Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Vizualizace komunikace virtuální sítě v rodině protokolů TCP/IP

2024

Thanh Tú Phan

Vedoucí práce:
Mgr. Radek Janoštík, Ph.D.

Studijní program: Informační technologie,
prezenční forma

Bibliografické údaje

Autor:	Thanh Tú Phan
Název práce:	Vizualizace komunikace virtuální sítě v rodině protokolů TCP/IP
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2024
Studijní program:	Informační technologie, prezenční forma
Vedoucí práce:	Mgr. Radek Janoštík, Ph.D.
Počet stran:	46
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Thanh Tú Phan
Title:	Visualization of virtual network communication in the TCP/IP protocol family
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2024
Study program:	Information Technologies, full-time form
Supervisor:	Mgr. Radek Janoštík, Ph.D.
Page count:	46
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Spletitost počítačových sítí může mnohé od jejího studia snadno odradit. Snahou této práce je vytvoření jednoduchého simulátoru počítačové sítě, který by prostřednictvím přehledné vizualizace usnadnil studentům pochopení základních principů síťové komunikace. Součástí práce je rozbor stávajících řešení, popis implementovaných částí síťové komunikace, použitých technologií a uživatelská dokumentace.

Synopsis

The complexity of computer networks can easily discourage students from studying them. The aim of this thesis is to create a simple computer network simulator that would facilitate understanding of the basic principles of network communication through comprehensive visualization. The work includes an analysis of existing solutions, a description of implemented parts of network communication, used technologies, and a user documentation.

Klíčová slova: počítačová síť, vizualizace, simulace, TCP/IP

Keywords: computer network, visualisation, simulation, TCP/IP

Děkuji vedoucímu práce Mgr. Radku Janoštíkovi, Ph.D. za velmi cenné rady, trpělivost a povzbudivé vedení během vývoje a psaní této práce. Rád bych také poděkoval své rodině, která mi během studia byla vždy velkou oporou.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Vznik počítačových sítí dovolil snadnou vzdálenou komunikaci, sdílení informací, zdrojů a poskytování široké škály síťových služeb. Tyto služby dnes denně využíváme k práci a ve volném čase, ať už se jedná o prostý e-mail nebo procházení internetu. Málokoho už ale napadne se pozastavit a popřemýšlet, co všechno se v pozadí odehrává. Problematika síťové komunikace je nesmírně složitá a rozmanitá. Zatímco běžní uživatelé se touto problematikou zaobírat nemusí, budoucí informatici by měli být alespoň s jeho základy obeznámeni.

Studium počítačových sítí však nemusí být mezi studenty informatických oborů zcela atraktivní. Uvážíme-li výuku programování, kterou lze v dnešní době vést různými poutavými formami, mohou studenti již při minimálních znalostech vidět pokroky a postupně se kreativně rozvíjet. Naproti tomu studium počítačových sítí vyžaduje pochopení řady netriviálních principů doprovázené množstvím technických záležitostí, které mohou studenty od jeho studia snadno odradit. Nedílnou součástí výuky je praktická práce se sítí, avšak její reálné zřízení může být velmi nákladné. V praxi se proto využívají různé nástroje, které počítačovou síť simulují. Nicméně, tyto nástroje jsou často technicky velmi náročné a může být obtížné se s nimi naučit pracovat.

Úkolem mé práce je tak vytvořit aplikaci umožňující tvorbu jednoduché virtuální sítě, která by usnadnila pochopení těch nejzákladnějších principů síťové komunikace za pomocí přehledné vizualizace. Pomoci by měla zejména vyučujícím počítačových sítí při demonstraci vybraných oblastí.

1.1 Motivace

Výuce počítačových sítí na katedře informatiky přírodovědecké fakulty Univerzity Palackého jsou ve druhém ročníku bakalářského studia věnovány dva kurzy. Kurz *Počítačové sítě 1* vyučovaný v zimním semestru seznamuje studenty se základy počítačových sítí. Rozšiřující kurz *Počítačové sítě 2* v letním semestru se pak podrobněji věnuje pokročilejším protokolům a technologiím, jmenovitě bezdrátovým sítím, protokolu *NAT* a *IPv6*.

Součástí obou kurzů jsou cvičení, na kterých jsme si na virtualizovaných strojích vyzkoušeli reálnou konfiguraci sítě. Bylo však náročné si na nich uvědomit analogii probrané látky z přednášek na reálné komunikaci. Často se stávalo, že při zpracovávání úkolu zaměřeného na jednu část problematiky jsme zapomněli na kritickou roli jiné části, která byla představena na předešlých hodinách. K identifikaci problémů se nabízí využít některý z nástrojů pro zachycení a analýzu síťové komunikace, jako je například *Wireshark*. Ten sice poskytuje detailní informace, kterým zkušení budou rozumět, nezkušené však spíše zahltí. Absence nástroje, který by usnadnil si síťovou komunikaci lépe představit, často vedla k metodě „pokus-omyl“ bez hlubšího chápání. Vysvětlování takové problematiky bez vhodné vizualizace je nejspíše obtížné i pro vyučující.

1.2 Intuitivní pohled

Mojí představou je uživateli umožnit základní konfiguraci sítě, která by reflektovala část cvičení z kurzu Počítačové sítě 1. Aplikace by měla umožnit si sestavit vlastní síť z několika základních síťových zařízení, libovolně je po obrazovce umisťovat, propojovat a nakonfigurovat. Veškerá práce by probíhala v grafickém rozhraní, čímž odstraníme závislost na platformě, konkrétních příkazech a umístění konfiguračních souborů. Součástí vizualizace by také měla být možnost krokování, zpomalení a náhled na vyměňované zprávy. Kromě demonstrace by vizualizace mohla pomoci i při konfiguraci sítě. Vzhledem k rozsahu problematiky síťové komunikace by v aplikaci byly implementovány pouze nejdůležitější části probrané v kurzu Počítačové sítě 1. Pro demonstrační účely ve výuce by tyto části byly značně zjednodušeny a nemusely by plně reflektovat reálné implementace. Podrobněji si popíšeme a odůvodníme výběr těchto částí v kapitole 3.2.

2 Analýza existujících řešení

Během průzkumu jsem narazil na dva přístupy, jak virtuální síť realizovat. Prvním přístupem je naprogramovat si simulovanou síť, která bude reálné chování pouze napodobovat. Výhodou tohoto přístupu je lehkost a větší možnost přizpůsobení. Druhým přístupem je síť emulovat za použití virtuálních strojů. Zjevnou nevýhodou je těžkost a také potřeba si obrazy virtuálních strojů sami zajistit.[1] Zajistí však věrnější práci se sítí. Některá řešení jsou komerční a jiná slouží primárně k výzkumu – ty nebudeme v analýze brát v úvahu. V následující části si rozebereme obecné přístupy a dostupné nástroje, které by se daly potenciálně využít k výuce počítačových sítí. Pokusíme se identifikovat jejich přednosti, které mohou návrhu aplikace pomoci, a zároveň také nedostatky, kterým se naopak pokusíme vyvarovat.

2.1 VirtualBox

VirtualBox [1], přesněji Oracle VM VirtualBox, je multiplatformní open-source virtualizační nástroj vyvíjený pod záštitou Oracle. Díky virtualizaci hardwaru na hostitelském stroji umožňuje nástroj spouštět několik virtuálních strojů s různými operačními systémy. Můžeme jej využít k tvorbě virtuální počítačové sítě, na které si lze věrně procvičit práci se sítí v reálných prostředích. Takovéto řešení je nepochybně levnější a dostupnější oproti reálnému řešení. U emulátorů obecně jsme ale stále omezeni svými hardwarovými prostředky, které určují velikost sestavitelné sítě. Navíc s rostoucím počtem strojů se správa jejich oken rychle stává nepřehledná, což znesnadňuje práci jak studentovi, tak vyučujícímu, který práci kontroluje. Ačkoliv jde o poměrně jednoduchý a dostupný způsob, jak reálnou práci se sítí umožnit, může z počátku působit poněkud složitě. VirtualBox jakožto virtualizační nástroj neposkytuje uživateli žádnou vizualizaci síťové komu-

[1] Mnohdy je také potřeba si zajistit licenci.

nikace, pomineme-li nástroje jako Wireshark [2], které si na jednotlivých strojích můžeme nainstalovat.

Poměrně složitý je také způsob, jakým se stroje ve VirtualBoxu propojují do sítě. Abychom je propojili do stejné sítě, musíme v nastavení síťového adaptéru ve VirtualBoxu nastavit shodný typ a název sítě. Pokud bychom chtěli propojit několik strojů do stejné lokální sítě, musíme nastavit každému stroji síťový adaptér typu „Vnitřní síť“ se shodnými názvy. Nicméně, to, jak jsou tato zařízení vnitřně propojena, uživateli skryto[2] a pouhý překlep v názvu či neshodující se typ sítě mohou být častým zdrojem problému. Během klonování virtuálních strojů z jejich obrazů navíc nesmíme zapomenout příslušně změnit MAC adresy síťových adaptérů, jinak by došlo ke konfliktům těchto MAC adres. Toto je pouze část problémů, na které můžeme při konfiguraci sítě ve VirtualBoxu narazit. Chyby můžeme udělat na mnoha místech a jejich odhalení pak může být obtížné.

2.2 Cisco Packet Tracer

Cisco Packet Tracer [3] je simulátor sítě společnosti Cisco navržený k podpoře kurzů Cisco's Networking Academy [4]. Nezávazně ho ale můžeme používat zdarma registrací v daných kurzech. K tvorbě topologie sítí nabízí širokou škálu modelů síťových zařízení Cisco, která lze po obrazovce umisťovat a různými typy spojů propojovat. Konfigurace a komunikace mohou být řízeny v simulované příkazové řádce nebo v grafickém rozhraní. Aplikace podporuje dva režimy. Ve výchozím *realtime* režimu probíhá veškerá komunikace v pozadí. Zajímavější je režim *simulace*, který vizualizuje průchod paketů po síti s možností filtrace, krokování a detailního náhledu na obsah paketů. Dále umožňuje zobrazit fyzický pohled na zařízení, na kterém si lze simulovat práci se zařízením. Pokud bychom například chtěli přidat nový síťový adaptér, musíme ve fyzickém pohledu zařízení nejdříve vypnout. Poté z mnoha typů adaptérů jeden vybrat, přetáhnout ho na určený slot, zařízení zapnout a čekat než se spustí. Tyto kroky by byly nutné provést pro každé zařízení, jehož fyzickou konfiguraci bychom chtěli upravit. Ačkoliv je důležité jednotlivé typy kabelů, zařízení a jejich účely umět rozlišit, tato míra techničnosti a snaha o realističnost se odchylují od našeho stanoveného cíle pro jednoduchost. Kromě toho může aplikace působit zastarale a množství nabízených funkcí ji dělá nepřehlednou. Možnosti aplikace plní účel v nabízených kurzech Cisco, které připravují na pokročilé certifikace počítačových sítí. K představení základů však může aplikace působit poměrně komplikovaně.

2.3 GNS3

GNS3 [5], celým názvem Graphical Network Simulator-3, je open-source emulátor počítačové sítě, který je tvořen klientskou částí poskytující grafické uživatelské rozhraní a serverovou částí řídící emulaci. Server lze vytvořit lokálně

[2]Uživatel nemá možnost si vyzkoušet práci s propojovacími prvky, jako jsou například hub nebo switch, o kterých se později v textu zmíníme více.

na počítači, spíše se ale doporučuje využít některého z virtualizačních softwarů, jako jsou VMWare Workstation, VirtualBox a dalších, které možnosti v GNS3 rozšiřují. Samotný instalační proces tohoto emulátoru je poměrně těžkopádný, jelikož je při něm nutná instalace několika závislostí třetích stran. Navíc se může instalace na jednotlivých platformách výrazně lišit. Rozdíl mezi GNS3 a VirtualBoxem je to, že v GNS3 jsou virtualizované stroje zobrazeny na obrazovce, po které je lze libovolně rozmístit a propojit. GNS3 také umožňuje k této emulované síti připojit reálné fyzické zařízení. Oproti Cisco Packet Traceru však GNS3 neposkytuje režim simulace, který by zobrazoval průchod přenášených zpráv. Tento emulátor je vhodným nástrojem pro osvojení práce na reálně používaných operačních systémech pomocí virtualizovaných strojů, testování, dokazování a podobně. V průběhu používání se ale můžeme setkat s množstvím technických problémů spojených jak se samotnou aplikací, tak emulovaným systémem.[3]

2.4 Shrnutí

Kromě přílišné technické zaměřenosti zmíněných řešení se můžeme potýkat s náročným zprovozňováním – zejména u emulátorů, které jsou ke své funkci závislé na softwaru třetích stran. Emulované řešení je značně komplikované a jeho implementace by vyžadovala znalost mnoha nízkoúrovňových záležitostí. Jednodušší variantou je si virtuální síť simulovat, čímž získáme plnou kontrolu nad implementací, při které se odstíníme od technických detailů.

Aplikace zmíněných řešení jsou implementovány nativně, a proto se může jejich instalace i samotná implementace na různých platformách lišit. Jelikož je naše aplikace určená pro výukové účely, bylo by vhodné instalační proces a chování aplikace co nejvíce sjednotit. Zajímavé by bylo se od stávajících řešení odlišit a pokusit se virtuální síť realizovat ve webové aplikaci. Ta má sice oproti nativním aplikacím omezené možnosti, jako je třeba výkon, přístup k úložišti a závislost na připojení k Internetu, nicméně vhodnou implementací můžeme z velké míry zajistit její přenositelnost na různých platformách. Nemusíme se tedy zaobírat komplikovaným instalačním procesem, se kterým jsem se potýkal u výše zmíněných řešení. Implementací se podrobněji budeme zabývat v kapitole 4.

[3] Podobně jako u VirtualBoxu.

3 Požadavky na aplikaci

V této kapitole si představíme síťový model *TCP/IP* a popíšeme výběr jeho vizualizovaných částí. Dále zmíníme funkční požadavky, které budeme na výslednou aplikaci klást.

3.1 Síťový model TCP/IP

Složitost síťové komunikace vedla k vzniku síťových modelů, které tuto problematiku abstrahují do vrstev. Jednotlivé vrstvy jsou tvořeny sadou protokolů řídící síťovou komunikaci. Protokolem můžeme rozumět sadu stanovených pravidel určující syntaxi, význam, pořadí a časování vyměňovaných zpráv.[4] Nejvýznamnějším síťovým modelem je zmíněný TCP/IP, který rozlišuje 4 vrstvy v následujícím sestupném pořadí: *aplikační*, *transportní*, *internetová* a *vrstva síťového rozhraní*. V mnoha literaturách, včetně knihy Počítačové sítě [6] od autorů Jim Kurose a Keith Ross, se setkáváme s pětivrstvou definicí, která vrstvu síťového rozhraní drobněji rozděluje na fyzickou a linkovou (linková je nad fyzickou). Toto pětivrstvé rozdělení nám usnadní vysvětlování této vrstvené síťové architektury i její následnou implementaci, a proto ji budeme v textu využívat.

Princip vrstvených architektur spočívá v tom, že jedna vrstva, vykonávající svojí část práce, využívá služeb vrstvy bezprostředně pod ní. Danou vrstvu přitom nezajímá, jakým způsobem nižší vrstvy svoje služby implementují. Taková nezávislost umožňuje snadnou výměnu vrstvy za jinou a napomáhá pochopení a identifikaci problémů. Během komunikace jednotlivé vrstvy TCP/IP zapouzdřují data z vyšší vrstvy svou hlavičkou.[5] Ta obsahuje potřebné režijní informace pro odpovídající vrstvu na uzlech, které data zpracovávají. Těmito uzly rozumíme buď mezilehlé uzly, jimiž vede cesta k cílovému uzlu, nebo samotný cílový uzel. Role mezilehlých uzlů bude postupně rozebrána dále v textu.

3.2 Vybrané části TCP/IP

S ohledem na množství protokolů TCP/IP je nutné se omezit na vybranou podmnožinu. V této práci zcela vynecháme vrstvu aplikační, která je tvořena řadou protokolů, jako jsou například HTTP, DNS či FTP. Z pohledu uživatele může být aplikační vrstva zajímavá, ale kvůli pestrosti ponecháme implementaci protokolů této vrstvy případnému rozšíření. V následujících částech této podkapitoly si popíšeme protokoly zbylých vrstev, které považuji pro vizualizaci a základní fungování sítě za důležité a nezbytné. Seřazeny jsou sestupně po vrstvách.

[4] Síťové protokoly jsou standardizovány různými autoritami. Tyto standardy nalezneme na https://www.rfc-editor.org/.

[5] V případě linkové vrstvy je to i patička.

3.2.1 Transportní vrstva

Transportní vrstva zajišťuje přenos dat mezi procesy aplikační vrstvy. Procesy jsou instancemi síťových aplikací, kterých může na uzlu běžet více. K jejich rozlišení se proto využívá dvoubajtový číselný *port*. Síťové aplikace realizují přenos dat skrze programové rozhraní, které nazýváme *soket*. Každý soket je definován využívanou službou transportní vrstvy, lokální a vzdálenou adresou – jedná se o dvojice IP adresy koncového uzlu a portu procesu, který na uzlu běží. Následovně si stručně popíšeme dva základní transportní protokoly *TCP* a *UDP*. Budeme přitom používat pojmy *klient* a *server*. Klientem můžeme rozumět proces, který komunikaci zahajuje, zatímco serverem rozumíme proces, který na kontaktování čeká.

3.2.1.1 Protokol TCP

Transmission Control Protocol, zkráceně TCP, je spojová transportní služba zajišťující *spolehlivé doručení dat* ve správném pořadí a *řízení toku dat* k zamezení zahlcení sítě či příjemce. K zaručení těchto vlastností se data z aplikační vrstvy zapouzdřují do *TCP segmentů*, které kromě portu odesílatele a příjemce uchovávají také řídící informace. Jde především o pořadové číslo posílaných dat, číslo potvrzení, řídící příznaky a informace o zahlcení. Před samotnou výměnou dat musí mezi klientem a serverem proběhnout takzvaný *třífázový handshake*, během kterého si vymění výchozí stavové informace o spojení. Ukončení pak probíhá tří nebo čtyřfázově v závislosti na vzájemném záměru o ukončení. Kvůli technické složitosti protokolu TCP se jím podrobněji zabývat nebudeme. Pokusíme se však v jednoduché podobě implementovat navázání spojení, třífázové a čtyřfázové ukončení.

3.2.1.2 Protokol UDP

Naproti tomu protokol UDP (User Datagram Protocol) žádné spojení nevytváří a ani nezaručuje doručení aplikačních dat. Výhodou jeho nespolehlivosti je to, že oproti TCP nemusí přenášet a zpracovávat tolik režijních informací. Důsledkem toho je pak větší propustnost přenosu, čehož využívají aplikace, které určitou míru ztrátovosti dokáží tolerovat. Příkladem jsou multimediální aplikace, kde drobné chyby v přenosu zvuku či obrazu nemají rozhodující dopad na kvalitu. Poznamenejme, že absence jakékoliv kontroly může způsobit zahlcení sítě či příjemce, v jehož důsledku může naopak dojít ke snížení efektivity přenosu. Protokol UDP pro svou poměrnou přímočarost ponecháme k případnému rozšíření.

3.2.2 Internetová vrstva

Internetová vrstva zajišťuje adresaci uzlů, určení cesty a předávání zpráv mimo lokální síť. Data z transportní vrstvy se v této vrstvě zapouzdřují do takzvaných *IP paketů* obsahující adresu odesílatele, adresu příjemce a další položky. Během

komunikace mimo lokální síť IP pakety běžně procházejí několika *směrovači*, které mají na starosti jejich předávání. Jednou z důležitých položek IP paketu je TTL (*Time-To-Live*), jehož hodnotu směrovače snižují k zamezení nekonečného přenosu. V nadcházející části si podrobněji popíšeme základní složky internetové vrstvy a jejich implementované části. V zájmu zachování jednoduchosti aplikace však nejsou zmíněny mnohé další služby, kterými internetová vrstva disponuje.

3.2.2.1 IP protokol

IP protokoly definují adresní konvenci, formát IP paketů a způsob, jak je s IP pakety na směrovačích manipulováno. V současné době existují dvě verze protokolů – starší *IPv4* a jeho nástupce *IPv6*. Hlavním důvodem vzniku IPv6 je nedostatečný adresní rozsah čtyřbajtových adres IPv4, který je vzhledem k velikosti soudobého Internetu nedostačující. Ačkoliv je přestup na IPv6 v budoucnu nevyhnutelný, v době psaní této práce je jak v Internetu, tak ve výuce IPv4 stále silně zastoupen. V práci proto budeme uvažovat pouze IPv4 a nebudeme rozlišovat speciální typy IP adres.[6]

3.2.2.2 Směrování

Důležitou složkou internetové vrstvy je směrování, jehož úkolem je určování cesty k cílového uzlu. K tomuto účelu je využíváno směrovacích tabulek obsahujících záznamy o další cestě. Součástí každého záznamu je *adresa sítě*, *maska sítě*, *next-hop* (adresa uzlu, kterému se má paket předat) a odchozí síťové rozhraní, na kterém se next-hop nachází.

V souvislosti se směrováním je důležité zmínit, že IP adresa má hierarchickou strukturu a rozdělujeme ji na dvě části – adresující síť a adresující uzel v síti. Maska sítě je čtyřbajtová hodnota, která je v bitové reprezentaci daná posloupností počátečních n jedniček. Při prohledávání směrovací tabulky se aplikuje bitová operace *AND* na masku sítě v záznamu směrovací tabulky a cílovou IP adresu. Výsledek se poté porovná s adresou sítě v záznamu a v případě shody se IP paket předá příslušnému next-hop skrze dané odchozí síťové rozhraní. Směrovací tabulky jsou organizovány od nejspecifičtějších adres sítí (adres s nejdelší maskou) po nejméně specifické. Existence masky sítě nám umožňuje agregaci sítí pomocí obecnější adresy (s kratší maskou), čímž se omezí množství směrovacích záznamů, které je nutné procházet. *Výchozí brána* je pak záznam s adresou sítě 0.0.0.0/0, na jehož odchozí síťové rozhraní je paket předán, pokud se žádný z předchozích záznamů neshodoval. Pro snadnější práci se síťovou maskou použijeme formát *CIDR*, kdy se za IP adresou připisuje lomítko s délkou masky sítě. Například zápis 192.168.1.10/24 značí, že prvních 24 bitů IP adresy je vyhrazeno pro adresu sítě, tedy 192.168.1.0, a zbývající bity adresují uzel v rámci této sítě.

[6] Například adresu zpětné smyčky (*loopback*), lokální či broadcastové adresy.

Obsah směrovacích tabulek se v praxi ve větších sítích naplňuje dynamicky podle některého směrovacího protokolu. Teoreticky bychom se mohli pokusit implementovat minimální variantu některého z nich. Vyvstává však přitom několik otázek: kdy protokol provádět, v jakém intervalu a jak jej vizualizovat. Ve výsledku by to spíše způsobovalo zmatky. Ponecháme tedy na uživateli, aby si směrovací tabulky naplnil a procvičil si tím principy směrování.

3.2.2.3 ICMP

ICMP (Internet Control Message Protocol) slouží k diagnostickým a signalizačním účelům v síti. Protokol je využíván nástroji *ping* nebo *traceroute*, které diagnostikují dosažitelnost uzlů v síti. Vytvářejí k tomu ICMP zprávy určené několika položkami – zajímat nás bude především typ těchto zpráv. Nejčastějšími typy ICMP zpráv, se kterými se můžeme setkat, jsou *time exceeded*, pokud vyprší TTL v IP paketu, a *destination unreachable*, pokud neexistuje cesta k cíli nebo cílový uzel není dostupný. Pro ověřování dosažitelnosti implementujeme zjednodušený nástroj ping, který ze zdrojového uzlu odešle *echo request* (požadavek na odpověď), během něhož může dojít k některé z následujících situací:

1. požadavek je doručen příjemci,
2. na směrovači byla snížena hodnota TTL IP paketu na 0,
3. ze směrovače cesta k příjemci nevede.

V první situaci odpoví příjemce zprávou *echo reply*. V druhé situaci je IP paket směrovačem zahozen a odesílatel informován zprávou *time exceeded*. V třetí situaci je odesílatel směrovačem informován zprávou *destination unreachable*.

Program traceroute kromě ověření dosažitelnosti poskytuje také informace o mezilehlých uzlech, kterými IP paket projde. Jeho implementace však nemá smysl, protože průchod paketů můžeme pozorovat v samotné vizualizaci.

3.2.3 Linková vrstva

Odpovědností linkové vrstvy je řízení přenosu dat na úrovni sousedních uzlů propojených komunikačními kanály, které dále budeme označovat jako *linky*. Pakety z IP vrstvy zapouzdřuje tato vrstva do takzvaných *linkových rámců*, jejichž formát je dán technologií fyzické vrstvy. Ta se může odlišovat jak typem fyzické linky či formátem přenášených rámců, tak poskytovanými linkovými službami.[7] V rámci naší aplikace se omezíme na *Ethernet II*, se kterým se setkáme v drátových (ethernetových) sítích. Součástí hlavičky linkového rámce Ethernet II je MAC adresa odesílatele, adresa příjemce a typ protokolu vyšší vrstvy. Linkový rámec je dále zaopatřen patičkou s redundantní informací kontrolního součtu

[7] V případě bezdrátové technologie Wi-Fi se k přenosu využívá rádiových vln ve vzduchu, které jsou obecně náchylnější k chybám. Oproti spolehlivějšímu ethernetu se tak musí implementovat pokročilejší mechanismy detekce a opravy chyb.

k detekci poškozených rámců. MAC adresy jsou šestibajtovými identifikátory *síťových adaptérů*,[8] které zprostředkovávají rozhraní k fyzickému propojení mezi uzly. Dodejme, že většina služeb linkové vrstvy bývá implementovaná v rámci samotného síťového adaptéru, tedy v hardwaru. Mezi tyto služby patří zapouzdření do linkových rámců, řízení přístupu k lince a detekce chyb. V případě vícenásobného přístupu na lince dochází k degradaci vysílaného signálu, a proto může být vyžadováno řízení přístupu k lince. Technikami řízení vícenásobného přístupu k lince a detekcí chyb se však vzhledem k náročnosti jejich vizualizace zabývat nebudeme.

V lokální síti lze síťová zařízení propojit linkami různými způsoby. Nejčastěji se dnes setkáme s propojením za pomoci centrálního prvku. Propojovacímu zařízení využívající ke své funkci informací z linkové vrstvy říkáme přepínač, případně *switch*. Switch disponuje několika fyzickými porty, ke kterým se připojují zařízení v lokální síti. Zařízením je switch, ke kterému jsou připojena, transparentní. Zařízení tedy pracují stejně, jako kdyby byla komunikující zařízení připojena přímo. Switch si udržuje mapování MAC adres a svých portů na základě proběhlé komunikace. Přijetím linkového rámce si zapamatuje mapování adresy odesílatele a příchozího fyzického portu. Switch je tak schopný přepínat linkové rámce přímo adresátovi, díky čemuž se zamezí zatížení zbylých linek. Pokud switch nezná fyzický port, na kterém se příjemce nachází, přepošle daný linkový rámec skrze všechny porty vyjma příchozího. Takový přenos se označuje jako *broadcast*. Jestliže je zařízením explicitně vyžadováno, aby všichni v lokální síti jeho zprávu přijali, zapouzdří zprávu do linkového rámce s broadcastovou MAC adresou ff:ff:ff:ff:ff:ff.

3.2.3.1 ARP

Address Resolution Protocol, zkráceně *ARP*, je nedílnou součástí síťové komunikace. Jelikož jsou uzly v lokální síti navzájem propojeny na linkové úrovni, potřebují ke vzájemné komunikaci znát adresy MAC. Jestliže odesílatel nezná MAC adresu příjemce, vyšle broadcastem v lokální síti *ARP požadavek*, kterým se dotazuje uzlu s danou IP adresou, zda by nemohl odpovědět se svojí MAC adresou. Příjemce s odpovídající IP adresou následně odešle *unicastem*[9] *ARP odpověď* se svými adresami IP i MAC. Odesílatel si po přijetí ARP odpovědi uloží mapování IP a MAC adresy příjemce do své ARP cache.[10] V tomto momentě má odesílatel všechny potřebné informace k odeslání požadované zprávy.

3.2.4 Fyzická vrstva

Jedná se o nejnižší vrstvu TCP/IP, která se zabývá fyzickým přenosem binárních dat po fyzické lince. Součástí vrstvy je problematika převodu signálů, modulace

[8] Jinými nazvy také síťová karta nebo síťové rozhraní.

[9] Příjemce odpoví ARP zprávou přímo adresátovi (odesílateli ARP požadavku).

[10] Tyto záznamy jsou v cache uloženy po vymezenou dobu. Cílem je zamezit frekventovaným ARP požadavkům.

a také jednotlivé fyzické prvky (kabely, konektory). Přihlídneme-li na úroveň techničnosti, omezíme se pouze na fyzické propojení, které bude napodobovat ethernetové sítě. Fyzické vlastnosti linky, jako jsou rychlost, propustnost, spolehlivost a zmíněné nízkoúrovňové záležitosti, nebudeme brát v úvahu. Zanedbáme také vliv vzdálenosti linek mezi zařízeními.

3.3 Shrnutí a přehled funkcionalit

Při fyzickém propojování zařízení se odstíníme od konkrétních typů fyzických linek a jejich specifických vlastností, které by mohly komplikovat tvorbu a porozumění vizualizace. Zařízení a fyzické linky se pokusíme implementovat co nejobecněji. Práce s fyzickými adaptéry a jejich propojení by u všech typů zařízení měla probíhat stejně a nebudeme ani omezovat počet fyzických adaptérů na zařízení.

K vytváření komunikace budou k dispozici: ping, třífázové navázání, tří a čtyřfázové ukončení TCP spojení. Průběh komunikace bude reflektovat stav zařízení a uživatel by měl mít možnost si obsah směrovacích tabulek a ARP tabulek prohlédnout a nastavit. Informace o proběhlé komunikaci použité ve vizualizaci budou vhodně vypočítány a uloženy tak, aby mohly být zpětně použity.

Kromě routerů a switchů zmíněných v předchozí kapitole budeme také implementovat koncové zařízení, které bude schopné vytvářet TCP spojení. K demonstraci výhod switche implementujeme propojovací zařízení `hub`, které se už dnes kvůli jeho omezením příliš nepoužívá. Následuje přehled síťových zařízení, které budeme implementovat:

- **Hub** (rozbočovač) – jednoduché propojovací zařízení, které přijatý signál zopakuje skrze zbylé porty, signál tedy broadcastuje.[11]
- **Switch** (přepínač) – propojovací zařízení využívající k efektivnějšímu přepínání informací linkové vrstvy.
- **Router** (směrovač) – propojovací zařízení využívající informací internetové vrstvy k předávání IP paketů mezi sítěmi.
- **Computer** (koncové zařízení) – obecný počítač připojený k síti schopný navazování TCP spojení.

Aby si uživatel nemusel pokaždé síť od základů sestavovat, bude důležité umožnit si stav aplikace uložit a načítat. Obrazovkou by mělo jít posouvat, zmenšovat a zvětšovat. Při vizualizaci komunikace by měly být možnosti krokování a zpomalení zprostředkovány pomocí přehrávače, jehož součástí by mohl být progress bar, který by ukazoval přehranou délku vizualizace.

[11] Hub pracuje na fyzické úrovni a přijatým datům v signálu nerozumí. Nemůže je tak využít k efektivnějšímu přepínání.

4 Technická dokumentace

V této kapitole si popíšeme a odůvodníme volbu architektury, technologií a knihoven použitých při vývoji aplikace. Dále si vysvětlíme, jakým způsobem byly tyto technologie využity při realizaci samotné vizualizace.

4.1 Architektura

Aplikace je implementována jako *Single-Page-Application* [7], zkráceně SPA. Tato aplikace je reprezentovaná jednou webovou stránkou, jejíž obsah se průběžně aktualizuje interakcí uživatele. Oproti tradičním webovým stránkám, které se při změně obsahu musí pokaždé celé načíst, poskytuje SPA uživateli výrazně plynulejší interakci. V principu to funguje tak, že klientovi, který si chce stránku zobrazit, poskytne webový server potřebné zdrojové kódy, které se ve webovém prohlížeči klienta spustí a zobrazí. V první řadě se jedná o kód ve značkovacím jazyce *HTML* [8] neboli Hypertext Markup Language, kterým popisujeme obsah, sémantiku a strukturu stránky. V druhé řadě kód v jazyce *CSS* [9], zkratka pro Cascading Style Sheets, kterým definujeme a upravujeme vzhled stránky. Jelikož jsou HTML a CSS velmi dobře známé technologie, nebudeme se jimi podrobněji zabývat a vystačíme si s těmito intuitivními definicemi.

S HTML a CSS však dokážeme vytvářet pouze statické stránky.[12] Důležitou roli v SPA hraje *JavaScript* [10]. Jde o dynamický skriptovací jazyk, který díky přístupu k *DOM* (Document Object Model) [11] struktuře umí za běhu upravovat obsah webové stránky. Strukturou DOM rozumíme programové rozhraní, které umožňuje přístup ke struktuře webové stránky v jeho objektové reprezentaci. JavaScript je tedy hlavní složkou, která nám umožňuje přinést interaktivitu do webových stránek. V kontextu SPA dále disponuje JavaScript důležitou technikou zvanou *Ajax*,[13] kterou lze pomocí klientského JavaScriptu asynchronně posílat a zpracovávat data z webového serveru. Klientským JavaScriptem tak může být na webové stránce zaregistrována událost, o které je asynchronně oznámen server. Server pak odpoví příslušnými daty či přímo částí stránky, podle nichž se stránka JavaScriptem překreslí, aniž by došlo k jejímu znovunačtení.

Z hlediska cílené jednoduchosti a povahy zamýšlené aplikace nám nebrání se vymezit čistě na klientský JavaScript. Jedinou rolí webového serveru je na začátku JavaScriptový soubor spolu se soubory HTML a CSS poskytnout. Pro úplnost dodejme, že nevýhodou takové SPA je pomalejší prvotní načtení stránky, protože se musí načíst většina zdrojů. Problém také bývá s indexací SPA webovými prohlížeči kvůli jejich dynamické povaze.[14] Uvážíme-li však problémy, s nimiž jsem se potýkal u existujících řešení, jsou problémy SPA poměrně drobné.

[12] Webový dokument s neměnným obsahem.

[13] Více na https://developer.mozilla.org/en-US/docs/Glossary/Ajax.

[14] Optimalizací webové stránky pro usnadnění její indexace webovými prohlížeči se zabývá SEO (Search Engine Optimization). Více na https://developer.mozilla.org/en-US/docs/Glossary/SEO.

4.2 Použité technologie

V této podkapitole se zaměříme na frameworky a knihovny, které usnadňují práci se základními technologiemi představenými v podkapitole 4.1. Rozdíl mezi knihovnou a frameworkem spočívá v tom, že frameworky kromě poskytování funkcí a abstrakce nad řešenými problémy také určují a prosazují osvědčené metody psaní a organizace kódu.

4.2.1 Tailwind CSS

Tailwind CSS [12] je CSS framework založený na principu *utility-first*. Na rozdíl od tradičního psaní CSS poskytují tyto frameworky *utility třídy* s malými předdefinovanými CSS pravidly. Kombinaci těchto utility tříd pak zapisujeme k vizualizovaným HTML elementům. Vizualizace elementů je díky tomu dána jednoznačně a není nutné přecházet mezi HTML a CSS soubory. Další výhodou je to, že nemusíme řešit složitou organizaci tradičního CSS kódu, který má tendenci se velmi rozrůstat. Velikost CSS utility-first frameworků zůstává téměř neměnný.[15] Použití frameworku značně šetří čas, jelikož není nutné se zabývat tvorbou vhodných jmen selektorů a dodržováním různých konvencí. Častou kritikou však bývá to, že tímto porušujeme oddělení odpovědností CSS a HTML a znepřehlední se tím výsledný kód HTML. Tento problém ale můžeme zmírnit rozdělením kódu do komponent použitím některých z JavaScriptových frameworků, o kterých se zmíníme ve zbytku podkapitoly.

Rád bych dodal, že značnou část aplikace jsem původně psal podle tradičních konvencí. Vzhledem k častým změnám jak vzhledu, tak struktury aplikace, byla aktuálnost CSS náročná udržovat. Z tohoto důvodu jsem se rozhodl přejít ke knihovně Tailwind, která se velmi osvědčila. Přepis trval několik málo dní a následný vývoj aplikace byl výrazně usnadněn.

4.2.1.1 DaisyUI

Ačkoliv jsem většinu uživatelského rozhraní aplikace napsal sám, v některých částech jsem pro úsporu času využil komponenty z knihovny DaisyUI [13], která je založena na Tailwind CSS.

4.2.2 Vue.js

Vue.js [14] je JavaScriptový framework usnadňující tvorbu uživatelských rozhraní, zejména pak SPA. Podobně jako React [15] či Angular [16] zasazuje rozdělování uživatelského rozhraní do komponent. V případě Vue.js se doporučuje komponenty organizovat do takzvaných *Single-File-Component*, zkráceně SFC, kdy stav a logiku (JavaScript), strukturu a sémantiku (HTML), vzhled (CSS) komponenty deklarujeme v jednom samostatném souboru. Podporovány jsou přitom dvě syntaxe: Options API a více doporučovaný Composition API,

[15] V sestaveném produkčním CSS souboru se navíc ponechávají pouze použitá CSS pravidla.

kterému je moje práce uzpůsobena. Příklad SFC a Composition API můžeme vidět ve zdrojovém kódu 1 (viz kapitola 4.5.2). Výhodou frameworků je obecně to, že zatímco v JavaScriptu bychom uživatelská rozhraní vytvářeli procedurálně nízkoúrovňovou úpravou DOM, frameworky umožňují problémy řešit více deklarativním stylem. Ve Vue.js je syntaxe HTML rozšířena o takzvaná *direktiva*, která zajišťují vazbu mezi daty a uživatelským rozhraním. Příklady jsou direktiva `v-if` pro podmínečné vykreslování, `v-for` pro vykreslení kolekce prvků nebo `v-model` pro obousměrnou vazbu mezi daty a formuláři. *Reaktivní systém* Vue.js zajišťuje detekci změn v datech a překreslení příslušných prvků uživatelského rozhraní. Frameworky navíc implementují různé přístupy práce s DOM, čímž dochází k zefektivnění jeho úprav. Výsledkem toho je mnohdy přehlednější, udržitelnější a zároveň efektivnější kód než ten, který bychom psali samotným JavaScriptem. Framework Vue.js je známý svou snadnou osvojitelností a použitím. Není náhodou, že jsem během vývoje své aplikace nesčetněkrát narážel na příklady použití s Vue.js v oficiálních dokumentacích či fórech různých technologií.

4.2.2.1 Pinia

Pinia [17] je jednou z oficiálních knihoven Vue.js pro centralizovanou správu stavu aplikace. Jak již bylo zmíněno, každá komponenta má svou logiku a stav, je ale běžné, že na některém stavu závisí více komponent.

Mějme stav `devices`, který reprezentuje přidaná zařízení. Dále komponenty `Board` a `DeviceList`, z nichž první je komponenta obrazovky, na které se zařízení vizualizují, a druhá poskytuje přehled zařízení. Otázkou je, kde tento stav vhodně definovat. Jednou z možností je ve společném rodiči, kterým je například komponenta `App`.[16] Tento přístup je ovšem poněkud ošemetný, zvláště pokud by byly komponenty Board či DeviceList hluboko zanořeny. Museli bychom předat tento stav skrze parametry každé komponenty mezi rodičem a komponentou, která tento stav vyžaduje. Situaci také komplikuje fakt, že při provádění kterékoliv změny musíme dbát na to, abychom tyto vazby nijak neporušili.

Pinia poskytuje rozhraní pro tvorbu `store`, který budeme dále označovat jako stavový sklad. Kromě stavu v něm definujeme metody, jejichž prostřednictvím mohou libovolné komponenty ke stavu přistupovat či jej modifikovat. Instance stavového skladu definujeme v odděleném souboru, který může být importován a použit v libovolné komponentě či JavaScriptovém souboru.

[16] Jedná se o kořenovou komponentu.

4.3 Použité knihovny

V této podkapitole se budeme věnovat knihovnám použitým při vývoji aplikace. Podrobněji se přitom zaměříme na rozbor možností, které lze využít k tvorbě animací, a zvolené řešení.

4.3.1 GSAP

Významnou součástí práce je kromě implementace virtuální sítě také vizualizace její komunikace. Musíme mít na paměti, že práce bude potenciálně rozšiřována a nároky na vizualizaci budou přibývat. Důležitý je proto výběr knihovny, která bude snadno a efektivně umožňovat tvorbu komplexních animací. K tomuto účelu byla využita JavaScriptová knihovna *Green Sock Animation Platform*, zkráceně *GSAP* [18]. Podívejme se ale prvně na alternativní možnosti a jejich nedostatky, které k použití této knihovny vedly.

Jde-li o tvorbu webové grafiky a animací, mnohé napadne *Canvas API* [19]. Jeho použití je však poměrně složité. Přestože existují knihovny, které práci s canvasem značně usnadňují, neodstiňují nás dostatečně od tvorby netriviální překreslovací logiky. Canvas je omezený na bitmapovou grafiku, která je vykreslována v rámci `<canvas>` elementu. To nám znemožňuje využít systému událostí JavaScriptu k detekci a zpracování událostí vizualizovaných objektů.

Snazší možností je animovat samotné objekty DOM, jejichž vzhled si dle potřeb můžeme vizualizovat pomocí CSS a využít externí grafiku včetně vektorové grafiky SVG. Nemusíme se tak zabývat složitým vykreslováním bitmapy, ani implementací detekce událostí. K animacím objektů DOM můžeme využít přístupů, jako jsou *CSS transformace* nebo *Web Animations API* – ty jsou však v jistých ohledech omezené. Pomocí CSS transformací dokážeme vytvářet spíše jednodušší efekty. Slibnější jsou knihovny založené na Web Animations API, mezi nimiž patří například *Motion One* [20], který umožňuje poměrně snadným způsobem animaci CSS atributů a pokročilé možnosti skládání animací. Výhodou těchto knihoven je malá velikost, jelikož využívají funkce poskytované prohlížečem. Tyto knihovny jsou však v době psaní práce relativně nové, a nemáme tak plnou jistotu v jejich spolehlivosti.

Naproti tomu JavaScriptová knihovna GSAP je již roky osvědčená. Kromě animací CSS umožňuje animovat libovolné atributy objektů. V porovnáních výkonnosti knihoven [21] GSAP také značně vyniká. Její základní funkce jsou zahrnuty v takzvaném *GSAP core*, který je dostupný bezplatně.[17] GSAP dále poskytuje pluginy, které rozšiřují jeho možnosti nad rámec GSAP core. Příkladem je plugin *Draggable,*[18] který umožňuje posouvání vizualizovaných objektů po obrazovce. Ukázky použití knihovny uvidíme dále v textu.

[17] Základní funkcionality GSAP core jsou k dispozici pod licencí No Charge, která nám dovoluje funkcionality používat zdarma a bez omezení, pokud si za námi vytvořený software, který knihovnu používá, nebudeme účtovat poplatky.

[18] Více na https://gsap.com/docs/v3/Plugins/Draggable/.

4.3.2 Panzoom

K uživatelské přívětivosti přispěla knihovna Panzoom [22], která umožňuje přibližování, oddalování a posuny po obrazovce, nebo přesněji po určeném HTML elementu, ve kterém se vizualizace odehrává. Tato funkcionalita je zvláště přínosná při tvorbě rozsáhlejších topologií sítí, protože nejsme omezeni velikostí viditelné obrazovky.

4.3.3 Nanoid

Knihovna Nanoid [23] byla použita k tvorbě unikátních identifikátorů pro vizualizované objekty. Tyto identifikátory jsou použity jako selektory a zjednodušují proces exportu a zpětného importu vizualizovaných objektů.

4.3.4 Lodash

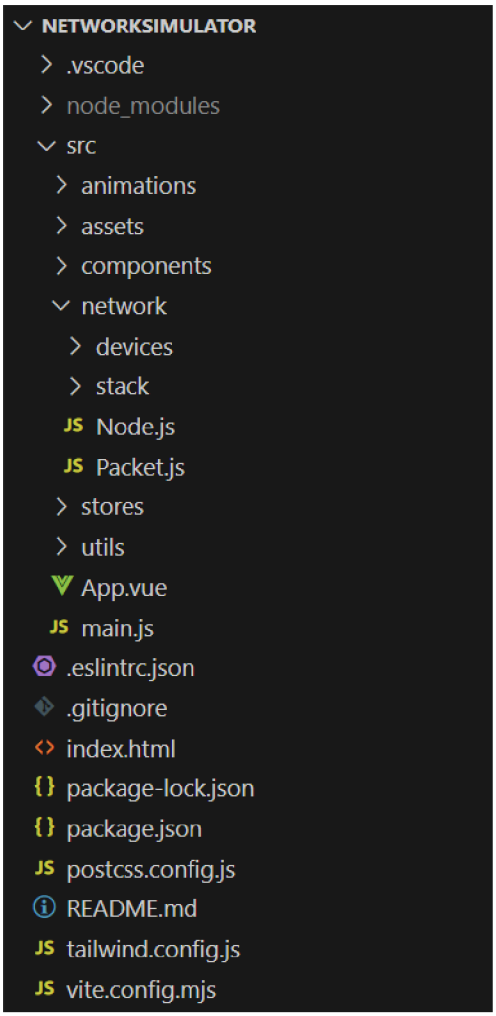
Práci s kolekcemi, čísly a řetězci mi se svojí sadou pomocných funkcí usnadnila utilitní knihovna Lodash [24].

4.4 Struktura projektu

Struktura projektu aplikace je poměrně přímočará (viz obrázek 1). Zajímat nás bude především adresář `/src`, ve kterém se nachází zdrojové soubory. Mezi nimi patří soubor `App.vue`, který reprezentuje kořenovou komponentu připojenou pomocí skriptu `main.js` k danému HTML elementu v `index.html`.[19] Tímto se do něho načte celá aplikace. Adresář je dále tvořen podadresáři s následujícími obsahy:

- `/animations` – animace vytvořené pomocí GSAP,
- `/assets` – obrázky a CSS,
- `/components` – komponenty definující uživatelské rozhraní,
- `/network` – síťová logika a struktury síťových zařízení,
- `/stores` – stavové sklady představené v 4.2.2.1,
- `/utils` – pomocné funkce, struktury.

[19] Typicky se k tomu vyhrazuje element s id `root`.



Obrázek 1: Struktura projektu

4.5 Vizualizace

V této části práce si popíšeme, jakým způsobem bylo využito reaktivního systému Vue.js a funkcí GSAP k tvorbě vizualizace virtuální sítě.

4.5.1 Správa dat

Vizualizované objekty si udržují svůj stav – v případě síťových zařízení je to minimálně pozice, u fyzických linek pak odkazy na zařízení, které propojují.[20] Informace o veškerých síťových zařízeních a jejich fyzických linkách jsou spravovány ve stavovém skladu `DeviceStore`. Ten mimo jiné obsahuje všechny metody pro jejich přidání a odebrání. Ostatní vizualizované objekty jsou řešeny analogicky.

[20] Přesněji, odkazy na fyzické adaptéry propojených zařízení.

4.5.2 Vykreslení objektů

Každému typu objektu vytvoříme komponentu, která ho bude na obrazovce reprezentovat. V rámci komponenty `Device` reprezentující zařízení je použit plugin Draggable, který zachycením myši umožňuje posuny po obrazovce. Komponenta `PhysicalLink` vykresluje úsečku mezi dvěma zařízeními. Při změně pozice některého zařízení, které `PhysicalLink` propojuje, je reaktivním systémem Vue.js linka příslušně překreslena.

Ve zdrojovém kódu 1 můžeme vidět komponentu obrazovky `Board`, ve které mezi značkami `<script>` načítáme vizualizované objekty z příslušných stavových skladů. Přesněji, načítáme kolekce těchto objektů, pomocí nichž pak v části `<template>` direktivou `v-for` komponenty vykreslíme. Kromě zařízení a fyzických linek načítáme i pakety, které si rozebereme v následující části. Dodejme, že níže uvedený zdrojový kód je pro účely demonstrace značně zjednodušen.

```
<script setup>
import { useDeviceStore } from "./stores/DeviceStore";
import { usePacketStore } from "./stores/PacketStore";

const { devices, links } = storeToRefs(useDeviceStore());
const { packets } = storeToRefs(usePacketStore());
</script>

<template>
<div class="board">
  <Device v-for="device in devices"
  :key="device.id"
  :device="device" />

  <PhysicalLink v-for="link in links"
  :key="link.id"
  :link="link" />

  <Packet v-for="packet in packets"
  :key="packet.id"
  :packet="packet" />
</div>
</template>

<style scoped>
.board { background-color: white; }
</style>
```

Zdrojový kód 1: Zjednodušený kód komponenty obrazovky

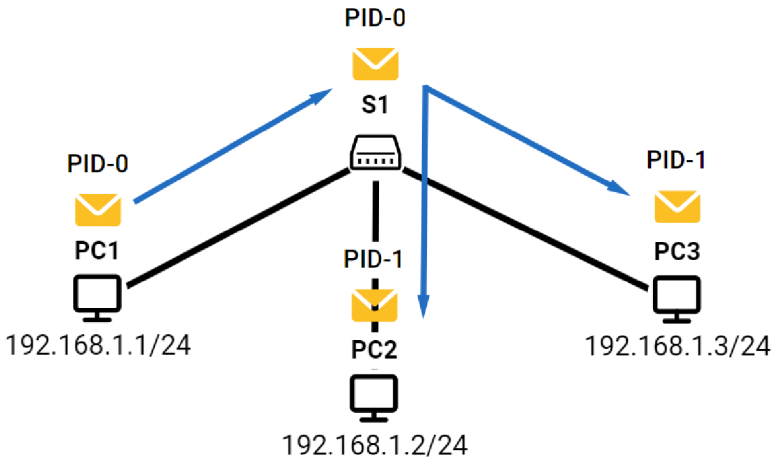
4.5.3 Animace objektů

Nyní umíme vykreslit síťová zařízení a jejich fyzické linky. Zbývá vyřešit, jak vizualizovat komunikaci mezi zařízeními. Pro přenášené zprávy definujeme strukturu a komponentu pojmenované jako `Packet`, dále paket. Pro lepší představu si níže uvedeme zkrácenou verzi konstruktoru paketu (viz kód 2).

```
1  class Packet {
2    constructor(data) {
3      this.id = `p-${nanoid(11)}`;
4      this.data = data;
5
6      this.startPoint = null;
7      this.endPoint = null;
8
9      this.order = 0;
10     this.root = null;
11     this.children = [];
12     this.preceding = [];
13
14     this.transmitted = false;
15     this.eventsBefore = [];
16     this.eventsAfter = [];
17   }
18 }
```

Zdrojový kód 2: Zjednodušený konstruktor struktury Packet

Struktura paketu je vytvářena a předávána vrstvami síťové architektury. Kromě dat a hlaviček s režijními informacemi (atribut `data`) nese paket také informace potřebné k vizualizaci jeho zpracování a přenosu. Celá síťová komunikace je ve své podstatě průchod grafem tvořeným posloupností paketů, které můžeme vnímat jako orientované hrany mezi uzly, jimiž paket prochází. Počátek paketu je dán atributem `startPoint` a konec atributem `endPoint`. V atributu `children` jsou uloženi přímí následníci v posloupnosti. Každý paket je také určen svým pořadím (atribut `order`). V případě prvního (zpravidla kořenový paket `root`) je pořadí rovno 0 a každý přímý následník má vždy pořadí o jedna vyšší. Pakety mohou mít stejná pořadová čísla – například u broadcastu, kdy se vytvoří kopie příchozího paketu s pořadovými čísly o jedna vyšší. Tyto pakety pak mohou být dále přepínány stejným způsobem na dalších uzlech. Pro lepší přehlednost jsou tato pořadí paketů reflektována ve vizualizaci. Výše zmíněný příklad je vyobrazen na obrázku 2.

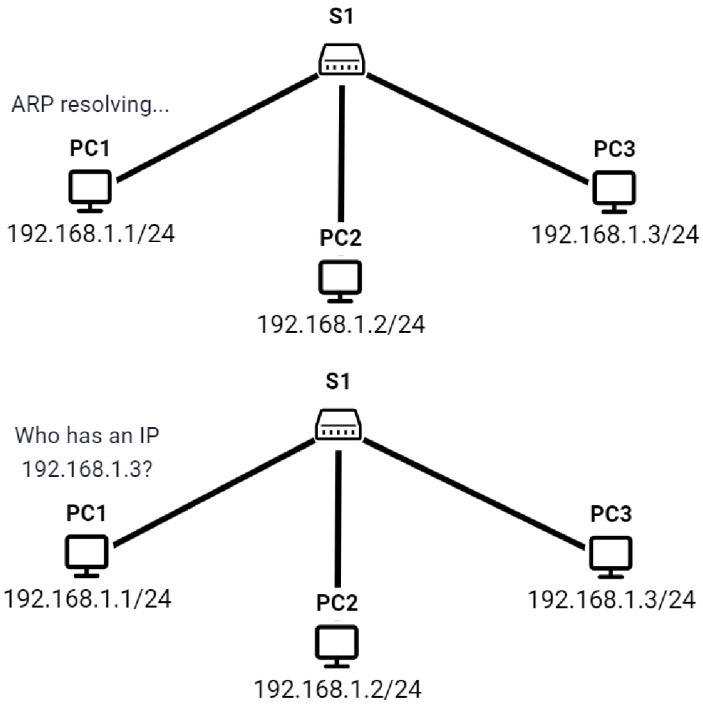


Obrázek 2: Pořadí vizualizovaných paketů

Dále si vysvětleme význam atributu `preceding` (nejde o přechůdce v pravém slova smyslu). Během zpracování paketu se může stát, že bude nutné vyřešit nějaký dotaz odesláním jiných paketů po síti. Příkladem je protokol ARP, kdy se vytvoří nová posloupnost paketů ARP, jejichž pořadí předchází aktuálně zpracovávanému paketu (i v případě kořene). Po průchodu těchto ARP paketů je původnímu paketu přenastaveno pořadí na $h + 1$, kde h je maximální hloubka (pořadí) v průchodu ARP paketů.

Součástí paketu jsou události, které jsou rozlišeny podle toho, jestli proběhly před přenosem `eventsBefore` či po přenosu `eventsAfter`. Struktura obsahuje metodu `report(msg, duration=1)`, která vhodně vloží informativní zprávu do jedné ze zmíněných skupin událostí. Tyto zprávy jsou animovány nad odesílajícím uzlem (případně přijímacím, pokud už k přenosu došlo).[21] Zda událost nastala před přenosem nebo po přenosu, je určeno atributem `transmitted`, jehož hodnota se nastaví přenosem po fyzické lince na `true`. Dodejme, že takových událostí může být více a vizualizaci lze snadno rozšířit o další typy událostí a animací. Příklad informativních zpráv můžeme vidět na obrázku 3.

[21] Voláním těchto funkcí s informativní zprávou v kódu zároveň dokumentujeme, kde se co děje.



Obrázek 3: Animace informativní zprávy

Nyní si popíšeme, jakým způsobem jsou data z paketů využita k animaci. Základem tvorby animací pomocí knihovny GSAP jsou takzvané `Tweens`. Jedná se o funkce, které interpolují atributy libovolných objektů mezi dvěma stavy v určeném časovém úseku. V našem případě jde o animaci objektů DOM. Knihovna GSAP nabízí několik typů tween funkcí: `to`, `from`, `fromTo`. Níže je uveden příklad (viz kód 3), ve kterém animujeme přesun paketu z pozice jednoho uzlu do pozice druhého v průběhu jedné sekundy.

```
function transferMove(packet) {
  return gsap.fromTo(
  `#${packet.id}`,
  {
    x: packet.startPoint.x,
    y: packet.startPoint.y,
  },
  {
    duration: 1,
    x: packet.endPoint.x,
    y: packet.endPoint.y,
  }
  );
}
```

Zdrojový kód 3: Ukázka použití `fromTo`

Každý tween v prvním parametru přijímá selektor HTML elementů, případně jejich přímé reference. Jako selektor vizualizovaných elementů[22] je použito id vygenerované pomocí `nanoid`. V uvedeném příkladě tweenu `fromTo` definujeme počáteční a koncový stav atributů. V případě `to` bychom v druhém argumentu definovali pouze konečný stav, do kterého se má přecházet. Ve `from` bychom opačně definovali počáteční stav, ze kterého se má přecházet.

Knihovna GSAP nabízí pokročilé možnosti skládaní animací (tweenů) do časové osy neboli `timeline`.[23] Timeline sdílí s tweeny podobné rozhraní, a proto můžeme i samotné instance timeline zanořovat. GSAP umožňuje absolutní a relativní pozicování animací v časové ose s podporou značek (`labels`). Tyto značky mohou být libovolně umístěny po časové ose. Asociací přidané animace s některou značkou zajistí její zařazení na příslušnou pozici v časové ose. Tohoto je využito k uspořádání animací paketů. Každý paket je v časové ose umístěn na pozici značky sestávající z `this.root.id + this.id`, tedy řetězce tvořeného z identifikátoru kořenového paketu a jeho vlastního pořadí. Zajistíme tak, že pakety se stejným kořenem a pořadím mají své příslušné animace umístěny na stejné pozici v časové ose.

Uveďme si příklad skládání animací síťové komunikace (viz kód 4). Nejdříve si vytvoříme instanci `timeline`, do které budeme jednotlivé části animace skládat. Funkci `animate` předáme kořenový paket, který chceme animovat, a námi vytvořený timeline. Tato funkce nejprve převede kořenový paket na jednorozměrný seznam paketů (zavoláním `packet.flatten()`). Při tomto převodu je využito průchodu do šířky, což nám zajistí žádané pořadí paketů ve výsledném seznamu.[24] Pro každý z těchto paketů se animují jeho události, které předcházejí samotnému přenosu `animateBefore(p)`, dále se animuje přenos `animateTransmission(p)` a v poslední řadě události po přenosu `animateAfter(p)`.

```
1  function animate(packet, tl) {
2    tl = tl ? tl : gsap.timeline();
3    const packets = packet.flatten();
4
5    packets.forEach((p) => {
6      tl.add(animateBefore(p), `${p.label}-b`);
7      tl.add(animateTransmission(p), p.label);
8      tl.add(animateAfter(p), `${p.label}-a`);
9    });
10
11   return tl;
12 }
```

Zdrojový kód 4: Skládání animací

[22] Přesněji vykreslených komponent paketů.
[23] Podrobněji o pozicování animací na https://gsap.com/resources/position-parameter/.
[24] Pořadí paketů v seznamu odpovídá pořadím animací paketů ve vizualizaci.

Povšimněme si, že animace jsou vloženy do timeline v prvním parametru metody `tl.add()`. Druhým parametrem je značka udávající pozici, kam se má animace v časové ose umístit. Značka událostí se odlišuje od značky, kterou je opatřena animace přenosu, postfixem „-b“ (before) před přenosem, případně postfixem „-a“ (after) u událostí po přenosu. Tímto je zajištěno, že události i přenos paketů se stejným kořenovým paketem a pořadím proběhnou souběžně a ve správném pořadí.

4.5.4 Správa proběhlých komunikací

Veškeré informace o komunikaci jsou obsaženy v kořenovém paketu. Stačí nám tedy si tento paket příslušně uložit – konkrétně do stavového skladu řídící veškerou komunikaci `TransmitStore`. Tento stavový sklad si uchovává informace o komunikujících zařízeních, soketech a poskytuje rozhraní pro různé typy komunikací, jako je ping, navázání a ukončení spojení mezi vybranými komunikujícími zařízeními. Každá z těchto funkcí zahajující komunikaci vrací kořenový paket, který ji zahájil. Tento kořenový paket je pak ve stavovém skladu uložen v historii, ze které ho můžeme opět kdykoliv načíst zavoláním funkce `loadPackets` ze stavového skladu `PacketStore`. Tímto se načtou (funkce `packet.flatten()`) a vykreslí všechny jeho pakety na obrazovku, jak jsme mohli vidět ve zdrojovém kódu 1. V tomto momentě můžeme spustit vizualizaci nad právě vykreslenými pakety. Uložení těchto paketů nám umožňuje se zpětně vrátit a prohlédnout si, jaká výměna zpráv předcházela aktuálním stavům zařízení.

5 Uživatelská dokumentace

V této kapitole si popíšeme zprovoznění aplikace a řešení problému, na který během toho narazíme. Dále se zaměříme na popis uživatelského rozhraní.

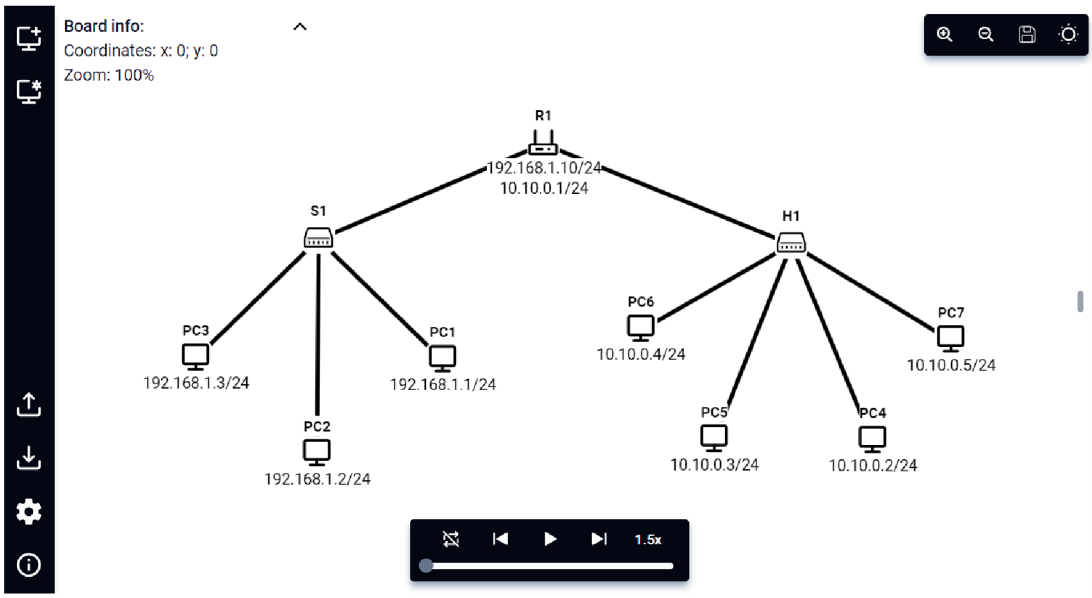
5.1 Zprovoznění aplikace

Výsledná aplikace je tvořena trojicí souborů HTML, CSS a JavaScriptu. Otevřením souboru *index.html* narazíme na výjimku *CORS*[25] [25] při pokusu načíst zdrojový JavaScriptový soubor. Opatření CORS omezuje načítání zdrojů z jiných domén a moderní webové prohlížeče z bezpečnostních důvodů považují i zdroje načtené ze souborového systému za nedůvěryhodné, přestože se nacházejí ve stejném adresáři. Nicméně, pokud nahrajeme zdrojové kódy na libovolný webový server a zajistíme načtení těchto souborů ze stejné domény, pak máme zmíněný problém vyřešený.

Aplikace byla vyvíjena a testována zejména v prostředí webového prohlížeče Google Chrome. Dále byla aplikace testována na prohlížečích Mozilla Firefox, Microsoft Edge a Opera, na kterých se chovala víceméně stejně. Doporučuji však k využívání aplikace používat výhradně Google Chrome. Aplikace byla navržena pro desktop a není optimalizována pro mobilní zařízení.

5.2 Uživatelské rozhraní

Celou obrazovku aplikace můžeme vidět na obrázku 4.



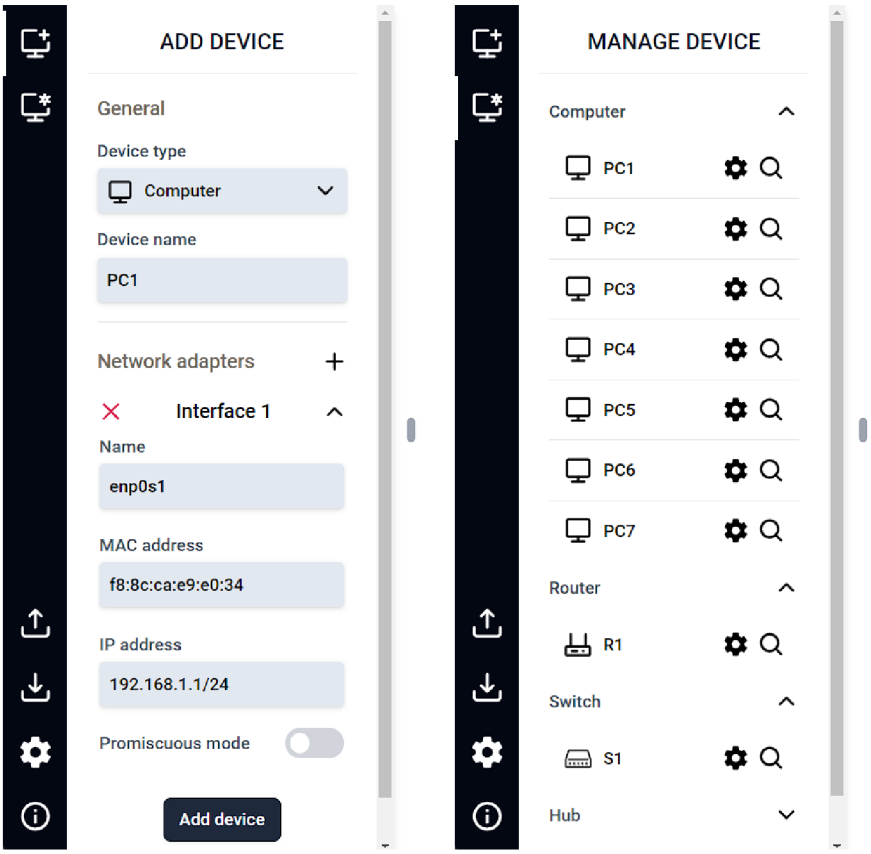
Obrázek 4: Celá obrazovka aplikace

[25] Cross-Origin Resource Sharing.

Na obrázku vidíme uživatelské rozhraní skládající se z levého menu, přehrávače, panelu nástrojů a obrazovky s jednoduchou topologií sítě. Ve zbylých podkapitolách si tyto části uživatelského rozhraní postupně rozebereme. Dále si popíšeme práci se zařízeními včetně toho, jak si podobnou síť sestavit.

5.2.1 Levé postranní menu

Levé postranní menu (viz obrázek 5) je rozděleno na dvě části: horní, které sestává ze sekcí pro správu zařízení, a dolní pro zbylé funkcionality. Níže si sestupně popíšeme jednotlivé možnosti postranního menu.



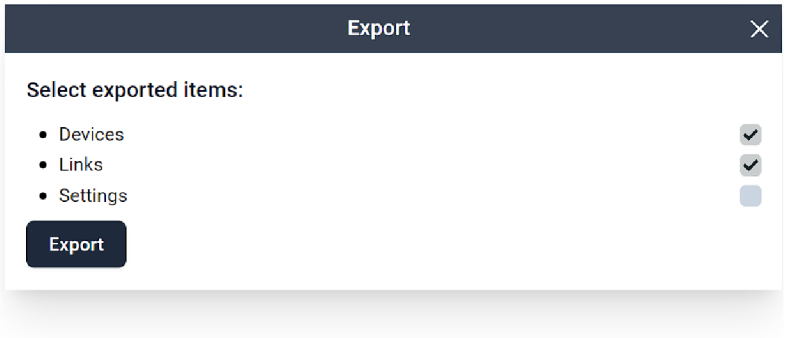
Obrázek 5: Sekce pro přidání a správu zařízení

1. **Přidání zařízení** – V této sekci (viz obrázek 5 vlevo) volíme typ přidávaného síťového zařízení pomocí dropdown menu, který zahrnuje hub, switch, router a koncové zařízení. Zvolenému zařízení nastavujeme jméno a přiřazujeme síťové adaptéry, jejichž počet není pro obecnost omezen. Nově přidanému síťovému adaptéru se náhodně vygeneruje MAC adresa, kterou můžeme dle potřeb změnit. MAC adresy se zapisují po bajtech hexadecimálně a oddělených dvojtečkou. U routerů a koncových zařízení navíc k síťovému adaptéru nastavujeme IP adresu, která je zapsána ve formátu

CIDR. Pokud není délka síťové masky specifikována, je implicitně nastavena maska /32. Síťovému adaptéru můžeme dále nastavit promiskuitní režim. Kliknutím na tlačítko „Add device“ bude nakonfigurované zařízení přidáno a umístěno na obrazovce.

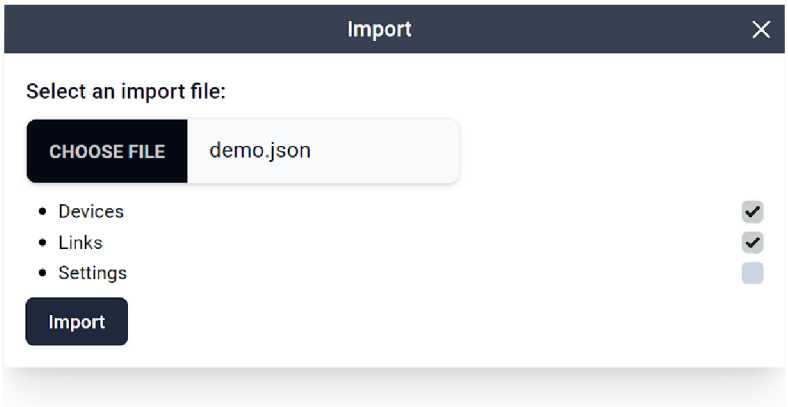
2. **Přehled zařízení** – Sekce (viz obrázek 5 vpravo) poskytuje přehled o všech přidaných zařízeních rozdělených do skupin dle jejich typu. Tento přehled nám umožňuje otevřít nastavení jednotlivých zařízení a snadno je lokalizovat na obrazovce. Tyto funkce jsou zejména užitečné, pokud pracujeme s větším počtem zařízení, která přesahují viditelnou obrazovku.

3. **Export stavu** – Důležitou funkcí je možnost ukládání aplikačních dat ukládaných ve formátu JSON. V okně pro export (viz obrázek 6) můžeme vidět možnosti uložení částí aplikace, jako jsou zařízení, fyzické linky a nastavení aplikace.



Obrázek 6: Okno pro export aplikačních dat

4. **Import stavu** – Exportovaná data lze snadno načíst zpět do aplikace (viz obrázek 7). Kliknutím na tlačítko „CHOOSE FILE“ vybereme exportovaný soubor JSON. Následně se zobrazí požadované části z exportovaných dat, které můžeme načíst, analogicky jako u exportu.



Obrázek 7: Okno pro import aplikačních dat

5. **Nastavení aplikace** – V nastavení aplikace máme možnost uložit veškerý stav aplikace do perzistentního úložiště *localStorage* webového prohlížeče. Existuje také možnost povolit načítání těchto dat z localStorage při otevření aplikace a možnost smazání veškerého stavu. V neposlední řadě je součástí nastavení možnost přepínání mezi tmavým a světlým režimem.

6. **O aplikaci** – V této sekci se nachází informace o vzniku aplikace.

5.2.2 Panel nástrojů

Panel nástrojů, nacházející se v pravém horním rohu, poskytuje rychlý přístup k důležitým funkcím. Panel je znázorněn na obrázku 8.

Obrázek 8: Panel nástrojů

V panelu nástrojů se nacházejí následující funkce:

1. přiblížení obrazovky,
2. oddálení obrazovky,
3. uložení stavu aplikace do localStorage,
4. přepínání tmavého/světlého režimu.

Ukládání stavu aplikace do localStorage a přepínání mezi tmavým a světlým režimem jsou dostupné také v nastavení aplikace.

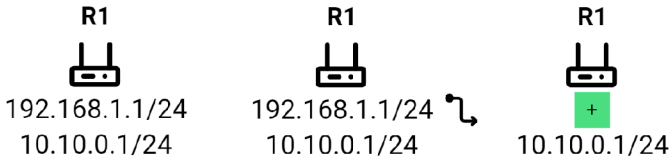
5.2.3 Obrazovka s vizualizací

Na obrazovce jsou vizualizována síťová zařízení, kterými lze rovněž libovolně posouvat. V následující podkapitole se jimi budeme věnovat podrobněji.

5.2.3.1 Zařízení

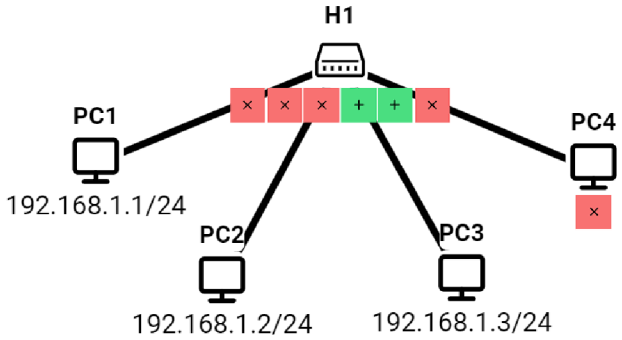
Vizualizovaná zařízení jsou rozlišena svým názvem, ikonou odpovídajícího typu a jednotlivými síťovými adaptéry, které jsou u zařízení pracujících na úrovni vrstvy internetové a výše vypsané pomocí jejich IP adres. U switche nebo hubu, které pracují pod internetovou úrovní, jsou informace o síťových adaptérech skryty a jejich fyzické porty se zobrazí, pokud na ně najedeme myší. Příklad routeru pojmenovaný jako „R1“ se dvěma síťovými adaptéry můžeme vidět na obrázku 9. Vlevo vidíme výchozí zobrazení routeru. Uprostřed je vyobrazeno najetí myší na síťový adaptér s IP adresou „192.168.1.1/24“, při němž se zobrazí ikona

indikující propojení. Vpravo je zobrazen volný fyzický port síťového adaptéru, který se zobrazil po kliknutí na ikonu propojení.[26]



Obrázek 9: Zobrazení fyzického portu na síťovém adaptéru

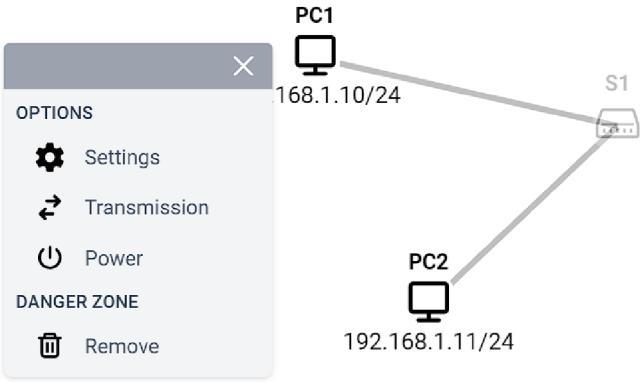
Síťové adaptéry koncových zařízení a routerů mají pouze jeden fyzický port, zatímco switch a hub jich mají šest.[27] Propojení zařízení lze snadno provést výběrem dvou volných portů u vybraných zařízení, čímž se mezi nimi vytvoří fyzická linka. Na obrázku 10 můžeme vidět propojení koncových zařízení s hubem, který disponuje zmíněnými šesti fyzickými porty na síťovém adaptéru. Červené čtverečky s křížkem označují obsazené porty. Odpojení provedeme jednoduše kliknutím na daný fyzický port nebo linku připojenou k portu – v obou případech se zobrazí dialogové okno pro potvrzení akce.



Obrázek 10: Propojení síťových zařízení

Pravým kliknutím na zařízení se otevře menu (viz obrázek 11), ve kterém si můžeme otevřít jeho nastavení, vybrat ho ke komunikaci,[28] vypnout a v neposlední řadě odstranit, přičemž se opět otevře dialogové okno s potvrzením akce. Při odstranění zařízení se odstraní i veškeré jeho fyzické linky. Vypnutá zařízení spolu s fyzickými linkami, které z nich vycházejí, jsou vyobrazena s mírně sníženou viditelností.

[26] Dostupnost fyzického portu je indikovaná zeleným čtvercem se symbolem „+“.
[27] Zvolený počet je jen moje osobní preference a nemá prakticky konkrétní význam.
[28] Otevře se pravý panel s řízením komunikace, ve kterém bude zařízení nastavené jako klient.

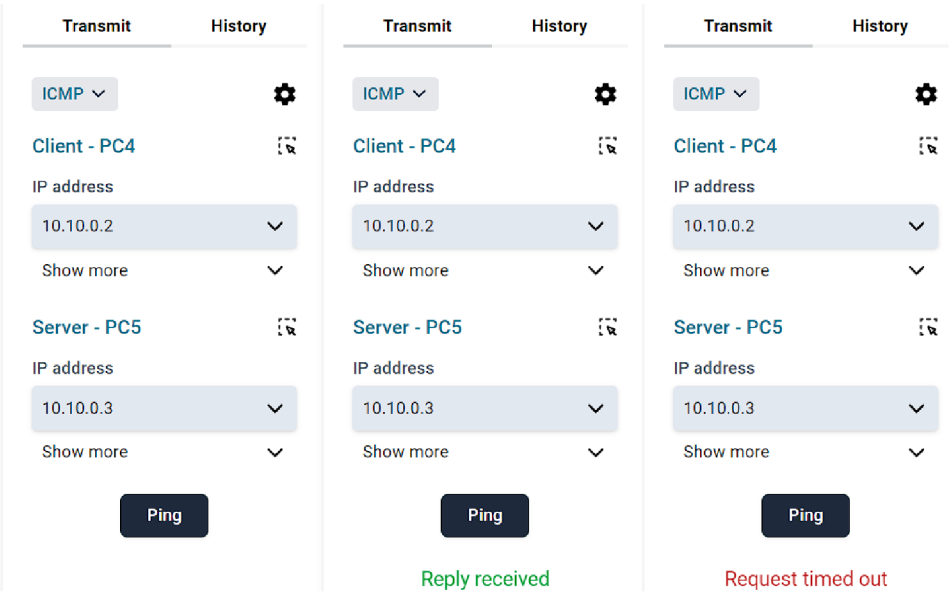


Obrázek 11: Menu síťového zařízení

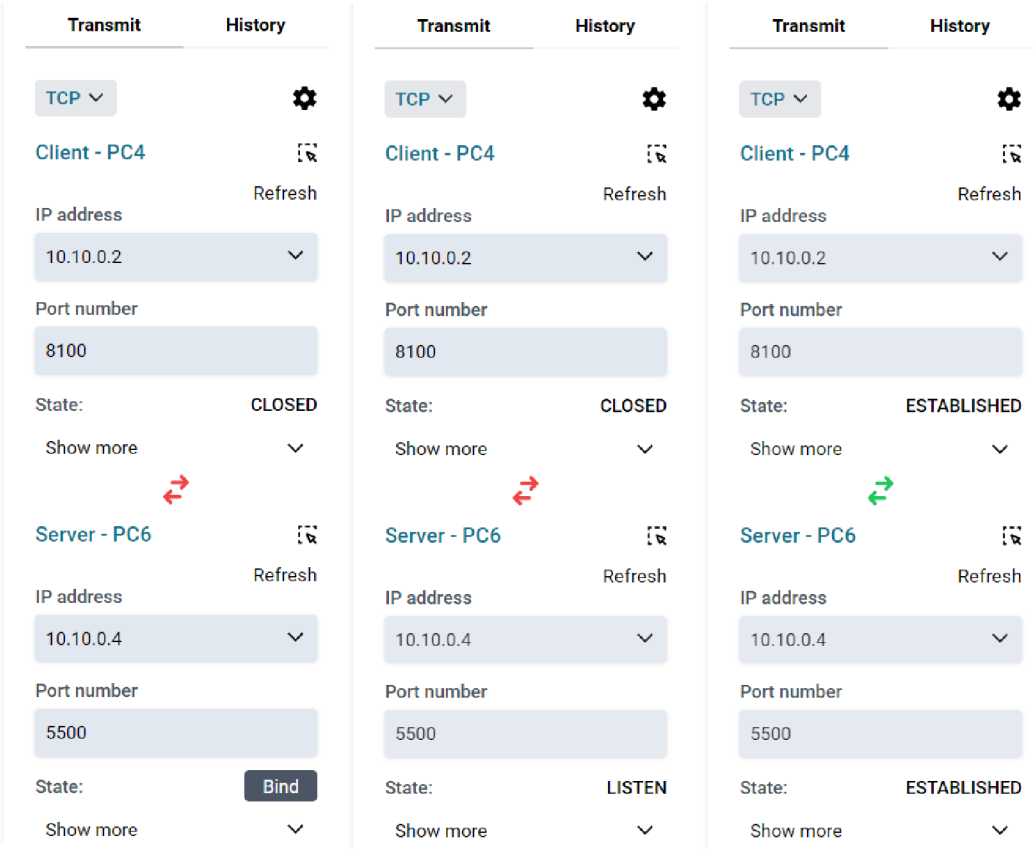
5.2.4 Pravý postranní panel

Kliknutím na tlačítko vpravo uprostřed obrazovky (případně pomocí zmíněného menu vybraného zařízení) se otevře pravý postranní panel sestávající ze dvou sekcí: řízení komunikace a historie komunikace.

1. **Řízení komunikace** – V rámci této sekce vybíráme klienta a server, kteří mezi sebou budou komunikovat. Ke komunikaci vybíráme pouze routery nebo koncová zařízení, nikoliv propojovací prvky, jako jsou switch a hub. K realizaci komunikace máme několik možností – první je nástroj ping, využívající protokol ICMP, kterým můžeme snadno ověřit dosažitelnost klienta a serveru (viz obrázek 12), dále třífázové navázání TCP spojení a třífázové nebo čtyřfázové ukončení (viz obrázek 13). Variantu vizualizovaného TCP ukončení můžeme změnit v nastavení, které otevřeme kliknutím na příslušnou ikonu vpravo nahoře.

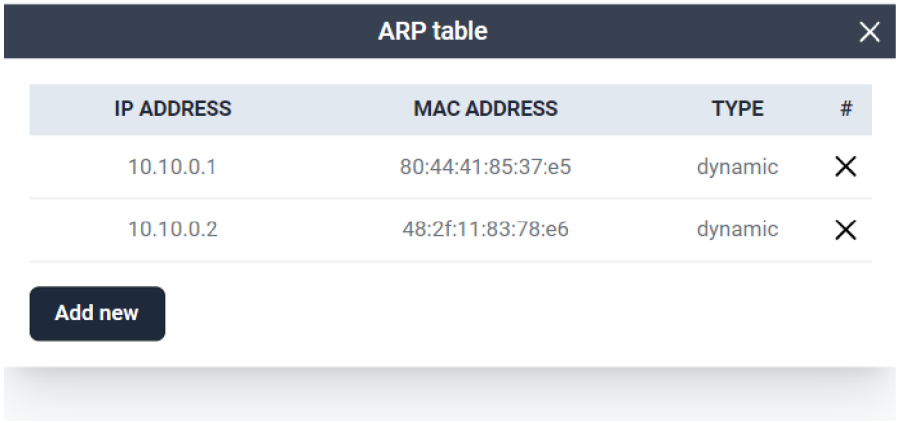


Obrázek 12: Nástroj ping



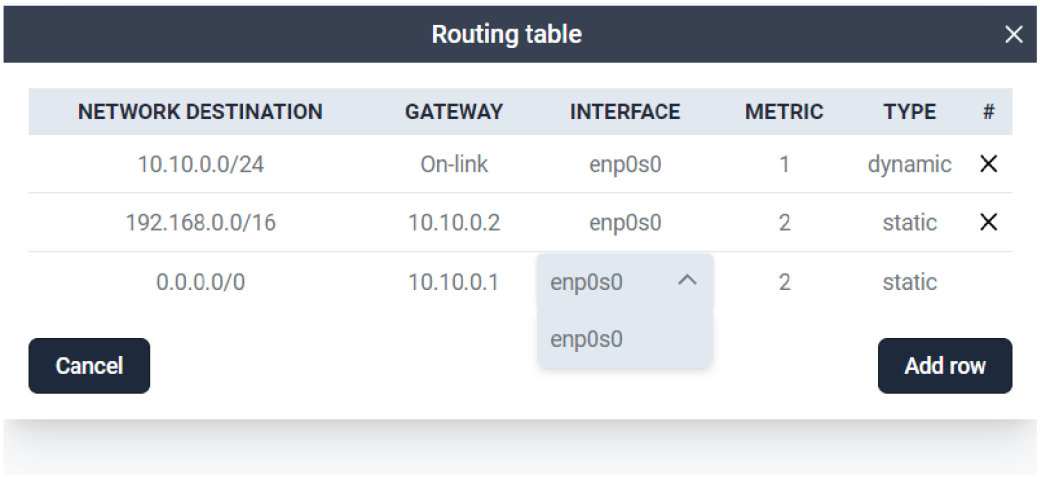
Obrázek 13: Řízení navázání a ukončení TCP spojení

Při výběru klienta či serveru si můžeme zobrazit a měnit obsah směrovacích a ARP tabulek, jakmile zaklikneme tlačítko „Show more“. V ARP tabulce (viz obrázek 14) vidíme dynamické záznamy, které se vytvořily předchozí komunikací. Tabulku lze však také naplnit staticky. Na rozdíl od reálných implementací nemají tyto záznamy expirační dobu, po které by se vymazaly. Záznamy v tabulkách můžeme snadno odebrat pomocí křížku.



Obrázek 14: ARP tabulka

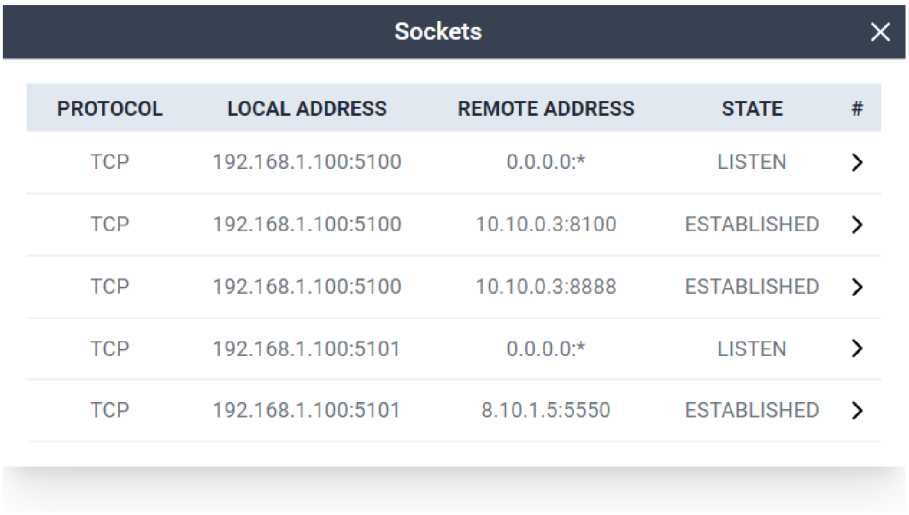
Na obrázku 15 je vyobrazeno nastavování výchozí brány ve směrovací tabulce – záznam s adresou sítě 0.0.0.0/0. Jako gateway je nastavena IP adresa směrovače „10.10.0.1“ a odchozí rozhraní je pak vybráno pomocí dropdown menu. Ve směrovací tabulce jsou vytvářeny dynamické záznamy na základě IP adres a masek síťových rozhraní, díky čemuž nemusíme ručně nastavovat směrovací tabulku pro komunikaci v lokální síti. Příklad dynamického záznamu pro rozhraní „enp0s0“ s IP adresou „10.10.0.10/24“ můžeme vidět v prvním řádku tabulky. Všimněme si, že takový záznam má gateway „On-link“, a paket tak stačí předat příslušnému rozhraní bez nutnosti směrovače.[29]



Obrázek 15: Přidání záznamu do směrovací tabulky

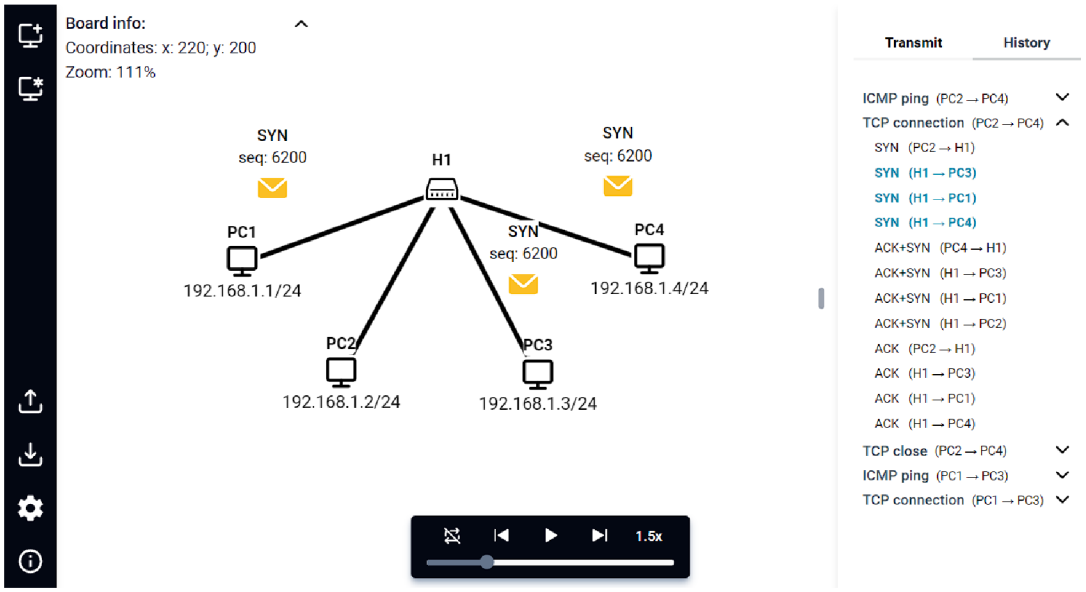
Dále si můžeme zobrazit tabulku soketů TCP spojení. Na obrázku 16 je znázorněna tabulka serveru s adresou 192.168.1.100, který poslouchá a přijímá požadavky ke spojení na portech 5100 a 5101. Tyto poslouchající sokety se vytvoří stisknutím tlačítka „BIND“, které jsme mohli vidět na obrázku 13. Můžeme si všimnout, že jejich vzdálená adresa je nastavena na „0.0.0.0:*“. Tyto sokety mají na starosti vytvoření nových instancí soketů, kterým je předána zodpovědnost obsluhy klienta, v našem případě je můžeme vidět na druhém, třetím a pátém řádku zmíněného obrázku 16. Abychom při řízení komunikace nemuseli manuálně zadávat IP adresy a porty již existujících soketů, je ke každému soketu v záznamu přidruženo tlačítko se symbolem pravé šipky, při jejímž zakliknutí se v sekci příslušně nastaví dvojice klienta a serveru.

[29] Takové rozhraní je připojeno k propojovacímu prvku, jako jsou hub nebo switch, případně jsou komunikující zařízení připojena linkou přímo.



Obrázek 16: Tabulka soketů TCP

2. **Historie komunikace** – Sekce poskytuje přehled proběhlých komunikací. Otevřením některé proběhlé komunikace se zobrazí seznam všech vyměněných paketů, jejichž zakliknutím se spustí jejich vizualizace. Na obrázku 17 je demonstrována vizualizace navázání TCP/IP spojení s přehledem paketů. Během vizualizace se v přehledu tučně zvýrazňují aktuálně přenášené pakety.



Obrázek 17: Vizualizace navázání TCP spojení s přehledem paketů

5.2.5 Další prvky vizualizace

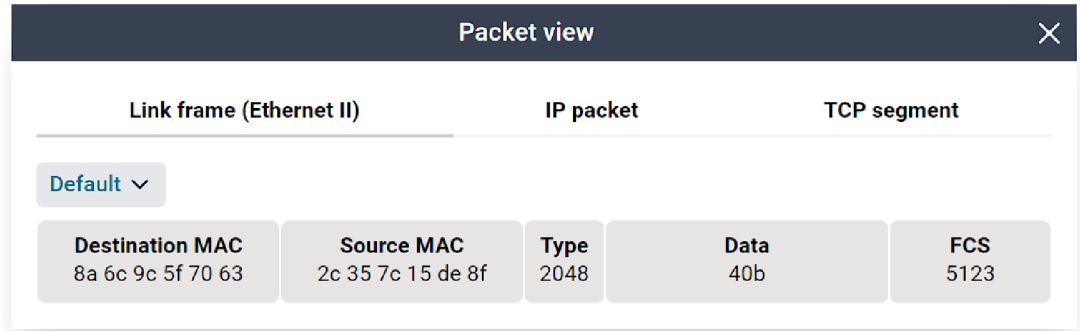
Ve výchozím nastavení se vizualizace spustí automaticky při zahájení komunikace. Toto automatické spouštění můžeme vypnout v nastavení sekce řízení komunikace. Kromě toho zde můžeme upravovat hodnotu TTL v IP paketech, nastavit rychlost přehrávání a přepínat zobrazování informativních zpráv. Zbývající prvky uživatelského rozhraní spojené s vizualizací jsou následující:

- **Přehrávač** – Přehrávač vizualizace (viz obrázek 18) umožňuje její opakování, krokování vzad, spuštění/zastavení, krokování vpřed a změnu rychlosti. Kroky jsou dány pořadím přenášených paketů. Součástí přehrávače je také *progress bar*, kterým můžeme vizualizaci libovolně posouvat.

Obrázek 18: Přehrávač vizualizace

- **Náhled paketu** – Zakliknutím vizualizovaného paketu na obrazovce se otevře okno s jeho obsahem. Na obrázku 19 vidíme linkový rámec, který zapouzdřuje IP paket a TCP segment. Můžeme si přitom povšimnout poněkud matoucího výpisu, kdy jsou MAC adresy uvedeny hexadecimálně (po bajtech), zatímco typ a kontrolní součet (FCS) decimálně. Důvodem je interní uložení dat hlaviček jako binárního pole.[30] Otázkou je, jak uživateli tyto hodnoty položek jednotně a srozumitelně zobrazit. Například šestibajtové MAC adresy jsme zvyklí číst po bajtech hexadecimálně a čtyřbajtové IP adresy taktéž po bajtech, ale decimálně. Hodnoty položek o čtyřech bajtech a méně jsou kvůli čitelnosti lepší zapisovat jednou decimální hodnotou. Ve výchozím nastavení jsou proto hodnoty položek zobrazeny výše uvedeným způsobem. Pro úplnost si však v náhledu můžeme formát výpisu jednotně změnit do decimálního, binárního či hexadecimálního zápisu (po bajtech).

[30] Takové uložení má své výhody, neboť více odpovídá tomu, jak jsou data reálně přenášena. Usnadní nám to do budoucna implementaci různých algoritmů pro manipulaci s daty v hlavičkách paketů, jako je například výpočet kontrolních součtů.



Obrázek 19: Náhled vizualizovaného paketu

6 Možná rozšíření

S ohledem na množství protokolů v síťové architektuře TCP/IP byla vybrána a vizualizována pouze její část, která se mi zdála pro pochopení této architektury nejpodstatnější. Nicméně, i opomenuté, ale neméně důležité protokoly mají v soudobém Internetu nezastupitelnou roli. Kromě těchto protokolů se následovně zmíníme také o funkčních vylepšeních, která by mohla používání aplikace zpříjemnit.

- **Aplikační vrstva** – Pro základní fungování počítačové sítě se implementace aplikační vrstvy nezdála být nezbytná, ovšem z pohledu uživatele by mohla být velmi zajímavá. Které aplikační protokoly implementovat a vizualizovat nechávám ještě na zvážení, zajímavé by však mohly být protokoly HTTP, DNS či DHCP.
- **IPv6** – Ačkoliv je IPv4 stále široce používán a vyučován, nasazení IPv6 se v budoucnu nevyhneme. Pokud by se aplikace osvědčila, mohlo by být rozšíření o protokol IPv6 nezbytné.
- **NAT** – Protokol NAT je v době, kdy je stále široce nasazen IPv4, nedílnou součástí fungování Internetu. Protokol mapuje lokální a veřejné adresy, čehož se využívá především k předcházení nedostatku IPv4 adres. Použití NAT je ale poněkud kontroverzní. Porušuje princip koncové *end-to-end* komunikace v Internetu, pro kterou byl původně navržen. V případě úplného přechodu na IPv6 by protokol NAT mohl svůj význam ztratit. Ponechávám tedy na zvážení, zda má jeho implementace smysl.
- **Spanning Tree Protocol** – Jedná se o protokol na linkové vrstvě, který zamezuje nekonečnému přenosu linkových rámců v cyklických topologiích sítí. V aktuální implementaci není mechanismus, který by takovému nechtěnému jevu plnohodnotně zabraňoval. Tento protokol by byl důležitý

nejen ke správnému chování sítě, ale také zajímavý z výukového pohledu. Jeho implementace a vizualizace by však mohly být poměrně náročné.

- **Simulace porušených paketů** – Simulace poškození dat může být složitější, než se na první pohled zdá. Jednou z možností realizace je porušení přenášených dat realizovat náhodně. Nedeterministický přístup však může vést k nepřehledným a obtížně reprodukovatelným výsledkům. Druhou možností je dát uživateli kontrolu nad výběrem poškozených paketů. Tato možnost je problematická v tom, že uživatel nemůže dopředu vědět, co se v komunikaci stane. Přesto by mohla být tato možnost užitečná při demonstraci protokolu TCP.

- **Uživatelské rozhraní** – Srozumitelnost uživatelského rozhraní je důležitou vlastností každé dobré aplikace. Avšak kvůli nedostatku zkušeností s jeho návrhem může moje aplikace místy trpět nedostatky, jejichž vylepšení by mohla výrazně zvýšit její uživatelskou přívětivost.

- **Undo/Redo funkce** – Umožnit přecházení dozadu-dopředu udržováním provedených akcí by mohlo velmi zpříjemnit především sestavování a konfiguraci sítě.

- **Účast serveru** – Zajímavé by mohlo být zadávání úkolů, jejichž řešení by byly validována vůči serveru. Server by si vedl záznamy o studentech a jejich postupech.

Závěr

V úvodu práce jsme si stručně nastínili problémy ve výuce počítačových sítí, její náročnost, a jak by mohla moje aplikace při výuce studium usnadnit. Při analýze existujících řešení jsem narazil na různé nedostatky, zejména na jejich přílišnou technickou zaměřenost a náročnost zprovoznění. Výsledkem mé práce je proto jednoduchý simulátor počítačové sítě, který má za cíl těmto problémům předejít.

Mnou zvolené řešení, kterým je webová Single-Page aplikace, odstraňuje platformovou závislost a nutnost složitého zprovoznění. Při vývoji aplikace jsem se snažil klást důraz na jednoduchost a přívětivost jejího používání. Aplikace umožňuje vytvoření jednoduché topologie sítě ze základních síťových prvků, pomocí kterých lze demonstrovat jejich rozdíly. Kromě toho by aplikace mohla také pomoci při procvičování statického směrování, které je díky vizualizaci usnadněno. K zachování jednoduchosti jsem se však musel omezit pouze na malou část síťové komunikace TCP/IP, která se zdála být pro pochopení základů nejdůležitější. Aplikace i přes tato omezení poměrně dobře demonstruje složitost síťové komunikace, ač se může na první pohled zdát banální. Vzhledem k množství protokolů, které jsem musel z důvodu náročnosti vynechat, je aplikace otevřená k rozšíření, při kterém se budu rád dále podílet.

Conclusions

In the introduction, we briefly discussed the challenges which students may face while studying Computer Networks and how my application could assist them in their studies. During the analysis of existing solutions, I encountered various shortcomings, especially their excessive technical focus and the difficulty of setup.

The solution I chose, a web Single-Page application, eliminates platform dependency and the need for complex deployment. During the development of the application, I focused on simplicity and user-friendliness. The application allows for the creation of a simple network topology from basic network devices which can be used to demonstrate their differences. To maintain simplicity however, I had to limit myself to a small portion of TCP/IP network communication which seemed to be the most important for understanding the basics of computer networks. Despite these limitations, the application demonstrates the complexity of seemingly trivial network communication such as ping fairly well. As many protocols were omitted due to their technical complexity, this application is open to extension, to which I will gladly contribute in the future.

A Obsah elektronických dat

Elektronická data odevzdaná v systému KI PřF UP v Olomouci obsahují veškeré soubory k textu práce, včetně zdrojových kódů potřebných ke zprovoznění aplikace. Struktura elektronické přílohy je následující:

`README.txt`
: Textový soubor obsahující postup ke zprovoznění aplikace.

`src/`
: Adresář se zdrojovými kódy potřebnými ke zprovoznění aplikace.

`exports/`
: V tomto adresáři jsou obsažena exportovaná data aplikace včetně přednastavených topologií virtuální sítě, které si lze v aplikaci načíst a využít k testování.

`text/`
: Adresář obsahuje text práce ve formátu PDF, který byl vytvořený použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce. Zahrnuje také všechny (textové) přílohy a soubory potřebné pro bezproblémové vytvoření PDF dokumentu (v ZIP archivu), tj. zdrojový kód textu a vložené obrázky.

Literatura

[1] Oracle Corporation. *VirtualBox* [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://www.virtualbox.org/⟩.

[2] Wireshark Foundation. *Wireshark* [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://www.wireshark.org/⟩.

[3] *Cisco Packet Tracer*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://www.netacad.com/courses/packet-tracer⟩.

[4] Cisco Systems, Inc. *Cisco Networking Academy* [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://www.netacad.com/⟩.

[5] *GNS3*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://www.gns3.com/⟩.

[6] James F. Kurose, Keith W. Ross. *Počítačové sítě*. 1st. 2014. ISBN 978-80-251-3825-0.

[7] *Single Page Application*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Glossary/SPA⟩.

[8] WHATWG. *HTML – HyperText Markup Language* [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://html.spec.whatwg.org/⟩.

[9] *Cascading Style Sheets (CSS)*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/CSS⟩.

[10] *JavaScript*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/JavaScript⟩.

[11] *Document Object Model (DOM)*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model⟩.

[12] Adam Wathan. *Tailwind CSS* [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://tailwindcss.com/⟩.

[13] *Daisy UI*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://daisyui.com/⟩.

[14] Evan You. *Vue.js* [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://vuejs.org/⟩.

[15] Facebook, Inc. *React* [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://reactjs.org/⟩.

[16] *Angular*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://angular.io/⟩.

[17] Eduardo San Martin Morote. *Pinia* [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://pinia.vuejs.org/⟩.

[18] *GreenSock Animation Platform (GSAP)*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://gsap.com/⟩.

[19] *Canvas API*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API⟩.

[20] *Motion One*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://motion.dev/⟩.

[21] *GSAP – JavaScript Animation Speed Test*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://gsap.com/js/speed.html⟩.

[22] *Panzoom*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://github.com/anvaka/panzoom⟩.

[23] *Nanoid*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://github.com/ai/nanoid⟩.

[24] *Lodash*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://lodash.com/⟩.

[25] *CORS*. [online]. [cit. 2024-3-21]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS⟩.