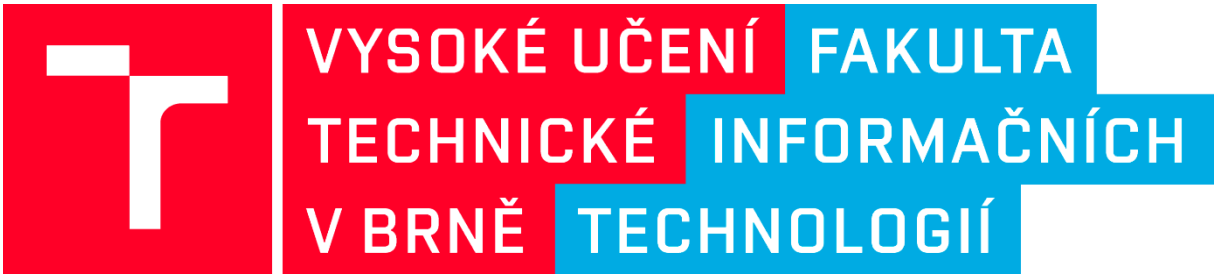
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Dokumentace k projektu IMS

Simulace toku dat v systému pro zpracování rádiových časoměrných signálů s využitím v pozičním systému

(téma 6: Počítačové služby)

3.12.2017

Autoři: Jan Ondruch, Richard Činčura

Faktulta Informačních Technologií

Vysoké Učení Technické v Brně

Obsah

1. Úvod

V této práci je řešena implementace systému (1 str. 7) pro zpracování rádiových časoměrných signálů s využitím v pozičním systému, která bude použita pro sestavení modelu (1 str. 7) simulujícího (1 str. 8) tok dat v dané síti od zdroje rádiových signálů až k vizuální reprezentaci získaných dat koncovému uživateli.

Cílem experimentů (1 str. 8) je demonstrovat závislost jednotlivých prvků systému na rychlosti a kvalitě zobrazování dat v reálném čase při akvizici informací z basketbalového utkání. Zjistíme také, zda-li je systém k tomuto účelu vůbec vhodný. Dále se zaměříme na velikost databáze uchovávající historická data, tedy kolik dat je do databáze systém schopen uložit. Posledním cílem je případná možnost detekce míst s nedostatečnou průchodností, která by bylo možno optimalizovat.

1.1 Autoři a zdroje

Autoři práce:

- Jan Ondruch, xondru14
- Richard Činčura, xcincu00 - nepodílel se na projektu (ukončil studium)

Údaje o architektuře systému byly získány od pana ing. Lubomíra Mráze z firmy Sewio[1], která daný systém implementuje. S panem Mrázem byla následně koncepce navrženého modelu konzultována. Další detailější informace byly získány z webových stránek firmy Sewio a také z technické dokumentace jednotlivých prvků systému (1 str. 18), která nám byla poskytnuta.

1.2 Validita modelu

Při spuštění simulace modelu bez parametrů příkazového řádku je modalována událost IBM Watson Summit Prague 2017[2] během které byl poziční systém reálně demonstrován. Získaný počet dat, jejich celková velikost a přenosová rychlost se v porovnání s reálnými údaji z oné události liší přibližně o 2%. Při modelování (1 str. 8) jiných událostí se výstupní hodnoty mohou s hodnotami referenčními lišit o méně než 1%. Model tedy můžeme považovat za validní (1 str. 37). Důvodem nepřesností je nedokonalý model serverové části spolu s aplikováním heuristik a aproximací spojených s časovou náročností algoritmu pro výpočet dat z rádiových signálů.

2. Rozbor tématu a použitých metod/technologií

Proces zpracování rádiových časoměrných signálů a jejich následné využití v pozičním systému sestává z několika částí, které jsou vzájemně provázány a tvoří tak celý systém od počátečního vysílání rádiových signálů až po jejich vizuální reprezentaci koncovému uživateli. Jednotlivými částmi jsou pohybové senzory, přijímače signálu, počítačová síť a server.

Pohybové senzory a přijímače signálu

Počátečními vstupy do systému jsou ultra-širokopásmové rádiové časoměrné signály vysílané v pravidelných intervalech z pohybových senzorů[3]. Obnovovací frekvence je nastavitelná a může se pohybovat v rozmezí od 50ms-1s[4] . 80ms se standardně využívá pro sport, 100-300ms pro průmyslové účely a 300ms-1s v maloobchodech[5]. Vysílané signály jsou poté zachyceny příjmači, které jsou

[1] Sewio [online]. Dostupné z: https://www.sewio.net/.
[2] IBM Watson Summit and IoT by Sewio. Dostupné z: https://www.sewio.net/portfolio-items/ibm-watson-summit-and-iot-by-sewio/.
[3] Sewio RTLS Platform. Dostupné z: https://www.sewio.net/rtls-tdoa/.
[4] Tag Piccolino. Dostupné z: https://www.sewio.net/product/piccolino-tag/.
[5] API Guide. Dostupné z: http://rtlsstudio.com/studio/wp-content/uploads/2017/09/AN07_API-Guide-v1.1.pdf.

pozicovány ve vzdálenosti 15-50m od pohybových senzorů[6]. Aby bylo možné jednotlivé pohybové senzory dále identifikovat, je nutné v systému nainstalovat alespoň 3 přijímače. Ve skutečnosti však firma doporučuje použít prijímačů 6 kvůli redundanci a optimálnějšímu zpracování. Tato zařízení mohou být připevněna na zdech, na stropě nebo na speciálních stojanech. Pro jejich správnou funkčnost je nutno při jejich instalaci nastavit jejich pravidelnou synchronizaci. Proces spočívá ve vybrání jednoho z přijímačů jako master, podle kterého se v pravidelných intervalech ostatní přijímače synchronizují[7]. Délka synchronizace trvá řádově desítky, maximálně stovky mikrosekund několikrát do minuty (Sewio konstatovalo, že jsou intervaly proměnné a závisí na přesnosti příchozích dat vyhodnocovaných na serveru). Při přenosu signálu z pohybového senzoru se může stát, že signál není zachycen a paket se tak ztratí. Tato situace nastane u maximálně 1% vysílaných paketů a vyjadřuje hodnotu Packet Error Rate[8].

Počítačová síť

Přijímače dále přeposílají časová razítka, reprezentující vzdálenost pohybového senzoru od přijímače, z rádiových signálů po Ethernetu nebo Wi-Fi do RTLS serveru[9]. Data mohou být vysílána buď použitím TCP, nebo UDP komunikačního protokolu. Pokud se jedná o přenos po Ethernetu, využívá se datových síťových kabelů kombinovaných s technologií Power over Ethernet (PoE). Ta umožňuje jejich přímé napájení bez nutnosti přívodu napájení dalším kabelem. Namísto PoE může být také využito technologie Mini USB. Dále se v síti nachází síťový přepínač, na nějž jsou jednotlivé přijímače napojeny. Výstup je poté již přímo napojen na RTLS server. Sewio odpozorovalo, že délka přenosu jednoho paketu po Ethernetu může být popsána normálním rozložením pravděpodobnosti se středem 1ms (1 str. 93). Přenos přes Wi-Fi je pomalejší a odpovídá rychlosti časové odezvy podle normálního rozložení pravděpodobnosti. Směrodatné odchylky (1 str. 82) reprezentují kolísání velikosti zpoždění paketů při průchodu sítí[10]. Během testování různých scénářů lišících se v počtu pohybových senzorů a počtu přijímačů bylo odpozorováno, že rychlost přenosu přes Ethernet či Wi-Fi se mění jen minimálně a k žádným výraznějším zpožděním nebo zahlcením nedochází.

RTLS server

RTLS server (Real-Time Location System server) je počítač řídící celý systém. Sewio používá dvoujádrový 64 bitový procesor x86_64 s frekvencí 2.3 GHz. Operační pamět má velikost 4 GB, operačním systémem je Ubuntu 14.04.5 LTS 4.4.0-31-generic[11]. V systému je nastaveno 18 Cron softwarových démonů, které jsou spuštěny v intervalech každých 6:25 h. Pokud se do serveru neposílají žádná data z pohybových senzorů, pak je procesor je průměrně využit přibližně z 23% procesorového času a operační pamět z 18% maximální kapacity[12].

Na serveru je stále spuštěných 14 procesů. Patří mezi ně např. mysql, apache2, nodejs nebo php. Tyto procesy zajišťují funkčnost 4 hlavních aplikací, které server poskytuje. Jedná se o rtlsmanager, rtlsserver, sensmap a sensmapserver. Rtlsmanager je webová aplikace pro správu systému a nastavení jednotlivých prvků, které obsahuje. Je zde možno spravovat synchronizaci přijímačů, zabezpečení či IP adresy zařízení. Rtlsserver spravuje aplikaci, která má na vstupu časové známky z pohyhových senzorů, které dále přepočítá a na výstup posílá již zpracovaná data, která se ukládají do databáze, popř. se mohou posílat koncovému uživateli. Aplikace vykonávající výpočty je program napsaný v programovacím jazyce C a jedná se o nejkomplikovanější část systému. Samotná konverze příchozích

[6] Anchor Router Cube. Dostupné z: https://www.sewio.net/product/anchor/.
[7] Time Difference of Arrival: Dostupné z: https://www.sewio.net/technology/time-difference-of-arrival/.
[8] Packet Error Rate (PER). Dostupné z: http://rfmw.em.keysight.com/rfcomms/refdocs/1xevdo/1xevdo_meas_cperror_desc.html.
[9] API Connectors Overview: Dostupné z: http://rtlsstudio.com/studio/index.php/api-connectors-overview/.
[10] Understanding Jitter in Packet Voice Networks. Dostupné z: https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/18902-jitter-packet-voice.html.
[11] Basic info. Dostupné z: http://rtlsstudio.com/studio/index.php/status/#/system-status.
[12] System status. Dostupné z: http://rtlsstudio.com/studio/index.php/status/#/system-status.

časových známek na člověkem již interpretovatelná data (pozice x, pozice y) je prováděna pomocí hyperbolické multilaterace - techniky též známou jako TDoA (Time Differential of Arrival). Během tohoto procesu se dle příchozích časových známek zjistí, o který paket z pohybového senzoru se jedná. Ty se stejnou časovou známkou se poté seskupí a vypočítá se pro ně pozice daného senzoru. Dochází zde také k řadě optimalizací, kdy se jednotlivé pakety pro výběr optimální pozice a urychlení výpočtu vzájemně vylučují, nekvalitní signály s příliš velkými odchylkami se zahazují apod. Výstupem výpočtu jsou data, která můžeme během následujícího procesu poslat přímo uživateli a uložit je do databáze. Třetí zmiňovanou aplikací je sensmap, což je webová aplikace, která v reálném čase (1 str. 21) zobrazuje pozice pohybových senzorů. Posledním softwarem je senmapserver, který má 2 hlavní funkce. První funkcí je přenos vypočítaných dat uživatel, který se k odběru dat připojí na daném portu. Data jsou poté posílána 3 možnými způsoby - REST API, Websockets nebo UDP. REST API a Websockets pro přenos dat využívají komunikační protokol TCP. Tyto 2 metody se od sebe liší v tom, že REST API přenáší podrobnější informace o daném záznamu. Websockets a UDP poté obsahují stejné množství dat. Vzhledem k použití protokolu TCP/UDP a množství přenesených dat se přenosová rychlost liší. Z informací dostupných v dokumentaci Sewio[13] a z konzultací s panem Mrázem byl čas přenosu pro REST API určen jako hodnota definována normálním rozložením pravděpodobnosti se středem 450ms a směrodatnou odchylkou 15ms, pro Websockets 200ms s odchylkou 10ms a UDP 75ms s odchylkou 5ms. Pro UDP navíc předpokládejme chybovost 0.01%, se kterou se koncový uživatel musí vypořádat sám. Při vyslání dat ze serveru se paket nejprve pošle do výstupního bufferu, ze kterého je pak poslán klientovi. Sewio zjistilo, že poslání jednoho záznamu na výstupní buffer a jeho zařazení na vstupní buffer databáze trvá přibližně 0.1ms. Druhou funkcí aplikace sensmapserver je ukládaní dat do databáze. Pro zvýšení efektivity systému se data ukládají až tehdy, pokud jich je na vstupním bufferu databáze nashromážděných alespoň 200. Poté se automaticky všech 200 záznamů přesune do databáze. Tato operace trvá přibližně 10ms. RTLS server využívá pevný disk o velikost 1TB, přičemž 913GB je vyhrazeno pro databázi a zbytek pro ostatní účely. Velikost jednoho záznamu v databázi je 255 byte[14].

Výchozí modelovaná situace

Jako výchozí situace je v simulačním modelu (1 str. 9) modelována událost zmíněná v 1. Událost trvala od 9:00 do 15:00[15], avšak Sewio konstatovalo, že sada 50 kusů pohybových senzorů byla aktivována až během části networkingu, který se namísto plánované 1 hodiny protáhnul a senzory vysílaly přibližně 100 min. Obnovovací frekvence byla nastavena na 500ms. Použito bylo 6 přijímačů. Data z nich byla do RTLS serveru vysílána přes Wi-Fi s latencí 15ms. Rychlost přenosu byla tedy odhadnuta na rychlost určená podle normálního rozložení pravděpodobnosti se středem 15ms a odchylkou 1ms reprezentující velikost zpoždění paketů při průchodu sítí. Během akvizice dat bylo nashromážděno 502325 záznamů, které byly uloženy do databáze. Celková velikost záznamů činila 128MB.

2.1 Použité postupy

Model byl implementovaný v programovacím jazyce C++ a knihovny SIMLIB (2). Jazyk byl zvolený z důvodu jeho rychlosti, spolehlivosti, nezávislosti na platformě a také rozšířitelnosti o externí knihovny, jakou je právě např. SIMLIB. SIMLIB obsahuje mnoho tříd a metod vhodných pro implementaci simulačních modelů diskrétních systémů (1 str. 120). Pro snadnější práci s pamětí a zajištění správného managementu zdrojů byl využit hlavičkový soubor memory[16].

2.2 Původ použitých metod/technologií

Použité metody a třídy byly poskytnuty knihovnou SIMLIB. Jedná se o třídy Facility (1 str. 163), Process (1 str. 163), Queue (1 str. 163), Event (1 str. 163) a Histogram (1 str. 196). Histogram je třída vhodná pro

[13] API Guide. Dostupné z: http://rtlsstudio.com/studio/wp-content/uploads/2017/09/AN07_API-Guide-v1.1.pdf.
[14] Database Diagrams. Dostupné z: http://rtlsstudio.com/studio/index.php/api-database/.
[15] Program. Dostupné z: http://www-05.ibm.com/cz/watson-prague-summit/agenda.html.
[16] Hlavičkový soubor memory. Dostupné z http://en.cppreference.com/w/cpp/header/memory.

sběr statistik (1 str. 195) a jejich následné zobrazení. Použité konstrukce a algoritmy byly převzaty z výukových materiálů předmětu IMS (1 stránky 119-207).

3. Koncepce

Simulační model modeluje systém v době, kdy jsou pohybové senzory aktivní a vysílají rádiové signály. Modelový čas (1 str. 21) ubíhá v milisekundách. Vzhledem k rychlosti toku dat vůči výsledným hodnotám, které se pohybují v řádech stovek ms je přenos dat z pohybových senzorů do RTLS serveru v simulačním modelu zjednodušen. Zpoždění několika mikrosekund při synchronizaci přijímačů s masterem zanedbáváme. V modelu tak můžeme zcela vynechat přijímače, u nichž se zpoždění příjmu a vyslání paketu pohybuje v řádech desítek mikrosekund. Tímto zjednodušením se v nejhorším případě dopustíme chyby menší než 0.5% (nastalo by tak při využití stovek pohybových senzorů, kdy dle zkušeností Sewio může u přijímačů nastat prodleva přibližně 1ms). K zjednodušení modelu dále v architektuře nastalo u zanedbání PoE splitterů a síťového rozdělovače. Namísto nich budeme v modelu uvažovat celkové zpoždění přenosu paketu přes Ethernet, nebo Wi-Fi, které bylo popsáno v Rozbor tématu a použitých metod/technologií.

V reálném systému (1 str. 7) exitustuje dvoujádrový procesor podporující multitasking a multithreading. V konceptuálním modelu (1 str. 48) byla koncepce procesoru a všech operací prováděných v počítači výrazně zjednodušena. Procesor je modelován jako jednotka podporující pouze multitasking a zanedbává možnost paralelního zpracování výpočtů a procesů. Vzhledem k časové náročnosti výpočtů TDoA dává tento konceptuální model procesoru smysl pouze u počtu přijímačů odhadem nižšího než 10. Předpokládá se, že při vyšším počtu přijímačů by simulační model nestíhal požadavky pro výpočet TDoA zpracovávat. V reálných systémech se obvykle používá přijímačů 6, pokud jich je více, mění se také architektura serveru a dvoujádrový procesor je nahrazen výkonějším procesorem. Taktéž je navýšena velikost operační paměti a diskového prostoru pro databázi. Sewio tyto rozsáhlejší systému používá v supermarketech. Tyto případy podrobněji studovat nebudeme. Tento konceptuální model také nemodeluje přístupy do operační paměti, diskové operace a další manipulace s daty. Prodlevy těchto operací jsou již zahrnuty do výpočetních časů výpočtů u jednotlivých aplikací. Multitasking je modelován tak, že každé ze 4 hlavních aplikací je přidělen čas 10ms. Po jeho vypršení se případně dokončí prováděný výpočet či operace a poté je procesor přidělen jinému z procesů. V 10% případů procesor získá aplikace rtlsmanager, sensmap jej získá se stejnou pravděpodobností. Předpokládejme, že pro sensmapserver bude vyhrazeno 30% procesorového času (ve smyslu simulačního modelu) a pro výpočetně nejnáročnější rtlsserver provádějící TDoA výpočty bude vyhrazeno 50% času. Cron softwarové démony můžeme zanedbat a budeme předpokládat, že tyto úlohy nebudou během aktivního běhu (sběru dat) systému spuštěny.

3.1 Zdůvodnění formy konceptuálního modelu

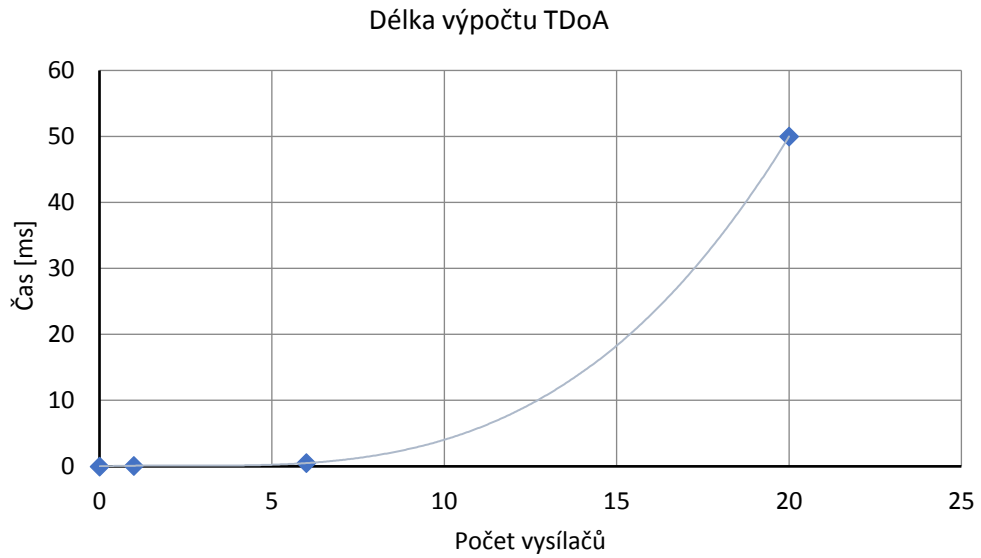
Jádrem systému je výpočet TDoA, jehož výpočetní délka je v simulačním modelu popsána pomocí Rovnice 1, kde proměnná x reprezentuje počet přijímačů, které jsou v daném systému nainstalovány. Sewio změřilo, že pokud je použito 6 snímačů, rychlost výpočtu se pohybuje v řádech stovek us. Pro 10-12 snímačů se doba výpočtu pohybuje v rozmezí 4-8ms. Pokud je však snímačů 20, náročnost výpočtu narůstá téměř exponenciálně a pohybuje se přibližně okolo 50ms, viz Graf 1. Jednotlivé komponenty systému jsou poté popsány pomocí Petriho sítí (1 str. 123).

3.2 Formy konceptuálního modelu

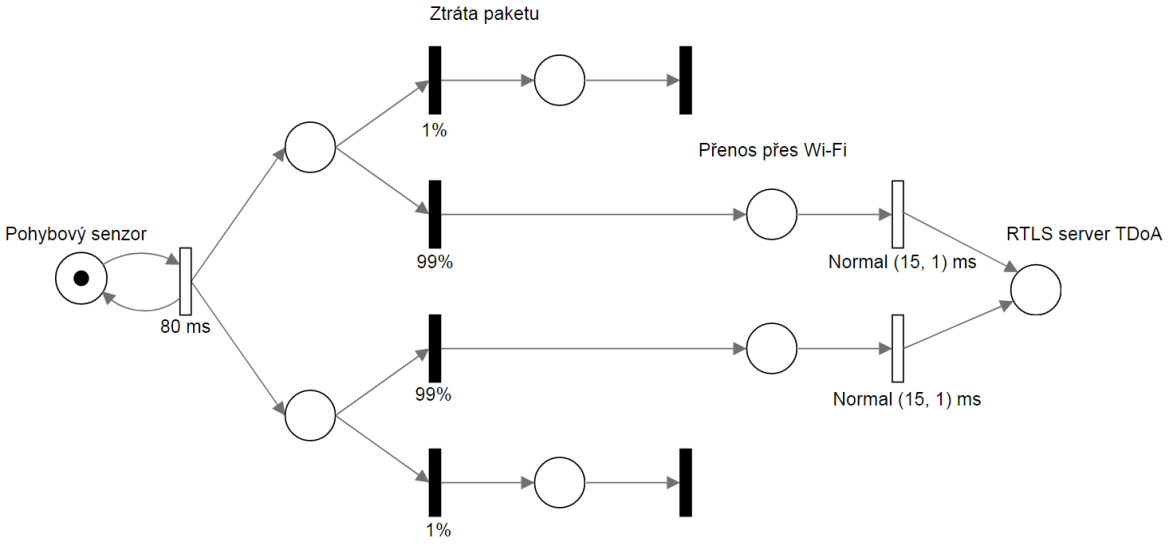
Konceptuální model je popsán pomocí Petriho sítí, jehož jednotlivé části jsou pro přehlednost simulačního modelu oddělené a mají svou vlastní Petriho síť – viz Obrázek 1, Obrázek 2 a Obrázek 3. Pro aproximaci délky časového výpočetu TDoA byl použit polynom třetího řádu.

$$f(x) = 0.0093x^3 - 0.682x^2 + 0.01589x + 0.0000000002\ [ms]$$

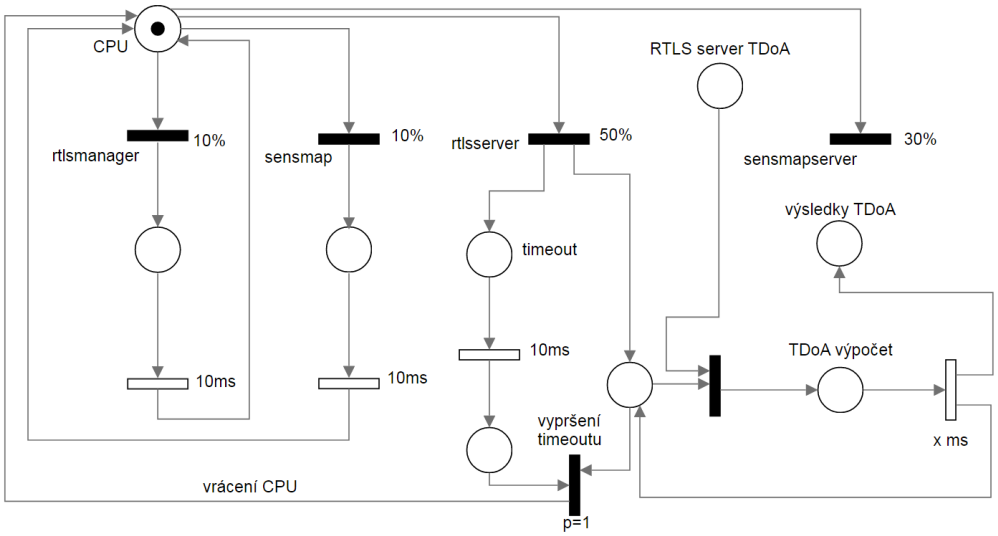
Rovnice 1 – Výpočet TDoA



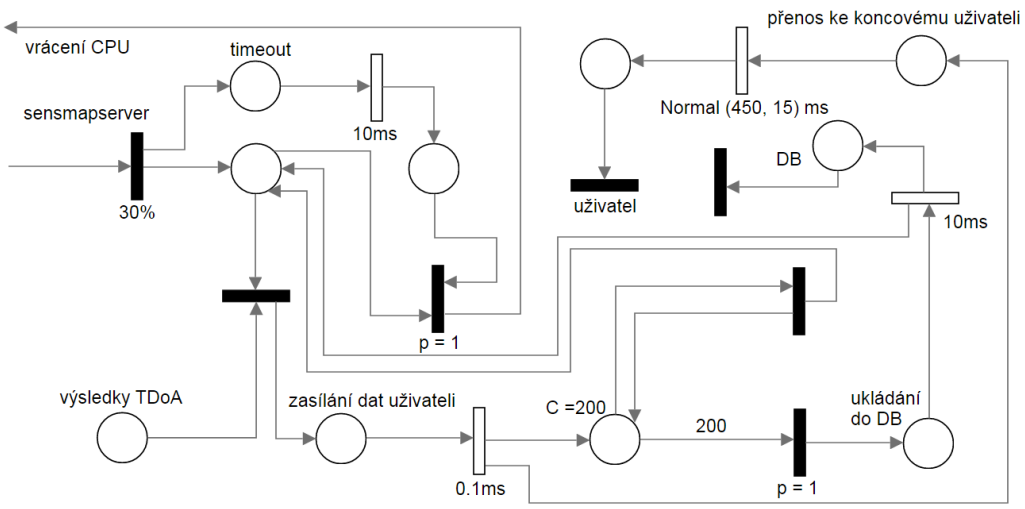
Graf 1 – Nárust délky výpočtu v závisloti na počtu vysílačů



Obrázek 1 – Petriho síť modelující cestu signálu od pohybového senzoru do RTLS serveru



Obrázek 2 – Petriho síť modelující část procesoru, rtlsmanager, sensmap a rtlsserver



Obrázek 3 – Petriho síť modelující sensmapserver a cestu dat k uživateli

4. Architektura simulačního modelu/simulátoru

Simulační model obsahuje několik základních tříd, ze kterých poté dědí třídy další. Třída `Facility` je použita pro reprezentaci procesoru. Třída `Queue` má právě tři instance, které shromažďují data čekající na zpracování. Model dále celkem obsahuje 5 tříd, přičemž 3 z nich dědí z třídy `Process` a další 2 z třídy `Event`. Simulační model navíc obsahuje třídu `Histogram`, která shromažďuje statistiky, a které jsou po ukončení běhu programu uloženy do výstupního souboru.

4.1 Mapování konceptuálního modelu na model simulační

Simulační model inicializuje systém pomocí události `SystemInitiator`, která aktivuje všechny třídy `Tag` a `Server`. Instance obou tříd jsou aktivní až do konce běhu programu. `Tag` v pravidelných intervalech pomocí metody `Activate` generuje nové pakety třídy `Packet`, které pohybové senzory vysílají. `Paket` slouží k simulaci toku dat přes Ethernet, či Wi-Fi až do serveru. Příchozí pakety se řadí do fronty `qTDOA` třídy `Queue`, kde čekají na výpočet. `Server` popisuje chování procesoru - aplikacím je v pravidelných intervalech přidělován procesorový čas. Data přeposílaná uživateli obsahuje třída

`Record`, jejíž instance jsou vytvářeny aplikací sensmapserver. Kromě tohoto sensmapserver také ukládá záznamy do databáze. Jednotlivé datové pakety jsou vloženy do fronty `dq`, jsou uspány pomocí metody `Passivate` a až při operaci hromadného nahrání do databáze jsou pakety znovu aktivovavány. Vytížení procesoru, kdy procesor aktivně provádí výpočty nebo provádí jiné operace je řešeno pomocí obsazení `Facility CPU`. Prováděná akce `CPU` obsadí, čeká určitou dobu `Wait` a poté `CPU` uvolní.

4.2 Spuštění programu

Program je možno buďto nejprve přeložit příkazem `make`, po jehož vykonání vzniká spustitelný binární soubor `tdoasystem`. Pokud se má program po přeložení ihned spustit, použije se příkaz `make run`.

Vstupy programu je nadále možno měnit pomocí těchto přepínačů:

```
$./tdoasystem [-h][-t NUM_OF_TAGS][-a NUM_OF_ANCHORS]
[-l SIM_LENGTH][-o OUTPUT][-r TAG_REF_RATE][-b BATCH_TO_DB]
[-s SEND_OVER_WIFI][-w WIFI_PING][-m METHOD_TRANSFER]
[-c CPU_SEIZE_TIME]
```

-h	Výpis nápovědy.
-t NUM_OF_TAGS	Počet pohybových senzorů.
-a NUM_OF_ANCHORS	Počet vysílačů.
-l SIM_LENGTH	Délka simulace v minutách.
-o OUTPUT	Výstupní soubor obsahující statistiky o modelu.
-r TAG_REF_RATE	Obnovovací frekvence pohybového senzoru v milisekundách.
-b BATCH_TO_DB	Počet údajů hromadně uložených do databáze.
-s SEND_OVER_WIFI	Využití Wi-Fi pro přenos dat (1 ano, 0 ne).
-w WIFI_PING	Wi-Fi latence v milisekundách.
-m METHOD_TRANSFER	Způsob přenosu dat k uživateli (1 REST, 2 Websockets, 3 UDP).
-c CPU_SIZE_TIME	Doba vyhrazená procesoru pro jeden proces v milisekundách (simulace multitaskingu).

5. Podstata simulačních experimentů a jejich průběh

V dalším textu bude popsáno několik simulačních experimentů – nejprve si definujeme postup práce, poté přejdeme do fáze experimentování a nakonec z výsledků vyvodíme závěr shrnující nově nabyté poznatky o systému.

5.1 Postup experimentování

První 2 experimenty budou zaměřeny na ověření správné implementace využití procesorového času během různých zatížení systému. Pokud budou jejich výsledky odpovídat očekávaným výsledkům, tak poté v následujícím experimentu změříme výstupy z reálného případu užití systému, kterým je akvizice informací z basketbalového utkání. Za předpokladu, že i tento experiment proběhně úspěšně, budeme následně model zkoumat podrobněji a experimentovat s jednotlivými prvky systému za účelem zjištění optimálního nastavení tak, aby byla průměrná rychlost přenosu dat ke koncovému uživateli co nejrychlejší a nejspolehlivější. Pokud budou výsledky z prvotního experimentu nesmyslné nebo velmi neuspokojivé, budeme muset model dále prověřit. Po provedení těchto experimentů dále zjistíme, kolik takových basketbalových utkání je možno do databáze uložit, než bychom museli diskový prostor

vyhrazený pro databázi navýšit. Dále zjistíme, jaký rozdíl v délce přenosu bude mezi REST API a UDP. Posledním experimentem vyzkoušíme, jak dobře si simulační model poradí s velmi vysokým tokem dat.

5.2 Experimenty

Experiment 1

Prvním a druhým experimentem ověříme, že využití procesorového času stoupá s počtem dat, která je nutno na serveru zpracovat. V první experimentu budeme simulovat pouze 2 pohybové senzory s obnovovací frekvencí 1000ms. Počet přijímačů je defaultně nastaven na 6. Pakety jsou do systému přenášeny přes Ethernet. Délka simulace je nastavena na 10 min.

Spuštění programu s parametry: `$./tdoasystem -t 2 -a 6 -s 0 -r 1000 -l 10`

FACILITY Procesor	
Status	not BUSY
Time interval	0 - 600000
Number of requests	14110
Average utilization	0.202263

Tabulka 1 – Vytížení procesoru Experimetu 1

Výsledné využití procesorového času (viz Tabulka 1) dopadlo dle očekávání - procesorový čas přiřazen aplikacím nebyl skoro téměř využit. Časová vytíženíženost procesoru 20% je způsobena jeho defaultním zabráním dvěmi aplikacemi. Přírustek v tomto experimentu činí pouhých 0.02263% procesorového času.

Experiment 2

Druhým experimentem již modelujeme reálnější situaci, kdy simulujeme systém využitý během části basketbalového utkání. Počet pohybových senzorů navýšíme na 24 (12 pro hráče každého týmu, senzory jsou připevněné pod dresem hráče a sledují jeho pohyb + další metriky jako je např. tepová frekvence apod.). Obnovovací frekvenci senzorů nastavíme na 50ms. Ostatní parametry ponecháme stejné jako v Experimentu 1.

Spuštění programu s parametry: `$./tdoasystem -t 24 -a 6 -s 0 -r 50 -l 10`

FACILITY Procesor	
Status	not BUSY
Time interval	0 - 600000
Number of requests	501744
Average utilization	0.464349

Tabulka 2 – Vytížení procesoru Experimentu 2

Využití procesorového času se v tomto experimentu navýšilo o více než 26% (viz Tabulka 2). Z experimentů 1 a 2 je možno konstatovat, že se simulační model chová dle očekávání.

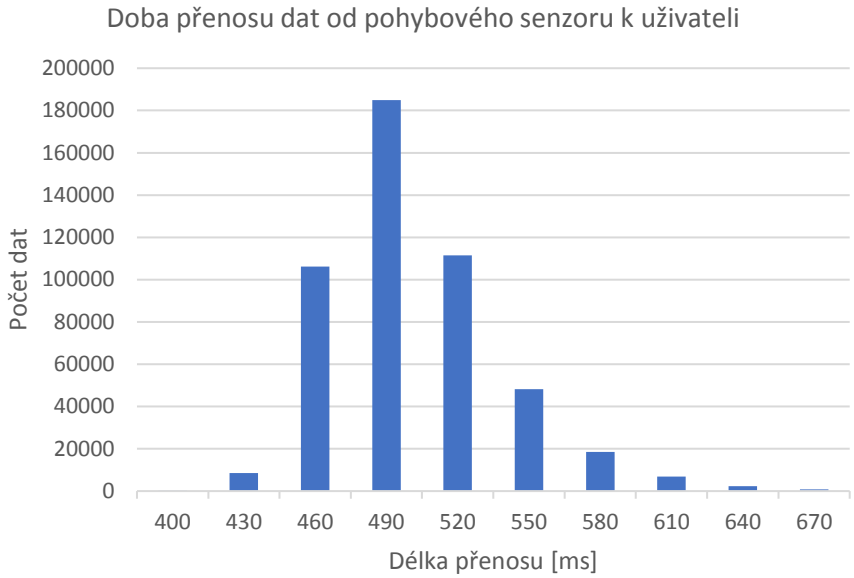
Experiment 3

V tomto experimentu budeme již modelovat reálnou situaci, kterou je akvizice dat během jednoho basketbalového utkání. Hodnoty parametrů zůstávají stejné jako v předchozím experimentu, změnou bude pouze délka simulace na 20 min, což je hodnota odpovídající přibližně délkce jedné čtvrtiny utkání (10 min + přerušení ve hře[17]). Celý zápas v tomto experimentu nemodelujeme z důvodu přestávek ve hře, během nichž nejsou pohybové senzory aktivní – pokud v systému dochází ke vzniku front, stihnout

[17] Ukázka délky utkání. https://www.youtube.com/watch?v=wwHAipKiwWo.

se během přestávek vyprázdnit, což by simulace zápasu jako celku neumožnila. Získaná data budou přenášena až k uživateli, který je bude sledovat. Přenos bude uskutečněn použitím REST API.

Spuštění programu s parametry: `$./tdoasystem -t 24 -a 6 -s 1 -r 50 -l 20 -m 1`



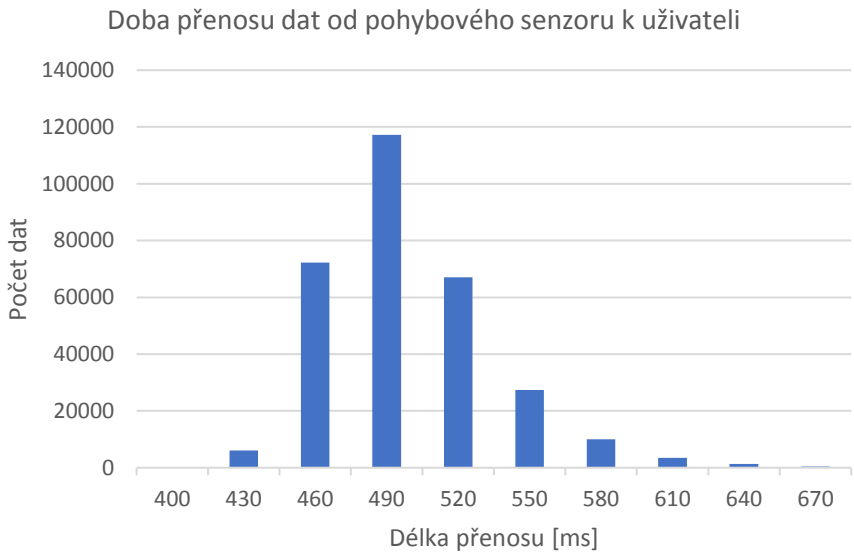
Graf 2 – Histogram popisující Experiment 3

Průměrná hodnota přenosu dat činí 516.547ms. Z experimentu můžeme vyvodit, že přesnost dat během utkání pomocí REST API je uskutečnitelný. Navíc se jedná o spolehlivý přenos, během kterého se posílají i podrobnější data. Více jak 94% dat je přeneseno v časovém úseku 460-580ms (viz Graf 2).

Experiment 4

V tomto experimentu se budeme snažit dosáhnout ještě rychlejšího přenosu dat. Obnovovací frekvenci snížíme na 80ms, která stále postačuje pro kvalitní real-time přenos dat. Změníme také počet záznamů, které se hromadně přesouvají do databáze na 500. Očekávaným výsledkem je nižší průměrná hodnota přenosu dat (počet přenesených dat bude samozřejmě nižší vhledem k nižší frekvenci).

Spuštění programu s parametry: `$./tdoasystem -t 24 -a 6 -s 1 -r 80 -l 20 -m 1 -b 500`



Graf 3 – Histogram popisující Experiment 4

Experiment ukázal, že průměrná rychlost přenosu se snížila na 514.054ms. Oproti předchozímu experimentu se nejedná o velké zlepšení, avšak větší rozdíl lze pozorovat v maximální délce přenosu - Experiment 3 naměřil maximální délku 789.408ms, zatímco v tomto experimentu bylo naměřena hodnota 772.877ms (tyto hodnoty na histogramech Graf 2 a Graf 3 zobrazeny nejsou, poněvadž se jedná se o ojedinělé hodnoty, které v porovnání s ostatními nejsou v daném měřítku a rozlišení zobrazitelné). U snížení také došlo u směrodatné odchylky z 36.4771 na 35.3574, která značí vyšší konzistenci přenosu dat, která bude znamenat menší výkyvy na straně uživatele. Ten tak data bude moci jednodušeji dopočítávat či vykreslovat.

Experiment 5

Pátý experiment bude vycházet z nastavení předchozího experimentu a bude zaměřen na velikost uložených dat v databázi. Velikost databáze je 913GB, viz 2. Na základě výsledku simulace jednoduchou aritmetikou zjistíme, kolik basketbalových utkání je možné do databáze uložit. Důvodem nesimulování všech zápasů je příliš vysoká délka simulace, která však výsledky s použitím tohoto simulačního modelu nijak neovlivní.

Spuštění programu s parametry: `$./tdoasystem -t 24 -a 6 -s 1 -r 80 -l 20 -m 1 -b 500`

Celkovy pocet dat ulozenych v databazi: 77.647499MB.

Výpočet počtu utkání uložitelných do databáze:

$$\text{Počet utkání} = \frac{\text{Velikost databáze [GB]}}{4 \text{ čtvrtiny utkání} \times \text{počet uložených dat během 1 čtvrtiny [GB]}} \text{ [utkání]}$$

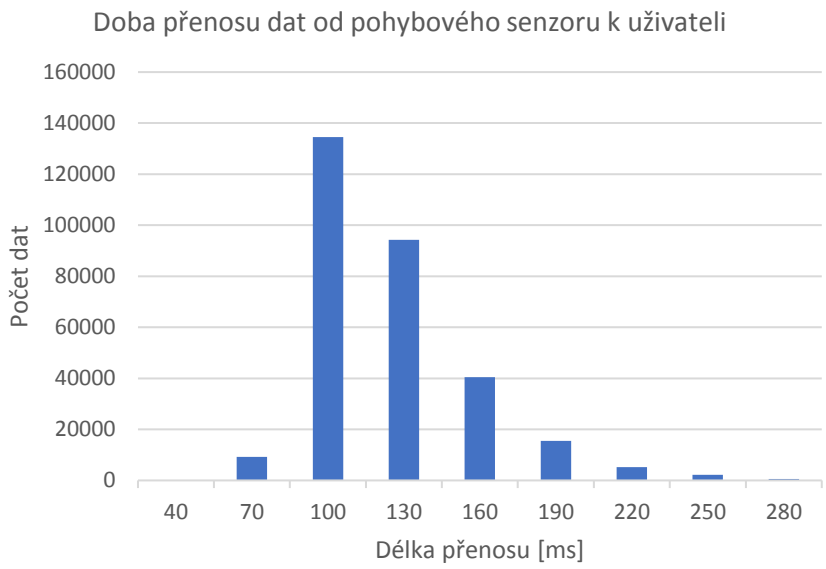
$$\text{Počet utkání} = \frac{913}{4 \times 0.077647499} \cong 2939 \text{ [utkání]}$$

Rovnice 2 – Výpočet popisující Experiment 5

Z výsledku je zřejmé (viz Rovnice 2), že velikost databáze pro ukládání historie sportovních utkání je více než postačující.

Experiment 6
Tímto experimentem zjistíme, jaká bude průměrná rychlost přenosu dat, pokud budeme data posílat přes UDP. Předpokládáme dosažení výrazně nižších hodnot, což je však vykoupeno omezeným počtem informací, které je možno přenášet a také nespolehlivým přenosem, který bude muset uživatel kompenzovat.

Spuštění programu s parametry: `$./tdoasystem -t 24 -a 6 -s 1 -r 80 -l 20 -m 3 -b 500`



Graf 4 – Histogram popisující Experiment 6

Průměrná hodnota 139.558ms dokazuje výrazné snížení doby přenosu. Hodnota je dostatečně nízká na to, aby se reálné pozice hráčů spolu s výstupními daty takřka překrývaly a přenos se tak jevil opravdu real-time (viz Graf 4).

Experiment 7
Poslední experiment bude zaměřen na zátěž systému. Předpokládejme situaci, kdy budeme mít 20 přijímačů, 20 pohybových senzorů s obnovovací frekvencí 500ms, přenos po Wi-Fi a délce simulace 15 min. Tento systém může modelovat supermarket, kdy jsou pohybové senzory připevněny na nákupních vozících a dávají nám tak povědomí o tom, kde se zákazníci nacházejí.

Spuštění programu s parametry: `$./tdoasystem -t 20 -a 20 -s 0 -r 500 -l 15 -m 1 -b 200`

HISTOGRAM Prenos dat od pohyboveho senzoru k uzivateli	
STATISTIC	
Min = 627.416	Max = 504336
Number of records	14936
Average value	253549
Standard deviation	145389

Tabulka 3 – Souhrnné statistiky přenosu dat popisující Experiment 7

FACILITY Procesor	
Status	BUSY
Time interval	0 - 900000
Number of requests	35921
Average utilization	0.903917

Tabulka 4 – Vytížení procesoru Experimentu 7

QUEUE Fronta pozadavku cekajicich na zpracovani RTLS serverem (TDOA)	
Time interval	0 – 900000
Incoming	712666
Outcoming	712666
Current length	0
Maximal length	398989
Average length	199987
Minimal time	39.1669
Maximal time	503999
Average time	252556
Standard deviation	145437

Tabulka 5 – Fronta požadavků čekající na zpracování TDoA u Experimentu 7

Z výsledů je zřejmé, že systém výpočet absolutně nezvládnul. Průměrná délka času přenosu paketu je 253s, která je tedy pro real-time přenos prakticky nevyužitelná. Pokud se podíváme na model systému podrobněji, tak z Tabulka 4 zjistíme, že procesorový čas je již z více než 90% zužitkován. Tabulka 5 navíc ukazuje, že fronta požadavků čekajících na výpočet TDoA je přeplněná a procesor požadavky nestíhá zpracovávat. Z těchto 2 faktů můžeme konstatovat, že systém dosáhl svého výpočetního limitu a pro velmi náročné výpočty není vhodný. Za zmínku však stojí řádek Current length v Tabulka 5 s hodnotou 0. Znamenalo by to, že se fronta stihla vyprázdnit, což při bližším zkoumání výsledků nedává smysl. Příčinou zde může být implementace programu, která čekající procesy ve frontě stihla dealokovat dříve, než byl vypsán histogram.

5.3 Závěry experimentů

Během experimentování se simulačním modelem bylo provedeno 7 experimentů, z čehož byly první 2 zaměřeny na validitu modelu a ověření základní funkčnosti modelovaného procesoru. Časové vytížení procesoru u Experimentu 2 dosáhlo přibližně 46%, přičemž systém stále stíhal požadavky zpracovávat. U posledního experimentu tomu už tak ale nebylo a přestože byl procesor využit z 90% času, došlo k zahlcení systému – tohle by se však u reálného systému nestalo a procesor by si s velkým počtem přijímačů díky multithreadingu a dvoujádrovému procesoru poradil lépe.

Dalšími experimenty by se dalo zjistit, jak moc systém ovlivňuje například přidělený procesorový čas, či změna modelu na model podporující preemptivní multitasking. U ještě pokročilejšího modelu bychom mohli navíc modelovat podporu více vláken.

6. Shrnutí simulačních experimentů a závěr

V rámci projektu vznikl nástroj pro simulaci toků dat v systému pro zpracování rádiových časoměrných signálů. Tento program umožnuje se simulačním modelem pomocí změny několika parametrů experimentovat a simulovat tak chování systému v mnoha situacích, které by bez tohoto nástroje nebylo možné jednoduše zrealizovat.

U experimentů simulujících akvizici informací z basketbalového utkání jsme zjistili, že je přenos dat v reálném čase s daným systémem uskutečnitelný. Pro REST API systém zaručuje jak spolehlivost, tak i rychlost přenosu dat. Pro UDP je rychlost ještě vyšší, přicházíme však o kvalitu přenosu dat a také o rozsah informací, které jsme schopni použitím tohoto protokolu přenést. Experimentálně bylo zjištěno, že nižší frekvence vysílání signálů z pohybových senzorů společně v kombinaci s méně častým hromadným ukládáním dat do databáze zvyšuje rychlost přenosu dat ke koncovému uživateli. Při využití REST API z Graf 2 a Graf 3 vyplývá, že délka přenosu dat se řídí normálním rozložením pravděpodobnosti. Interpretací výsledků je možno také zjistit, že mezi hlavní prediktory pro rychlost systému je výkonný procesor, jehož případné nedostatky se obzvláště projevují v architekturách systému obsahující více přijímačů.

Z počtu přenesených dat u experimentů můžeme konstatovat, že experimenty budou při opakovaném spuštění vykazovat velice podobné chování. Model můžeme považovat za validní.

Reference

1. **Peringer, Petr a Hrubý, Martin.** Vysoké Učení Technické Brno. [Online] 17. Říjen 2017. [Citace: 3. Prosinec 2017.] http://www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf.
2. **Peringer, Petr.** *Simlib.* [Online] 4. Říjen 2017. [Citace: 3. Prosinec 2017.] http://www.fit.vutbr.cz/~peringer/SIMLIB/.