
Odpověď:
Protokol Selective Repeat vyžaduje komplexnější implementaci na straně vysílače i přijímače. Vysílač musí být schopen najít a odeslat pouze poškozený rámec. Přijímač musí po přijetí poškozeného rámce držet v paměti další přijaté rámce do té doby, než je v pořádku doručen poškozený rámec.

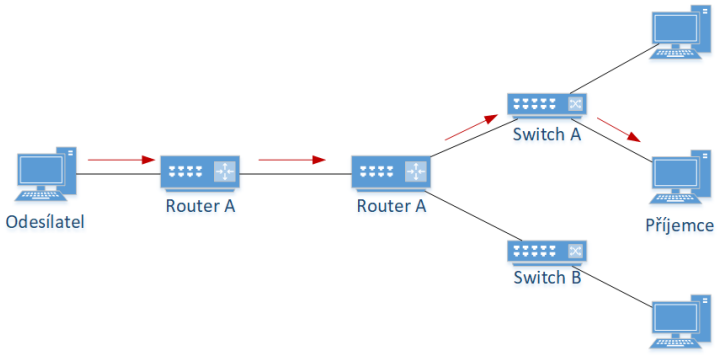
B Návod k úloze – Směrování paketů

B.1 Úvod

V moderních sítích probíhá komunikace mezi síťovými prvky přepojováním paketů, pro které se hledá nejvhodnější cesta k cílovému uzlu s ohledem na současný stav topologie sítě. Při výměně informací po paketových sítích rozlišujeme mezi čtyřmi typy přenosu podle toho, kolika cílovým zařízením je informace posílána – neboli kolik má odeslaná zpráva přijemců. V této laboratorní úloze se zaměříme na objasnění těchto režimů přenosu v IPv4 a IPv6 sítích.

B.1.1 Jednosměrový přenos

První typem je jednosměrový přenos (též známý jako *unicast*), což znamená, že informace je odeslána jedním odesílatelem k jednomu příjemci (1:1), viz. obrázek B.1. Unicast je pravděpodobně nejpoužívanějším typem přenosu, jelikož jej využívá většina dnešních internetových aplikací[4]



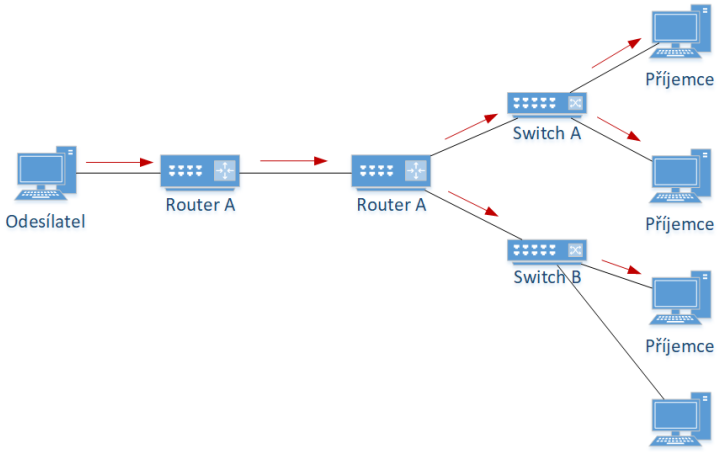
Obr. B.1: Princip jednosměrového přenosu

B.1.2 Vícesměrový přenos

Dalším typem je vícesměrový přenos (známý jako *multicast*), který se používá v případě, kdy chceme typicky informaci poslat skupině příjemců (1:`mnoho`). Tento model je výhodný pro přenos multimediálních dat, kde se uživatelé dle libosti přihlašují k odběru nějakého toku dat (např. streamovaná videa). Dále mohou nastat i případy, kdy se v komunikaci nachází i více zdrojů (`mnoho:mnoho`), kde narozdíl od předešlého typu může vysílat prakticky kdokoliv ze skupiny[4]. Zprvu se multicast používal v lokálních sítích, později však začal být podporován i v sítích rozlehlých.

Díky multicastovému přenosu se může zpráva snadno dostat k mnoha příjemcům a přitom nebýt odeslána na stejnou linku více než jednou, jak by to bylo s použitím

unicastu. S použitím multicastu, odesílatel odešle jedinou kopii dat a přepojovací zařízení v síti pak duplikují tuto zprávu kdykoliv je potřeba, aby se zpráva dostala ke všem příjemcům z cílové skupiny, jak ukazuje obrázek B.2[2].

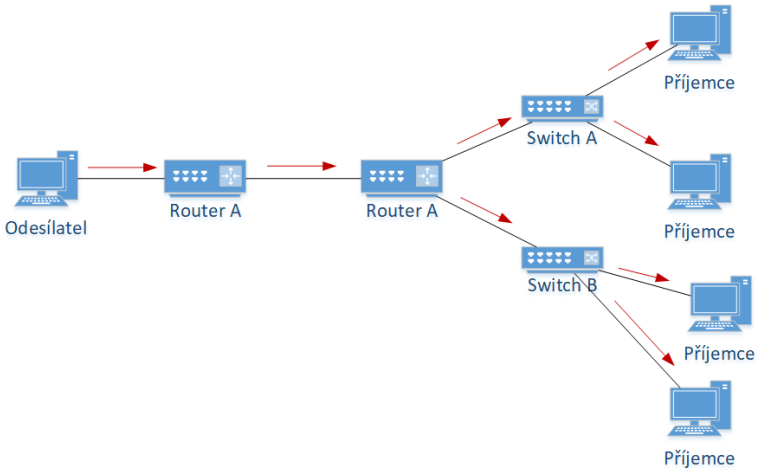


Obr. B.2: Použití skupinového přenosu pro více příjemců

B.1.3 Všesměrový přenos

Všesměrový přenos, neboli *broadcast*, je typ přenosu, kdy je jeden zdroj dat a příjemci jsou všechna ostatní zařízení v lokální síti nebo podsíti (1:`všichni`). Používá se k tomu speciální všesměrová adresa, která se utvoří tak, že všechny bity vyjadřující adresu uzlu v IP adrese se nahradí binární jedničkou – což je vždy poslední adresa v daném rozsahu. Všechny stanice v síti pak obdrží tuto zprávu a musí ji zpracovat (viz. obrázek B.3).

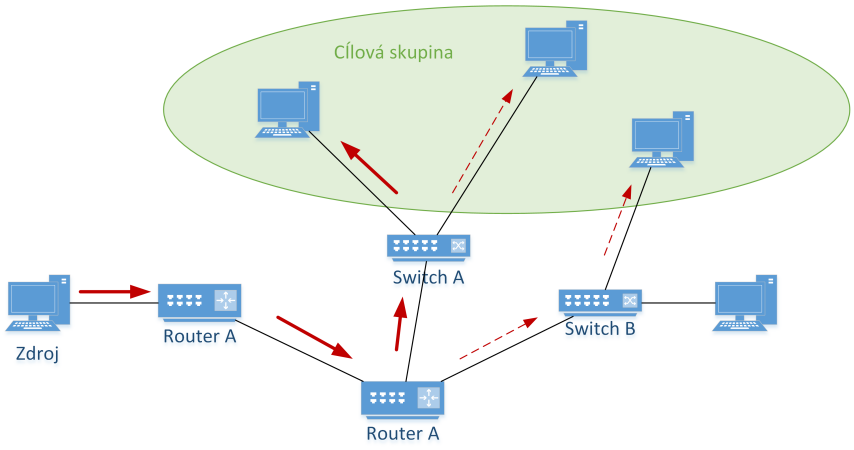
Hlavní výhodou všesměrového typu přenosu je šetření přenosového pásma a snadná dostupnost všech stanic v síti. Nevýhodou je naopak zbytečné zatěžování stanic, které zpráva nezajímá nebo se jich netýká. Aby se zahlcování nevyžádaným síťovým provozem omezilo, je vhodné sítě rozdělit na menší podsítě[4].



Obr. B.3: Princip všesměrového přenosu

B.1.4 Výběrový přenos

Posledním typem přenosu je výběrový (známý jako *anycast*), který pracuje se skupinou příjemců, kteří vystupují pod stejnou IP adresou. Když odesílatel pošle zprávu k této skupině, síť se postará o to, aby se tato zpráva dostala k jednomu příjemci ze skupiny – typicky nejbližší nebo nejvhodnější stanice (1:`kdokoli`), jak je ukázáno na obrázku B.4. Tento typ přenosu je vhodný k eliminaci redundantních zpráv (když příjemce vypadne ze sítě, zpráva je poslána jinému příjemci ze skupiny)[2].

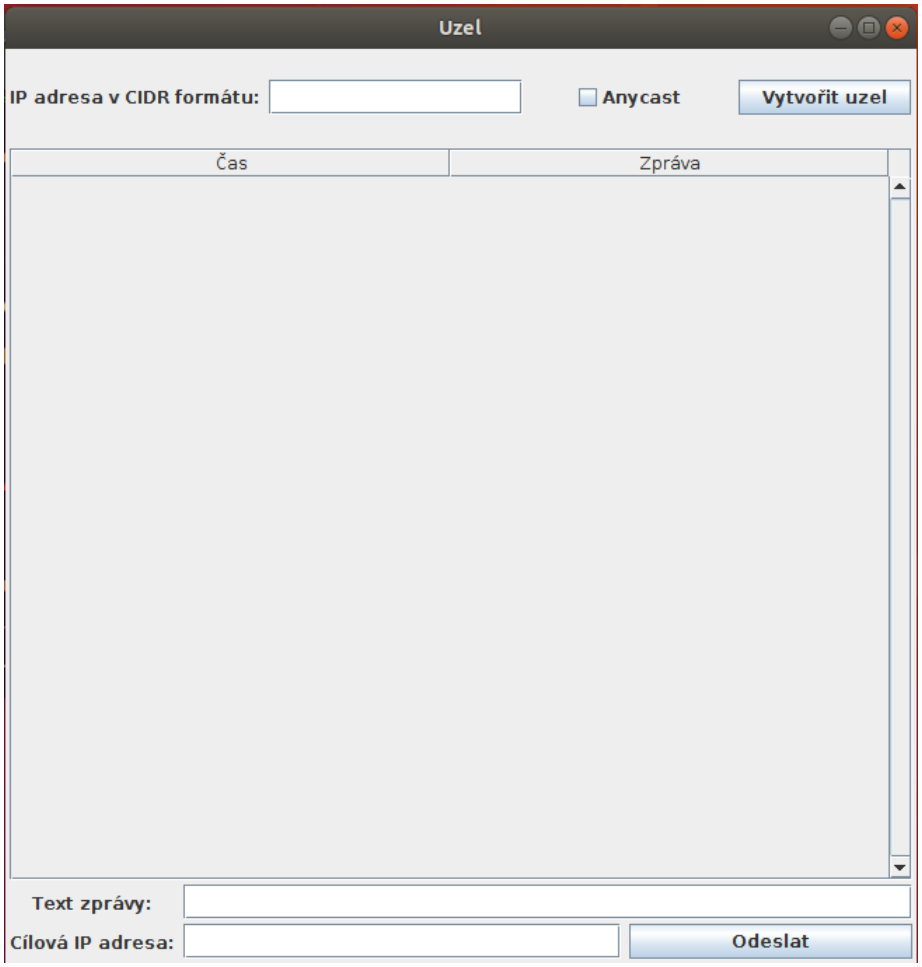


Obr. B.4: Princip výběrového typu přenosu

B.2 Implementace různých typů přenosů v prostředí s protokolem IPv4

Před samotnou implementací je potřeba se seznámit s prostředím, se kterým budete pracovat, **následně si vyzkoušíte jednotlivé úkoly na referenčním programu** a nakonec vytvoříte vlastní řešení doplněním kostry připraveného programu.

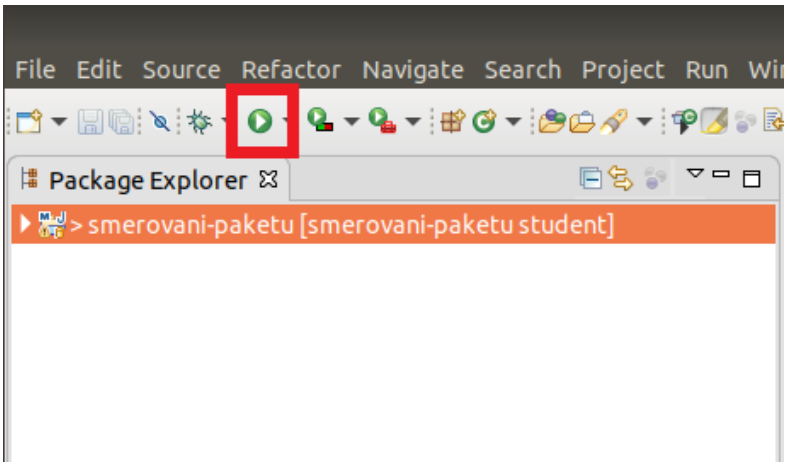
- Nabootujte se do virtuálního prostředí `Smerovani paketu` – údaje k přihlášení jsou *guest/guest*.
- Otevřete si příkazovou řádku (CTRL + ALT + T) a spusťte referenční aplikaci (*java -jar smerovani-paketu.jar &*)
- Po stvrzení příkazu se vám otevře okno, jako na obrázku B.5.



Obr. B.5: Ukázka rozhraní aplikace, reprezentující jeden uzel. Horní část se věnuje nastavení daného uzlu. Prostřední část obsahuje seznam zpráv, které tento uzel přijal. Spodní část slouží k odesílání zpráv – Do pole `Text zprávy` vložíte tělo zprávy, kterou chcete odeslat, pole `Cílová IP adresa` slouží ke specifikaci IP adresy, na kterou bude zpráva odeslána.

Toto předpřipravené řešení budeme používat pro znázornění jednotlivých režimů přenosu a ilustraci úkolů. Případně je můžete použít pro ověření správnosti vašeho řešení.

- Dále budeme potřebovat otevřít zdrojový kód, který budete pozdější upravovat, ten otevřete příkazem *sudo eclipse/java-2019-03/eclipse/eclipse*. Po potvrzení příkazu se vám spustí vývojové prostředí Eclipse s otevřenými záložkami tříd `Node.java` a `RoutingService.java`.
- Třída `RoutingService` se stará o rozhodování, kterým uzlům bude zpráva odeslána. Její metoda `sendMessage(message, destAdr)` je jediný úsek kódu, který budete v rámci tohoto laboratorního cvičení upravovat. Třída *Node* popisuje datový objekt uzlu spolu s metodami pro snadnou operaci s daty těchto objektů.
- Vaše řešení spouštíte prostřednictvím vývojového prostředí, jako na následujícím obrázku B.6:



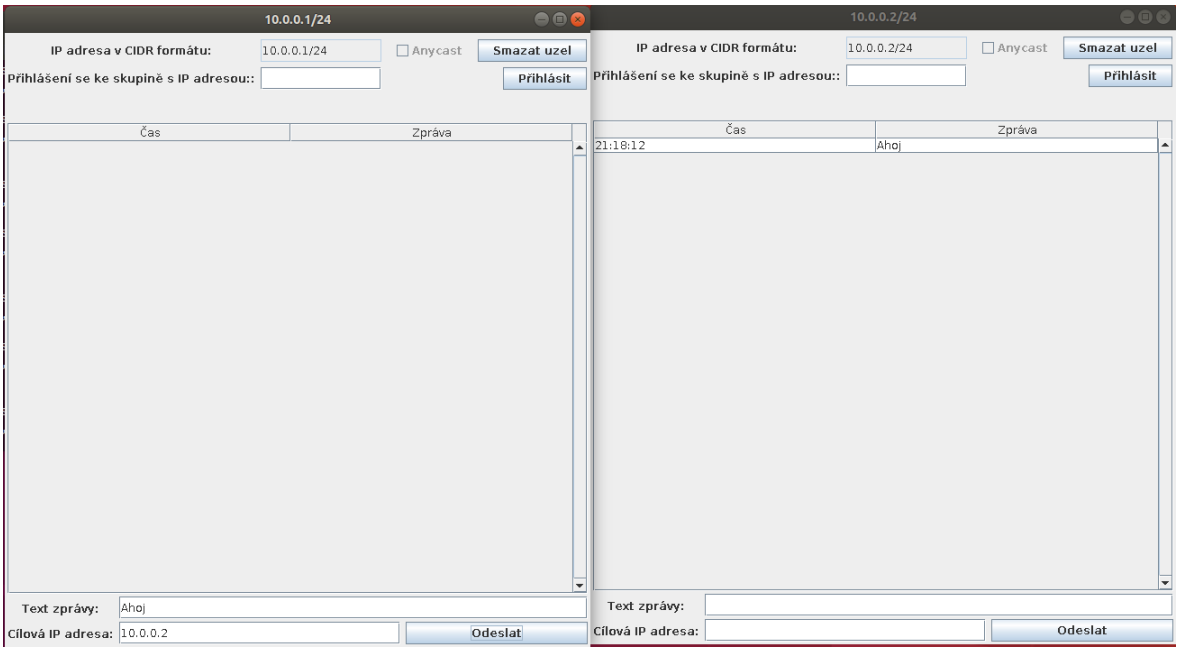
Obr. B.6: Ukázka spuštění vlastního řešení z vývojového prostředí.

- **Mějte na paměti, že pokud odesíláte zprávu z ulzu A s nějakou cílovou adresou, a mezi příjemci je opět uzel A, aplikace sama odfiltruje tyto zprávy. (Např. mám uzel A s IP adresou 10.10.10.10/24, uzel B s IP adresou 10.10.10.11/24 a odešlu z uzlu A zprávu s cílovou IP adresou 10.10.10.255, zpráva dorazí pouze do uzlu B, ačkoliv uzel A má stejnou broadcastovou adresu).**

B.2.1 Jednosměrný přenos

Jakmile se seznámíte s prostředím, začneme s prvním typem přenosu, kterým bude jednosměrný přenos. Mějte otevřená dvě okna referenční aplikace a nastavte jejich IP adresy na 10.0.0.1/24 pro uzel A a 10.0.0.2/24 pro uzel B.

Nyní zadejte v rozhraní uzlu A nějaký řetězec jako tělo zpávy a odešlete ji uzlu B. V okně uzlu B následně zkontrolujte, že zpráva přišla v originálním znění. Konfigurace by měla vypadat jako na obrázku B.7.



Obr. B.7: V horní části obou oken vidíme nakonfigurované IP adresy. Skutečnost, že proběhlo vytvoření uzlu poznáme tak, že se znění prvního tlačítka změnilo na **Smazat uzel** z předchozího **Vytvořit uzel**. V prostřední části druhého okna vidíme, že byla přijata zpráva '*Ahoj*', v závislosti na konfiguraci ve spodní části prvního okna.

1. Princip jednosměrného přenosu si experimentálně ověřte tak, že nastavíte více uzlů s různými adresami a opakujete přenos mezi různými uzly.
2. Jako krajní příklad nyní zadejte alespoň dvěma uzlům stejnou IP adresu a odešlete na ni zprávu. Všimněte si, že zpráva dorazí do všech uzlů s danou IP adresou, ačkoliv se jedná o jednosměrný přenos. **Otázka 1: Zdůvodněte proč?**
3. Jednosměrný přenos je předem pro ukázku naimplementován i ve vašem řešení. Přesuňte se nyní do kódu a seznamte se s konstrukcemi v metodě `sendMessage`, kterými bylo tohoto chování dosaženo. Ověřte si na úvodním příkladu, že toto chování skutečně funguje i ve vašem řešení.

B.2.2 Všesměrový přenos

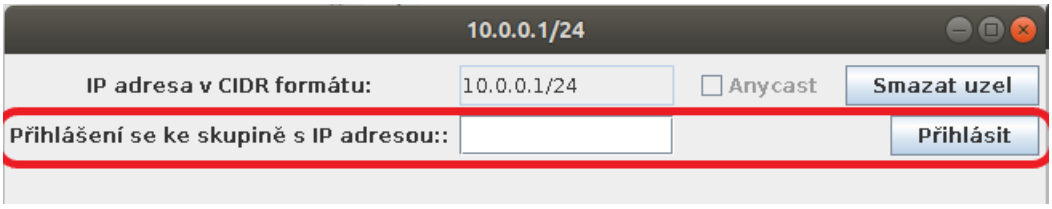
V předchozím bodě jste se seznámili s uživatelským rozhraním aplikace a úsekem kódu, do kterého budete v následujících úkolech implementovat zbylé typy přenosu.

V tomto bodě se přesuneme k dalšímu typu komunikace, kterým je všesměrový přenos.

1. Otevřete si čtyři okna referenčního řešení a proveďte konfiguraci jednotlivých uzlů následovně:
   - Uzel A má IP adresu 10.0.0.1/24
   - Uzel B má IP adresu 10.0.0.2/24
   - Uzel C má IP adresu 10.0.0.3/24
   - Uzel D má IP adresu 11.0.0.4/24
2. Nyní odešlete z uzlu D zprávu s cílovou adresou 10.0.0.255, což je všesměrová IP adresa sítí uzlů A,B a C.
3. Ověřte, že zpráva byla koretkně doručena do všech tří uzlů, a že nebyla doručena do uzlu D. **Otázka 2: Proč? Jakou cílovou adresu bychom museli zadat, aby zpráva došla do uzlu D?**
4. Volitelně si zkuste zaexperimentovat s různými délkami prefixů.
5. Po seznámení se s všesměrovým přenosem se přesuňte to kódu a zahrňte tyto vlastnosti do vašeho řešení.
6. Jakmile dokončíte implementaci, nakonfigurujte čtyři uzly vaší aplikace stejně jako v bodě 1 a ověřte funkčnost vašeho řešení na následujících scénářích:
   - z uzlu A odešlete zprávu s cílovou adresou 11.0.0.255
   - z uzlu C odešlete zprávu s cílovou adresou 255.255.255.255 – **Otázka 3: má zpráva dorazit do uzlu A i B? proč?**
   - z uzlu D odešlete zprávu s cílovou adresou 255.255.255.255 – **Otázka 4: popište a odůvodněte chování.**
   - z uzlu D odešlete zprávu s cílovou adresou 10.255.255.255 – **Otázka 5: má zpráva dorazit do uzlů A, B a C? proč?**

B.2.3 Skupinový přenos

Dostáváme se k vícesměrovému přenosu, který bude vyžadovat přihlašování stanic k odběru zpráv ze skupin. Po vytvoření uzlu se vám objeví nové pole s názvem *Odběr skupiny* (viz obrázek B.8), do kterého budete vkládat IP adresu skupiny, jejíž zprávy si přejete odebírat.

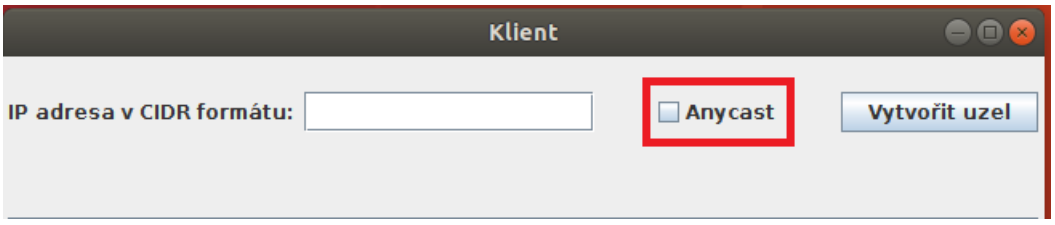


Obr. B.8: Pole pro zadání skupinové IP adresy.

1. Nakonfigurujte si nyní tři uzly referenční aplikace vzhledem k následující topologii:
   - Uzel A má IP adresu 20.0.0.1/24
   - Uzel B má IP adresu 20.0.0.2/24 a je přihlášen k odběru zpráv ze skupiny 239.7.7.7
   - Uzel C má IP adresu 20.0.0.3/24 a je přihlášen k odběru zpráv ze skupiny 239.7.7.7
   - Uzel D má IP adresu 20.0.0.4/24 a je přihlášen k odběru zpráv ze skupiny 239.7.7.7
2. Odešlete z uzlu D zprávu na cílovou adresu 239.7.7.7 a ověřte, že zpráva dorazí opravdu jen do očekávaných uzlů B a C.
3. Opět si libovolně zaexperimentujte s přihlašováním do skupin a jakmile vám bude princip jasný, tak se přesuňte k implementaci těchto vlastností do vašeho řešení, aniž by došlo k omezení funkčnosti z minulých úkolů. *Při implementaci by se vám mohla hodit metoda* `containsMulticastGroupSubscription` *třídy* `Node`*, která zkontroluje, zda je hledaná IP adresa ve skupinových adresách daného uzlu.*
4. Správnost řešení ověřte provedením uvedeného scénáře a výsledky si zaznamenejte.

B.2.4 Výběrový přenos

Posledním typem přenosu, který si u IPv4 představíme je typ výběrový. Jak už bylo naznačeno v teoretické části, výběrové směrování se od vícesměrového liší tím, že je paket odeslán pouze jedné stanici ze skupiny. Uživatelské rozhraní naší aplikace disponuje checkboxem (viz obrázek B.9), kterým určíme, zda se uzel nachází v anycastové skupině.



Obr. B.9: Checkbox, kterým řekneme, že uzel bude součástí anycastové skupiny.

1. Nakonfigurujte si tři uzly referenční aplikace tak, že:
   - Uzel A má IP adresu 10.0.0.5/24 a zapnutý Anycast
   - Uzel B má IP adresu 10.0.0.5/24 a zapnutý Anycast

- Uzel D má IP adresu 20.0.0.5/24 a zapnutý Anycast

2. Nyní několikrát odešlete z uzlu D zprávu s cílovou IP adresou 10.0.0.5 a sledujte, že zpráva dorazí vždy pouze do jednoho z uzlů A a B. Obdobně několikrát odešlete zprávu z uzlu A s cílovou IP adresou 20.0.0.5/24 a ověřte, že zpráva vždy dorazí pouze do uzlu D. **Otázka 6: Vysvětlete proč.**
3. Jakmile si osvojíte princip výběrového přenosu, opět se přesuňte do kódu a tuto funkci zabudujte do svého řešení. Informaci o tom, zda se uzel nachází v anycastové skupině zjistíte v kódu jednoduše metodou `isAnycastNode`, kterou zavoláte na instanci daného uzlu `Node`, jako je ukázáno na obrázku B.10. Typicky se při výběrovém směrování vybere uzel, který je nejblíže odesílateli, ale vzhledem k tomu, že nikde neurčujeme metriky jednotlivých uzlů proveďte volbu cílového uzlu náhodně.

```
Node node = new Node();
boolean anycast = node.isAnycastMode();
```

Obr. B.10: Způsob, jakým v kódu zjistit, zda je uzel v módu anycast

4. Správnost vašeho řešení si ověřte na příkladě se čtyřmi uzly s následující konfigurací:
   - Uzel A má IP adresu 10.0.0.5/24 a zapnutý Anycast
   - Uzel B má IP adresu 10.0.0.5/24 a zapnutý Anycast
   - Uzel C má IP adresu 10.0.0.5/24 a vypnutý Anycast
   - Uzel D má IP adresu 20.0.0.5/24 a vypnutý Anycast

   Nyní několikrát odešlete z uzlu D zprávu s cílovou IP adresou 10.0.0.5 a sledujte, že zpráva dorazí vždy pouze do jedno z uzlů A a B, a zároveň do uzlu C. **Otázka 7: Vysvětlete proč**.

B.2.5 Seřazené odpovědi na otázky dle výskytu

Otázka 1:
Jako krajní příklad nyní zadejte alespoň dvěma uzlům stejnou IP adresu a odešlete na ni zprávu. Všimněte si, že zpráva dorazí do všech uzlů s danou IPadresou, ačkoliv se jedná o jednosměrný přenos. Zdůvodněte proč?
Odpověď:
Pokud má vícero uzlů stejnou IP adresu a nesdílejí ji prostřednictvím anycastové skupiny, pak je zpráva vždy doručena do všech uzlů s touto adresou.

Otázka 2:
Ověřte, že zpráva byla koretkně doručena do všech tří uzlů, a že nebyla doručena do uzlu D. Proč? Jakou cílovou adresu bychom museli zadat, aby se zpráva doručila do uzlu D?
Odpověď:
Protože uzel je v jiné síti (11.0.0.0/24). Cílová adresa by musela být 11.0.0.255.

Otázka 3:
Má zpráva dorazit do uzlu A i B? proč?
Odpověď:
Ano. IP adresa 255.255.255.255 určuje, že zpráva má být doručena všem uzlům v dané síti – uzly A i B jsou ve stejné síti s odesílatelem, kterým je uzel C.

Otázka 4:
Z uzlu D odešlete zprávu s cílovou adresou 255.255.255.255 — popište a odůvodněte chování.
Odpověď:
Zpráva není doručena do žádného z uzlů. Uzel D je jediným členem sítě 11.0.0.0/24 a vzhledem k tomu, že je zároveň i odesílatelem zprávy, nemůže mu být doručena.

Otázka 5:
Z uzlu D odešlete zprávu s cílovou adresou 10.255.255.255 -– má zpráva dorazit do uzlů A, B a C? proč?
Odpověď:
Nemá. Adresa 10.255.255.255 je broadcastovou adresou sítě 10.0.0.0/8, ne 10.0.0.0/24.

Otázka 6:
Nyní několikrát odešlete z uzlu D zprávu s cílovou IP adresou 10.0.0.5 a sledujte, že zpráva dorazí vždy pouze do jedno z uzlů A a B. Obdobně několikrá todešlete

zprávu z uzlu A s cílovou IP adresou 20.0.0.5/24 a ověřte, že zpráva vždy dorazí pouze do uzlu D. Vysvětlete proč.

Odpověď:

Uzly A a B sdílejí IP adresu prostřednictvím anycastové skupiny, což znamená, že zpráva je doručena pouze jednomu z těchto dvou uzlů. Výběr mezi vyhovujícími cílovými uzly je náhodný.

Druhý případ ukazuje, že ačkoliv je uzel D v režimu anycast, je jediným členem této skupiny a zprávy směřované na jeho adresu tedy vždy musí skončit zde. Pokud je uzel v režimu anycast a neexistuje žádný další uzel v této skupině, chování je ekvivalentem k unicastu.

Otázka 7:

Nyní několikrát odešlete z uzlu D zprávu s cílovou IP adresou 10.0.0.5 a sledujte, že zpráva dorazí vždy pouze do jedno z uzlů A a B, a zároveň do uzlu C. Vysvětlete proč.

Odpověď:

Uzly A a B v režimu anycast sdílejí IP adresu 10.0.0.5/24, takže zpráva je náhodně doručena do jednoho z těchto dvou uzlů. Zároveň je zde však uzel C se stejnou IP adresou, který stojí mimo anycastovou skupinu, takže se na něj předchozí algoritmus nevztahuje a aplikuje se kalsický unicastový režim – proto obdrží všechny zprávy směřované na IP adresu 10.0.0.5.

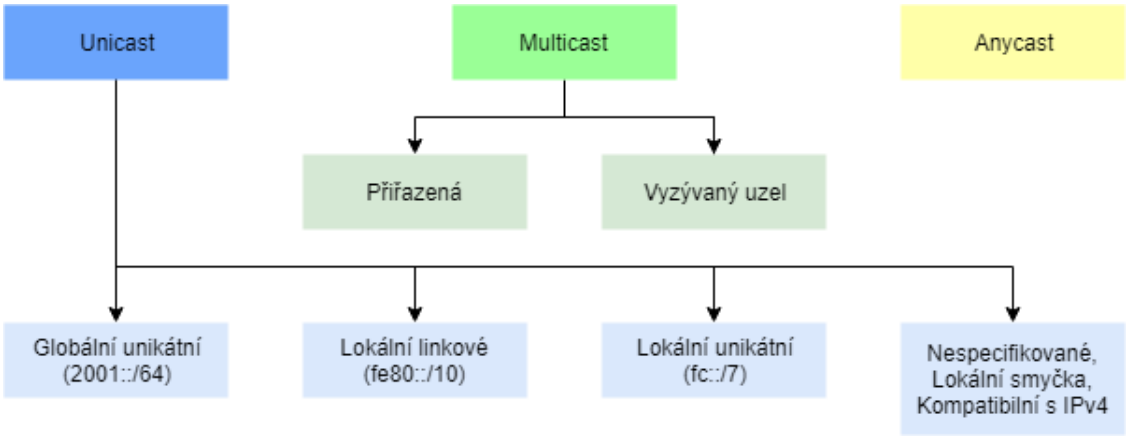
B.3 Implementace různých typů přenosů v prostředí s protokolem IPv6

IP protokol verze 6 přichází s novým způsobem adresování. Definovány jsou pouze tři druhy adres:

B.3.1 Individuální a výběrové adresy

Individuální adresy podobně jako u IPv4 identifikují jednotlivá síťová rozhraní. Výběrové adresy identifikují skupinu adresátů s tím, že je paket odeslán typicky nejbližšímu členu této skupiny – obecně se tento přístup může hodit v případě, kdy máme nějakou velmi vytíženou službu, u které si přejeme rozložit zátěž mezi více uzlů, např. systém DNS.

U IPv6 je běžné, že pro síťové rozhraní existuje hned několik adres. IPv6 přichází s konceptem různých typů individuálních adres (viz obrázek B.11). V tomto úkolu se zaměříme na globální unikátní adresy, které slouží k běžnému užití (srovnatelné s veřejnými IPv4 adresami). Prefix globální unikátní adresy je opět velice podobný tomu, jaký známe z IPv4.



Obr. B.11: Základní rozdělení typů IPv6 adres[4].

Vzhledem k tomu, že se princip globálního unikátního a výběrového přenosu u IPv6 výrazněji neliší od IPv4, bude následující úkol spíše ilustrační.

1. Otevřete si čtyři okna referenční aplikace a proveďte nássledující:
   - IP adresu uzlu A na 2002:db8:a319:15:22a:fff:fe32:5ed1/64, bez anycastu
   - IP adresu uzlu B na 2002:db8:a319:15:22a:fff:fe32:5ed1/64, s anycastem
   - IP adresu uzlu C na 2002:db8:a319:15:22a:fff:fe32:5ed1/64, s anycastem

- IP adresu uzlu D na 2003:db8:a319:15:22a:fff:fe32:5ed1/64, s anycastem

2. Z uzlu D odešlete zprávu s cílovou IP adresou 2002:db8:a319:15:22a:fff:fe32:5ed1 a ověřte, že zpráva doputovala do uzlu A a zároveň do jednoho z uzlů B nebo C.
3. Z uzlu A odešlete několik zpráv na adresu 2003:db8:a319:15:22a:fff:fe32:5ed1 a ověřte, že zpráva vždy doputuje do uzlu D. **Otázka 1: Proč se zpráva objeví v uzlu D pokaždé i přesto, že se jedná o výběrovou adresu?**
4. Podobně jako u předchozích úkolů si vyzkoušejte různé kombinace, dokud si princip neosvojíte.
5. Upravte implementaci svého řešení tak, aby podporovala přenos zpráv za použití globálních unikátních adres a správnost si ověřte na příkladu z bodu 1.

B.3.2 Skupinové adresy

Skupinové adresy slouží k adresování skupin. Stejně jako u IPv4 jsou tyto adresy vhodné nejen pro přenos multimediálních dat v reálném čase. Ve skupinové adrese má prvních 8 bitů hodnotu `FF`, další 4 bity slouží ke specifikaci určitých voleb, následující 4 bity definují dosah platnosti adresy a zbylé bity slouží pro specifikaci samotné adresy skupiny. V rámci této laboratoře budeme pracovat pouze se skupinovými adresami, které mají lokální (hodnota `2`) a globální (hodnota `E`) dosah platnosti.

Ačkoliv protokol IPv6 přímo neobsahuje všesměrové adresy, existuje v každé IPv6 síti adresa `FF02::1`, která kontaktuje všechny uzly na lokální síti. Pomocí této adresy může zařízení kontaktovat všechny koncové stanice na lokální síti, což je efektivně režim broadcastu, který známe z IPv4.

Všechny koncové stanice se automaticky přihlašují ke *skupinové adrese pro vyzývaný uzel* pro každou přidělenou individuální a výběrovou adresu. Tato adresa má vždy stejnou formu: **ff02::1:ff** + posledních 24 bitů přidělené individuální adresy. Pokud mám tedy přidělenou individuální adresu 2001::01:800:200E:8C6C, má skupinová adresa vyzývaného uzlu bude ff02::1:FF0E:8C6C. Tyto adresy se používají v protokolu NDP (Neighbor Discovery Protocol) pro zjištění, které uzly se nacházejí v lokální síti.

1. Otevřete si nyní dvě okna referenční aplikace a nakonfigurujte uzly tak, že:
   - Uzel A má IP adresu 2001:db8:a319:15:20a:fff:fe32:5ed1/64
   - Uzel B má IP adresu 2001:db8:a319:15:2**1**a:fff:fe32:5ed1/64
   - Uzel C má IP adresu 2001:db8:a319:15:21a:fff:fe32:5ed**2**/64
   - Uzel D má IP adresu 2001:db8:a319:15:2**2**a:fff:fe32:5ed2/64
2. Odešlete zprávu z uzlu C na cílovou adresu ff02::1:ff32:5ed1. **Otázka 2: Proč zpráva dorazí pouze do uzlů A a B?**

3. Vytvořte další uzel E s IP adresou 200**2**:db8:a319:15:21a:fff:fe32:5ed1/64 a odešlete z něj zprávu na stejnou cílovou adresu ff02::1:ff32:5ed1. **Otázka 3: Proč zpráva nedorazí do žádného uzlu?**
4. Odešlete zprávu z uzlu A na cílovou adresu ff02::1. **Otázka 4: Popište, co se stalo, a zdůvodněte.**

5. Naimplementujte do svého řešení podporu pro zmíněné skupinové adresy. Implementaci následně ověřte na stejném postupu, jako s referenční aplikací.

B.3.3 Seřazené odpovědi na otázky dle výskytu

Otázka 1:
Proč se zpráva objeví v uzlu D pokaždé i přesto, že se jedná o výběrovou adresu?
Odpověď:
Protože je uzel D jediným uzlem v dané anycastové skupině, takže není mezi kým vybírat.

Otázka 2:
Proč zpráva dorazí pouze do uzlů A a B?
Odpověď:
Uzle A a B se při vytvoření automaticky přihlašují ke skupinové adrese vyzývaného uzlu ff02::1:ff32:5ed1 (konstanta **ff02::1:ff** + posledních 24 bitů přidělené individuální adresy **32:5ed1**). Skupinová adresa vyzývaného uzlu pro C a D se liší, takže nemohou být mezi adresáty odesílané zprávy.

Zároveň je splněna podmínka síťového rozsahu: Uzle A a B se nacházejí ve stejné síti 2001:db8:a319:15::/64 s uzlem C, který je odesílatelem zprávy.

Otázka 3:
Proč zpráva nedorazí do žádného uzlu?
Odpověď:
Ačkoliv je skupinová adresa vyzývaného uzlu pro uzel E shodná s uzly A a B, není dodržena podmínka dosahu (ff02). Uzel E se nachází v síti 2002:db8:a319:15::/64, zatímco uzly A a B jsou v síti 2001:db8:a319:15::/64.

Otázka 4:
Popište, co se stalo, a zdůvodněte.
Odpověď:
Zpráva doputuje do uzlů B, C a D (není v A, jelikož se jedná o odesílatele zprávy). IP adresa ff02::1 se chová podobně jako 255.255.255.255 u IPv4, tedy adresáty jsou všechny uzly v lokální síti uzle A (2001:db8:a319:15::/64). Uzel E se v této síti nenachází.

C Obsah přiloženého média

- Textový soubor `prostredi.txt` obsahující webový odkaz na virtuální prostředí pro laboratorní scénáře. Tyto soubory nejsou součástí média z důvodu jejich velikosti.
- Zdrojové soubory vyvinutých aplikací – `Sources/`
- Návody k laboratorním úlohám – `Navody/`
- Text této práce – `dp.pdf`
- Zdrojové soubory tohoto textu pro případné rozšíření scénářů – `dp.zip`