Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Розподілена комп'ютерна система спеціалізованого призначення згідно топології "тор"
Назва теми

КвРКІ.180103.18.01.03 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія»*
Назва

Виконав: студентка IV курсу, група КІ–18–1 _____ А.Я. Гнатчук
Підпис Ініціали, прізвище

Керівник _____ О.М. Березький
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ С.М. Лисенко
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної
інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«8» червня 2022 р.

Хмельницький 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 11 " 01 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Гнатчук Аліні Ярославівні

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Розподілена комп'ютерна система згідно топології "тор"

Керівник проекту (роботи) Березький О.М., д.т.н., проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 06.01.2022 р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 07.06.2022 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Топології у розподілених комп'ютерних системах

Архітектура розподіленої комп'ютерної системи спеціалізованого призначення

Дослідження ефективності

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Схематичні зображення топології "тор"

Схеми будови комп'ютерних архітектур

Приклади побудови моделі системи

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийн
Нормоконтроль	Лисенко С.М., професор кафедри КІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІС		

7. Дата видачі завдання «__06__»_____09_____2021 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примі
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	11.01.2022	виконан
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2022	виконан
3	Робота над розділом 1 – топології у розподілених комп'ютерних системах	01.03.2022	виконан
4	Робота над розділом 2 – архітектура розподіленої комп'ютерної системи спеціалізованого призначення	01.04.2022	виконан
5	Робота над розділом 3 – дослідження ефективності	30.04.2022	виконан
6	Оформлення пояснювальної записки згідно вимог	31.05.2022	виконан
7	Попередній захист ВКР	02.06.2022	виконан
8	Захист ВКР на засіданні ЕК	Червень 2022 року	

Студент _____ А. Я. Гнатчук
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ О. М. Березький
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 180101.18.01.03 ПЗ	Пояснювальна записка	58		
			Графічні матеріали			
2		КвРКІ 180101.18.01.03 Е8	Схематичні зображення	1		
			топології “тор”			
3		КвРКІ 180101.18.01.03 Е8	Схеми будови	1		
			комп’ютерних архітектур			
4		КвРКІ 180101.18.01.03 Е8	Приклади побудови	1		
			моделі системи			

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 180101.18.01.03 ВП		
Розробив		Гнатчук			Відомість проекту	Літера	Аркуш 1 / Аркушів 1
Перевір.		Березький				У	
Н. контр.		Лисенко				ХНУ, КІ-18-1	
Затв.		Говорущенко					

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Розподілена комп'ютерна система пеціалізованого призначення згідно топології "тор"».

Автор роботи: Гнатчук Аліна Ярославівна.

Керівник роботи: Березький Олег Миколайович.

Пояснювальна записка: 58 с., 26 рис., 3 табл., 3 дод., 50 джерел.

Графічна частина: 8 презентаційних слайдів.

РОЗПОДІЛЕНА КОМП'ЮТЕРНА СИСТЕМА СПЕЦІАЛІЗОВАНОГО ПРИЗНАЧЕННЯ ЗГІДНО ТОПОЛОГІЇ "ТОР"

Метою роботи є розробка розподіленої комп'ютерної системи спеціалізованого призначенні згідно топології "тор".

У цій роботі було побудовано модель системи SmarTor, що містить релейні вузли, приховані в Tor веб-сервіси і клієнти Tor, також було враховано можливість атаки на дану систему, а щоб запобігти цьому, було розглянуто можливі шляхи нанесення атак, при яких система може постраждати. Також було оглянуто систему високого рівня, щоб продемонструвати взаємодію між залученими сутностями. Використовуючи функцію виключення, було продемонстровано ймовірність виключення вимірювача пропускної здатності після кількох раундів вимірювання.

Підпис студента  Дата 08.06.2022

ЗМІСТ

ВСТУП

Під час всього історичного розвитку обчислювальних машин спроби введення певної класифікації, що визначала би комп'ютерні архітектури, були не зовсім вдалими. Однак наразі, вони являються основою ряду часто використовуваних термінів.

Кожна обчислювальна машина може досягти найкращої продуктивності завдяки можливості реалізовувати багато паралельних операцій. Тому, перспектива застосування виконання завдань паралельно є найважливішою.

Одним з основних напрямків розвитку обчислювальної техніки являється постійне запровадження розподілених комп'ютерних систем та побудова таких обчислювальних сукупностей.

Розподілена комп'ютерна система являє собою системо-утворюючу складову інформаційних структур. Вона представляє собою набір декількох незалежних комп'ютерів, які сприймаються користувачем як об'єднана система.

Програмне забезпечення, розроблене для розподіленої комп'ютерної системи, конкретно відрізняється від програмування для інших централізованих систем. Для того, щоб розробити мережеві застосування, потрібно організувати спільну роботу частин мережевих застосувань, які виконуються на різних машинах.

Залежність продуктивності розподіленої комп'ютерної системи полягає в якості та швидкості роботи розподіленої мережі. У свою чергу, проблемою являється намагання забезпечити безпеку певних складових розподіленої комп'ютерної системи, а також захистити інформацію, яка надходить різноманітними каналами зв'язку, або зберігається у різних розподілених ресурсах.

На сьогодні актуальність цієї роботи являється важливою тому, що нинішній рівень науки та технологій потребує вирішення не простих завдань. Ось чому сфера застосування розподілених комп'ютерних систем все більше впливає на різноманітні сфери життя.

1 ТОПОЛОГІЇ У РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

1.1 Означення топології "тор"

Поняття топології багатопроцесорної системи означає собою певний спосіб з'єднання між собою процесорних вузлів каналами передачі даних. Найкраще топології подаються у вигляді графів, у яких вершини являються процесорними вузлами, а ребра — каналами зв'язку [1]. В свою чергу, топології умовно поділяються на фіксовані, реконфігуровані, регулярні та нерегулярні. До прикладу, топологія "тор 3x3" належить до регулярних (рис. 1.1).

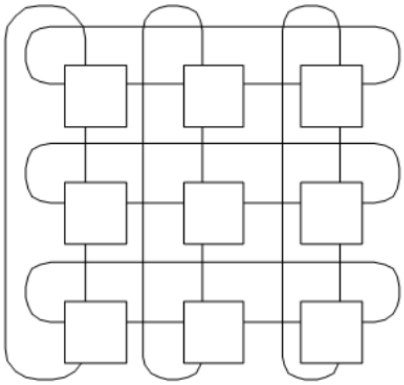


Рисунок 1.1 — Топологія "тор 3x3" [2]

Властивості, які притаманні певній топології, мають здатність визначати ефективність виконання паралельної програми і можливість масштабування обчислювальної системи [3]. Для побудови топології типу "тор" необхідно n1·n2 процесорів. Це означає, що при збільшенні кількості процесорів, потрібні лише кванти розміру n1·n2.

Для підвищення ефективності виконання програми на обчислювальних системах, потрібно здійснювати узгодження топології задачі та фізичної топології системи. Перші багатопроцесорні системи були обмежені з боку реконфігурування. Системи з топологією "тор", які були побудовані на основі трансп'ютерів, називали лишу умовно [4]. Це відбувалося тому, що для того, щоб реалізувати саме цю топологію, необхідно мати саме чотири link, де для приєднання системи до обчислювальної машини не залишається ні одного link (рис. 1.2).

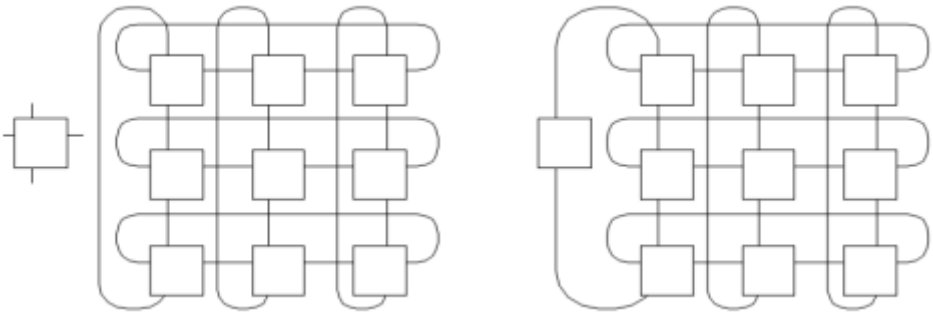


Рисунок 1.2 — Під’єднання процесорів до керувальної машини за допомогою топології “тор” [2]

В загальному можна сказати, що торовим з’єднанням являється топологія мережі без комутатора; вона створена для з’єднання вузлів обробки в паралельній комп’ютерній системі (рис. 1.3). Такий взаємозв’язок не обмежується 8-ма вузлами, однак, він може складатися з будь-якої кількості вузлів у подібному прямолінійному масиві [5].

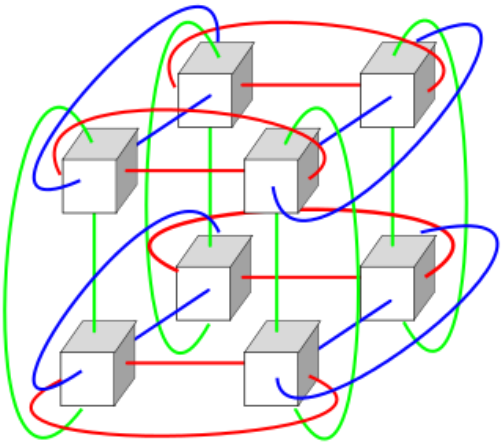


Рисунок 1.3 — Схема 3-вимірного торичного взаємозв'язку [6]

У геометрії тор створюється шляхом обертання кола навколо осі, компланарної колу [7]. Це поняття являється доволі загальним в геометрії, топологічні властивості цього типу фігури описують топологію мережі в її суті (рис. 1.4).

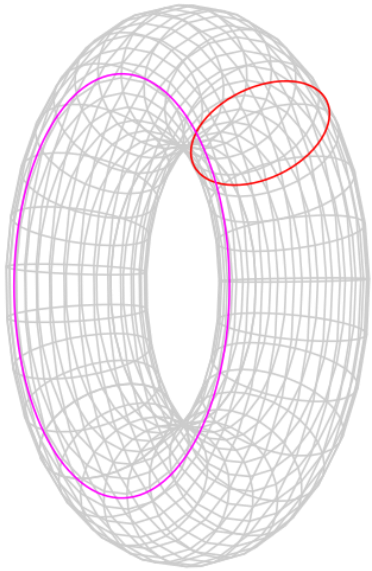


Рисунок 1.4 — Тор із співвідношенням сторін як добуток меншого (червоного) і більшого (рожевого) кола [8]

Загалом, у геометрії тор може бути визначений параметрично (формула 1.1):

$$x(\theta, \varphi) = (R + r\cos\theta)\cos\varphi$$
$$y(\theta, \varphi) = (R + r\cos\theta)\sin\varphi \qquad (1.1)$$
$$z(\theta, \varphi) = r\sin\theta$$
$$\theta, \varphi \in [0, 2\pi)$$

де θ, φ - кути, які утворюють повне коло, тому їх значення починаються і закінчуються в одній точці,

R - відстань від центра трубки до центру тора,

r - радіус труби.

1D-тор являє собою просте коло, а 2D-тор представлений у формі пончика. Поняття тора використовується для опису початку і кінця послідовності вузлів, які пов’язані, як пончик [9]. Для кращого розуміння означення топології в мережевому взаємоз’єднанні, наведемо 3 приклади паралельних взаємопов’язаних вузлів з використанням топології тора. В одному вимірі топологія тора еквівалентна кільцевій мережі, що має форму кола. У 2D топологія тора є еквівалентною 2D сітці, проте з додатковим з’єднанням на крайових вузлах (рис. 1.5-1.6).

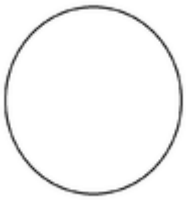


Рисунок 1.5 — Приклад 1D тора, коло [6]

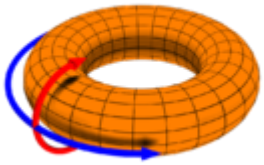


Рисунок 1.6 — Приклад 2D тора, пончик [6]

З наведених вище малюнків можна узагальнити певне правило. Торове з'єднання — це топологія без комутатора, яку можна розглядати як сітку з вузлами, розташованими в прямолінійному масиві N = 2, 3 або більше, з процесорами, підключеними до своїх найближчих сусідів, і відповідними процесорами на протилежних краях масив підключений. У цій решітці кожен вузол має 2N з'єднань. Ця топологія отримала назву через те, що сформована таким чином решітка топологічно однорідна N-вимірному тору [10].

1D Тор являє собою лише один вимір, де n вузлів з'єднані в замкнутий цикл, причому кожен вузол, з'єднаний з 2-ма його найближчими сусідами, зв'язок може відбуватися в 2 напрямках, +x і -x. 1D тор є тим самим, що і кільцеве з'єднання.

2D-тор є двовимірним з ступенем 4, де вузли уявляються викладеними у двовимірну прямокутну решітку з n рядків і n стовпців, причому кожен вузол з’єднаний з 4-ма найближчими сусідами, а відповідні вузли з’єднані на протилежних ребрах. З’єднання протилежних ребер можна візуалізувати наступним чином: згорнути прямокутний масив у «трубку», для з’єднання двох протилежних ребер, а потім зігнути «трубку» в тор, щоб з’єднати два інших.

Комунікація може відбуватися в 4 напрямках, +x, −x, +y і −y. Загальна кількість вузлів 2D тора дорівнює n^2.

Уявити та описати (рис. 1.7-1.9) перші 3 топології мережі тора легко.

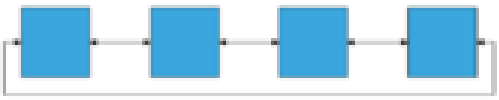


Рисунок 1.7 — Ілюстрація 1D тора [6]

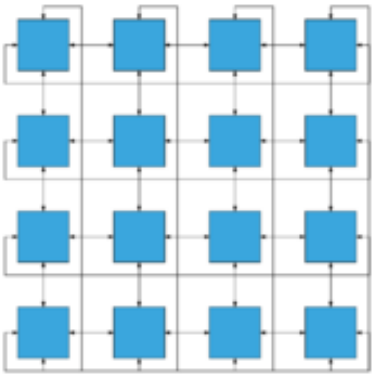


Рисунок 1.8 — Ілюстрація 2D тора [6]

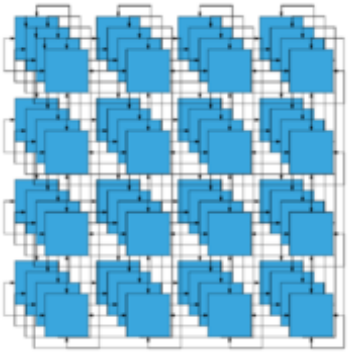


Рисунок 1.9 — Ілюстрація 3D тора [6]

3D-тор є тривимірним, де вузли уявляються у тривимірній решітці у формі прямокутної призми, причому кожен вузол з'єднаний зі своїми 6-ма сусідами, з відповідними вузлами на протилежних гранях масиву. Кожне ребро складається з n вузлів. Комунікація може відбуватися у 6 напрямках, +x, −x, +y, −y, +z, −z. Кожне

ребро 3D-тора складається з n вузлів. Загальна кількість вузлів 3D-тора дорівнює n^3.

Тор ND, у свою чергу може мати розмірність N, де кожен вузол тора розмірності N має 2N сусідів. Зв'язок може відбуватися в 2N напрямках. Кожне ребро складається з n вузлів. Загальна кількість вузлів цього тора дорівнює n^N. Основна мотивація наявності більшого розміру тора полягає в досягненні більшої пропускної здатності, меншої затримки та більшої масштабованості.

Масиви більшого розміру важко візуалізувати, однак, з вищезгаданого правила, що кожен вищий вимір додає ще одну пару найближчих сусідніх з'єднань до кожного вузла.

Деякі суперкомп'ютери зі списку TOP500 використовують тривимірні торові мережі, напр. IBM Blue Gene/L і Blue Gene/P, а також Cray XT3 [11].

IBM Blue Gene/Q використовує п'ятивимірну мережу тора. Комп'ютер Fujitsu K і PRIMEHPC FX10 використовують запатентовану тривимірну тривимірну сітку з тором під назвою Tofu [6].

Сандіп Палур і доктор Іоан Райку з Іллінойського технологічного інституту провели експерименти для моделювання роботи 3D-тора. Їхні експерименти проводилися на комп'ютері з 250 ГБ оперативної пам'яті, 48 ядрами та архітектурою x86_64. Симулятором, який вони використовували, був ROSS. Вони в основному зосередилися на трьох аспектах:

1. Різний розмір мережі.

2. Різна кількість серверів.

3. Різний розмір повідомлення.

Вони дійшли до висновку, що пропускна здатність зменшується зі збільшенням серверів і розміру мережі. В іншому випадку пропускна здатність збільшується зі збільшенням розміру повідомлення.

Компанія Fujitsu Limited розробила комп'ютерну модель 6D-тора під назвою «Tofu» [12]. У їхній моделі 6D-тор може досягати пропускної здатності поза чіпом 100 ГБ/с, у 12 разів більшої масштабованості, ніж 3D-тор, і високої відмовостійкості.

Далі розглянемо переваги та недоліки.

Переваги:

1. Вища швидкість, менша затримка. Через з’єднання протилежних країв, дані мають більше можливостей для переміщення від одного вузла до іншого, що значно збільшує швидкість.
2. Краща справедливість. У з’єднанні сітки 4×4 найдовша відстань між вузлами – від верхнього лівого кута до нижнього правого кута. Кожним даним необхідно 6 стрибків, щоб пройти найдовший шлях. Але в інтерконнекті тору 4×4 шлях від верхнього лівого кута до нижнього правого кута може займати лише 2 стрибки.
3. Менше споживання енергії. Оскільки дані, як правило, при переміщенні роблять менше стрибків, то споживання енергії, як правило, нижче.

Недоліком є складність електропроводки. Додаткові дроти можуть ускладнити процес маршрутизації на фазі фізичного проектування. Якщо необхідно розкласти більше проводів на мікросхемі, то буде потрібно збільшити кількість металевих шарів чи зменшити щільність на чіпі, що дорожче. В іншому випадку дроти, які з’єднують протилежні краї, можуть бути набагато довшими ніж зв’язки, що обертаються, можуть бути найпростішим способом візуалізації топології з’єднання, однак, на практиці обмеження на довжину кабелю часто роблять довгі зв’язки непрактичними. Натомість вузли, що безпосередньо пов’язані, включаючи вузли, які вищенаведена візуалізація розміщує на протилежних краях сітки, з’єднані довгим зв’язним зв’язком, — фізично розміщені майже поруч один з одним у складеній мережі тора. Кожна ланка в мережі зі складеним тором дуже коротка — майже така ж коротка, як і зв’язки найближчого сусіда в простому з’єднанні мережі — і, отже, має низьку затримку.

1.2 Концепції програмних рішень

Розподілена комп’ютерна система здійснює доволі ефективний обмін даних при взаємодії певних програмних компонентів, які розташовані на одному чи декількох комп’ютерах.

Ці системи працюють як менеджери ресурсів (resource managers) певного апаратного забезпечення.

Це надає допомогу користувачам та програмам спільно використовувати процесори, мережу, дані, пам’ять та інше. Крім того, розподілена система може надати віртуальну машину для того, щоб виконувати прикладні завдання.

При цьому вона здатна приховати свою складність та гетерогенну природу свого забезпечення.

Системи проміжного рівня та операційні системи є основними програмними компонентами (табл. 1.1).

Існує два види операційних систем для комп’ютерів з розподіленою системою — сильнозв’язні та слабкозв’язні.

Сильнозв’язні операційні системи також називаються розподіленими операційними системами (англ. Distributed Operating System, DOS).

Головною метою є приховання деталей керування апаратним забезпеченням, яке в свою чергу виконує декілька обчислювальних процесів.

Слабкозв’язні мережні операційні системи (англ. Network Operating Systems, NOS) створені для керування мульти-комп’ютерними системами. Ці мережі мають RAS, тобто сервіс віддаленого доступу.

Таблиця 1.1 — Опис розподілених, операційних та проміжного рівня систем

Система	Опис	Призначення
Розподілена система	Є сильнозв'язною системою для мультипроцесорів	Керування апаратним забезпеченням
Операційна система	Представляє собою слабозв'язну систему для локальних чи глобальних мереж	Забезпечення віддалених клієнтів локальними службами
Система проміжного рівня	Являється додатковим рівнем над операційною системою. Допомагає реалізувати служби загального призначення	Прозорість розподілу

Крім того, існують також два типи розподілених операційних систем — мультипроцесорна та мультикомп'ютерна операційні системи. Вони відповідають за керування ресурсами мультипроцесора та гомогенних мультикомп'ютерів відповідно [13].

Зазвичай операційні системи створювались, щоб керувати комп'ютером з одним процесором. Звідси походить і їхня назва — однопроцесорні. Ключовою метою цієї системи було організувати доступ користувачів та прикладних програм до поділювальних ними пристроїв. Для прикладної програми ресурси знаходяться у розпорядженні, в той же час, декілька прикладних програм можуть виконуватися в одній такій системі, при цьому кожна програма володіє своїм набором ресурсів. У такій ситуації реалізується віртуальна машина та надає засоби мультизадачності для прикладних програм.

До прикладу, програма А може змінювати дані програми В тому, що вона співпрацює з тією ж частиною загальної пам'яті, де і дані зберігаються. Операційна

система може надати первинні операції зв'язку тільки в тому випадку, якщо прикладні програми користуються засобами так, як необхідно.

Більшість процесорів здатні підтримувати 2 режими роботи — режим ядра та режим користувача. Всі інструкції можуть виконуватися у режимі ядра, якщо доступна наявна пам'ять та регістри. Доступ до пам'яті та регістрів у режимі користувача є обмеженим. Іншими словами, робота з пам'яттю для прикладних програм чи звернення до регістрів є неможливими. Процесор переходить у режим ядра при виконанні коду. Для того, щоб переключитись з режиму користувача в режим ядра, потрібно зробити системний виклик. Так як вони можуть здійснюватись тільки базовими службами, то операційна система має повний контроль доступу до пам'яті та регістрів.

Завдяки тому, що є 2 режими роботи, весь код виконується в режимі ядра. Звідси створюються величезні монолітні програми, які можуть працювати у єдиному адресному просторі. Однак, даний підхід можу значно ускладнити процес переналаштування системи тому, що доволі проблематично замінити чи адаптувати певні компоненти цієї операційної системи без перезавантаження. Так як основними вимогами до проектування програм є відкритість, легкість обслуговування та надійність, то монолітні операційні системи є неефективними.

Операційна система, яка має вигляд двох частин, являється зручнішою. В одній частині містяться набір модулів, які призначені для керування апаратним забезпеченням, що без проблем виконується при режимі користувача. До прикладу найголовнішим у керуванні пам'яттю є відстеження конкретних блоків цієї ж пам'яті та визначення вільні вони чи ні. Також, для встановлення регістрів блока керування пам'яттю потрібно працювати у режимі ядра.

У другій частині операційної системи міститься невелике мікроядро. У нього є лише код, який виконується лише в режимі ядра [14]. Загалом, у мікроядрі повинні знаходитись коди для установлення регістрів, для перехоплення апаратних переривань, перемикання процесора та код роботи з блоком керування пам'яттю. Також, у певному мікроядрі є такий код, який здатний здійснити перетворення викликів відповідного модуля в системні виклики (рис. 1.10).

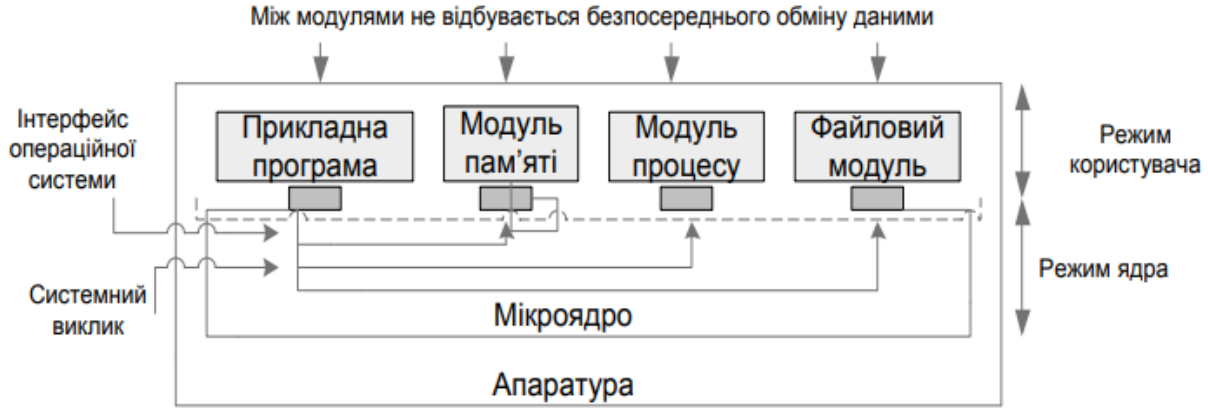


Рисунок 1.10 — Організація операційної системи з мікроядром [2]

Основною перевагою мікроядра є гнучкість системи, бо більшість операційної системи виконується у користувацькому режимі. Це означає, що здійснити заміну одного з модулів без поторної компіляції стає набагато простіше. Також, перевагою є те, що модулі користувацького рівня можуть бути розміщеними на різних машинах. Даний підхід є дуже зручним для того, щоб працювати з однопроцесорними операційними системами на розподілених комп'ютерах.

Однак, у використанні мікроядер також є недоліки — вони працюють іншим чином, ніж операційні системи, а також додатковий обмін даних, що значно понижує продуктивність.

Подальшим розвитком для однопроцесорних операційних систем є підтримка декількох процесорів, що мають власний вільний доступ до спільної пам'яті. Структури даних, які важливі для підтримки апаратури, розміщують в пам'яті, яка також доступна для декількох процесорів. Тому вони повинні бути захищеними від паралельного доступу для того, щоб було можливим забезпечити їх цілісність.

Ці багатопроцесорні операційні системи важливі для того, щоб підтримувати високу продуктивність різних конфігурацій з декількома процесорами. Основною метою є забезпечення прозорості кількості процесорів для прикладних програм.

Системами з розподіленою поділюваною пам'яттю називають такі системи, які можуть застосовувати віртуальну пам'ять окремих вузлів, щоб підтримувати загальний адресний простір. Це сприяє використанню розподіленої поділювальної

пам’яті (DSM). Робота даної пам’яті полягає у тому, що адресний простір розділено на сторінки, які зазвичай розподіляються по процесорах системи. Внутрішнє переривання можу відбуватися, коли процесор хоче адресуватися до ще не локальної пам’яті. Операційна система може перезапустити виконання інструкції, через яку відбулося переривання, після цього дана інструкція може успішно виконатися. Для тимчасового сховища інформації використовують віддалену оперативну пам’ять.

Різницею між мережними та розподіленими операційними системами є відсутність потреби у гомогенному апаратному забезпеченні. Набір однопроцесорних систем являються основою побудови мережних систем. Не зважаючи на те, що машини можуть відрізнятися від власних операційних систем, вони у будь-якому випадку будуть з’єднанні в одну мережу. Користувачам дозволено використовувати певні служби, які містяться в конкретній машині, а саме: віддалені копіювання файлів та з’єднання з іншою машиною.

Крім того, переваги також і властиві мережним операційним системам. До прикладу, використати можливість додавання чи видалення машини дуже легко. Це може відбутися завдяки тому, що вузли мережних операційних систем не принципово залежать одне від одного. Для того, щоб додати вузол, необхідно здійснити два кроки: приєднання певної машини до загальної мережі та повідомлення про наявність цієї машини іншим. Наприклад, існує спосіб, щоб додати новий сервер в Internet. Він полягає у тому, щоб надати мережну адресу, або символічне ім’я машині, що потім буде заноситися у доменну систему DNS. DNS (англ. Domain Name System) є доменною системою іменування; вона також є протоколом прикладного рівня, який перетворює рядкові адреси серверів Internet у числові IP-адреси.

У загальному, керування набором незалежних комп’ютерів не є основним призначенням розподілених операційних систем, а уявлення однієї узгодженої системи не створюються мережними операційними системами. Найбажанішою можливістю розподілених систем є організація швидкого обміну даних при взаємодії різних програмних компонентів, що у свою чергу знаходяться на різних чи одному комп’ютері. До служб мережної операційної системи необхідно додати

прикладні програмні компоненти для того, щоб побудувати розподілену систему. Це також дозволить покращити прозорість розподілу. Використовуючи додатковий рівень програмного забезпечення, можна певним чином задовольнити вимоги відкритості та масштабованості. Сам додатковий рівень дозволяє у мережних операційних системах приховати різнорідність набору апаратних платформ від користувача. Велику кількість сучасних розподілених систем побудували саме з цим додатковим рівнем. Крім того, його ще можуть називати системою проміжного рівня, платформою розподілу чи проміжним середовищем.

Програмний інтерфейс, який міститься у мережних операційних системах чи у інтерфейсах локальних файлових систем, безпосередньо використовуються розподіленими прикладними програмами.

Однак, у такому підході є проблема: занадто конкретна наявність розподілу. Рівень програмної підтримки знаходиться між мережною операційною системою та прикладною програмою (рис. 1.11).

Рисунок 1.11 — Загальна структура розподілених систем [2]

Розподіленим середовищем є віртуальний обчислюваний простір; він може бути обмеженим однією розподіленою системою чи містити декілька взаємодіючих між собою тих самих розподілених систем.

Для користувача він постає у формі систематизованого сховища інформаційних та програмних ресурсів; також він має певну структуру. У нього доволі зрозуміла система адресації ресурсів і конкретні моделі обчислювальних процесів відповідають конкретним видам робіт.

Бізнес-процеси отримують потрібні ресурси для своєї роботи при звертанні користувача.

2 АРХІТЕКТУРА РОЗПОДІЛЕНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ СПЕЦІАЛІЗОВАНОГО ПРИЗНАЧЕННЯ

2.1 Архітектура комп'ютерної системи

Архітектурою називають концепцію, структуру та функції взаємодії деяких робочих станів в мережі, або визначальний взаємозв'язок. Основною функцією архітектури є визначення головних елементів певної мережі, характеристика загальної логічної організації мережі, опис методів кодування (рис. 2.1). Крім того, інтерфейс користувача і принципи функціонування також визначає архітектура [15].

Рисунок 2.1 — Архітектура комп'ютерної системи [16]

За допомогою блоків даних у мережі відбувається передача інформації, використовуючи процедуру обміну між об'єктами. Ця процедура має назву протокол передачі даних.

Протоколом називається певна сукупність правил, що встановлюють формат процедури обміну даними між двома чи кількома пристроями. Трафік допомагає здійснювати завантаження системи. Трафіком називається певний потік повідомлень у мережі. Це є кількісним виміром у деяких вибраних точках системи, яке виражається у бітах в секунду.

Крім того, істотним впливом на певну характеристику системи є метод доступу. Іншими словами методом доступу являється спосіб, який допомагає

визначити робочу станцію, яка зможе наступною скористатися каналом зв'язку і як здійснювати управління доступом до зв'язку [17].

Інтерфейсом являється певна система уніфікованих апаратних, програмних та конструктивних засобів, які є потрібними для того, щоб організувати надійну взаємодію всіх елементів системи. Комплекс часових та електронних параметрів, які допомагають керувати командами та сигналами є основою інтерфейсу. Також, для інтерфейсу важливий набір конструктивних та системотехнічних рішень та засобів тому, що вони є відображенням структури та порядку обміну інформацією, що здійснюється згідно обраних протоколів обміну даних.

За допомогою технічних засобів забезпечується об'єднання комп'ютерів в одну систему. Сюди відносять маршрутизатори, адаптери, сервери, шлюзи, робочі станції та інше.

У свою чергу, призначенням програмних засобів є управління роботою системи та забезпечення потрібного інтерфейсу з користувачем. Сюди також входять операційні системи мереж.

2.2 Вибір архітектури системи

В загальному існує три види архітектур, а саме: однорангова архітектура, архітектура типу термінал – головний комп'ютер та архітектура типу клієнт – сервер.

Однорангова архітектура (англ. peer-to-peer architecture) є концепцією інформаційної системи, де її ресурси повністю розосереджені по всіх системах [18]. Головною характеристикою цієї архітектури є рівноправність всіх систем (рис. 2.2).

Кожна невелика мережа, де будь-яка робоча станція здійснює і функції файлового сервера, і робочої станції одночасно, може відноситися до однорангових мереж. В однорангових локальних обчислювальних мережах можуть бути загальними і файли на будь-якому комп'ютері і дисковий простір. Для того, щоб ресурс був загальним, потрібно наддати його у загальне користування, використовуючи певні служби віддаленого доступу мережевих однорангових операційних систем. Інші користувачі можуть використовувати файли одразу ж

після їх створення, якщо захист даних було встановлено певним чином. Найкращими для невеликих робочих груп будуть однорангові локально обчислювальні мережі.

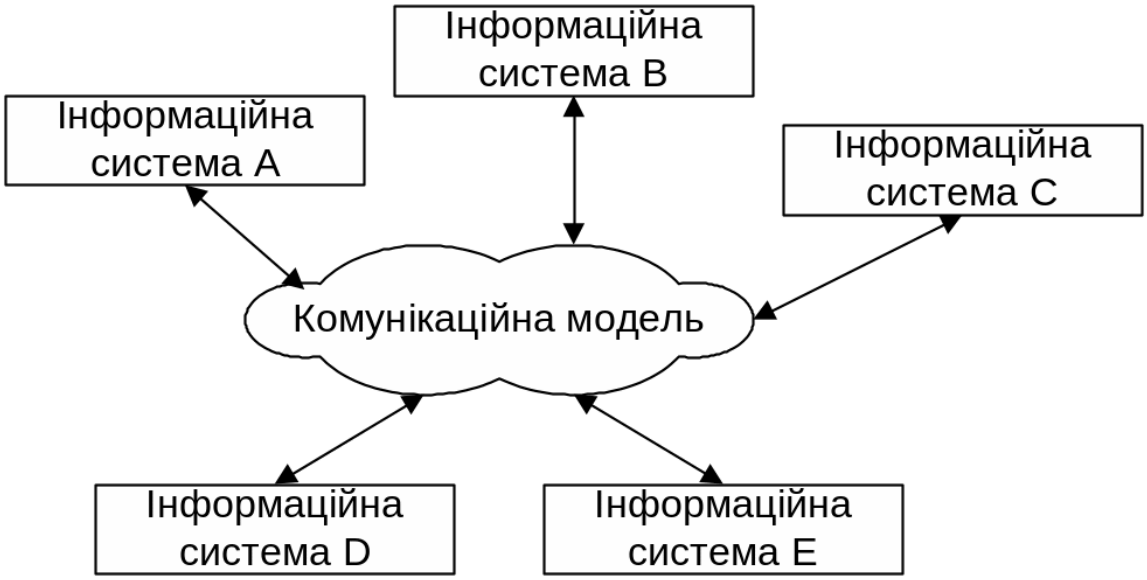


Рисунок 2.2 — Однорангова архітектура [16]

Однорангові локально обчислювальні мережі являються найдешевшим та найпростішим типом мереж для встановлення. Окрім мережевого носія та карти, вони потребують лише операційної системи. Коли комп'ютери з'єднуються, то користувач може надати ресурси та інформацію в спільне користування. Наступними перевагами однорангової архітектури є легкість у встановлені та налаштуванні, незалежність окремих персональних комп'ютерів від виділеного сервера, змога для користувачів контролювати свої ресурси, легка експлуатація, невисока вартість, потреба у мінімальному програмному забезпеченні та устаткуванні, відсутність потреби в адміністраторі.

Однак, також існує і проблема у одноранговій архітектурі. Нею являється відключення комп'ютерів від мережі. При цьому зникають певні види сервісу із мережі. На жаль, одночасно застосувати мережеву безпеку можна тільки до одного ресурсу; при цьому користувач повинен не забувати, що кількість паролів дорівнює кількості мережевих ресурсів. Під час того, як отримується доступ до ресурсу,

відбувається падіння продуктивності комп’ютера. Крім того, помітним недоліком однорангової архітектури є відсутність централізованого адміністрування.

З іншого боку, використання цієї архітектури не обмежує також і використання інших архітектур, тобто архітектура типу термінал – головний комп’ютер та архітектура типу клієнт – сервер.

Архітектурою термінал – головний комп’ютер являється концепцією інформаційної мережі, де повна обробка даних відбувається за допомогою одного чи декількох комп’ютерів (рис. 2.3).

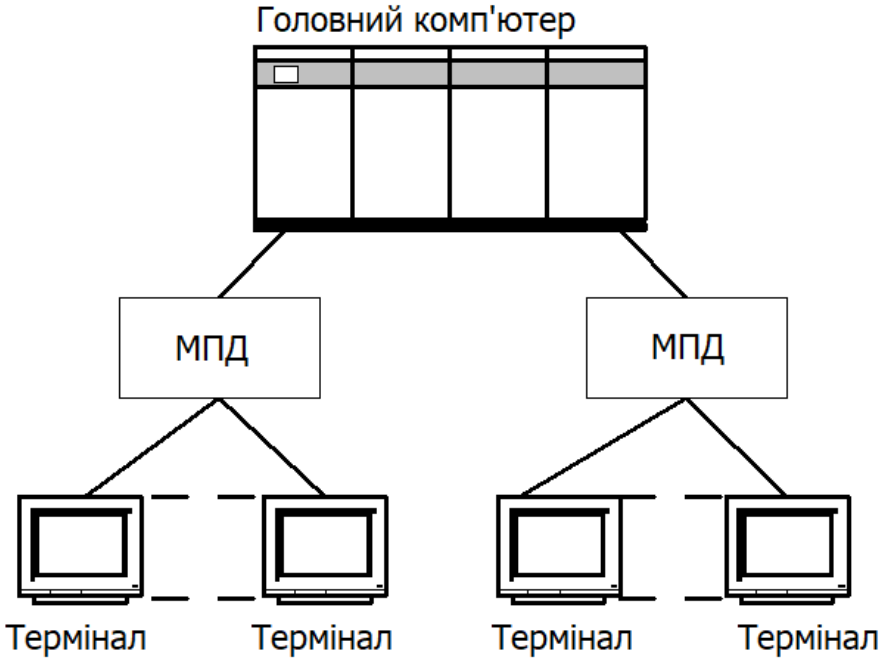


Рисунок 2.3 — Архітектура типу термінал – головний комп’ютер [16]

Ця архітектура має два типи обладнання. Перша, коли є головний комп’ютер, в якому здійснюється мережеве управління, зберігання та обробка даних, а друга – коли присутні призначені для передачі команд головному комп’ютеру для організації сеансів та виконання завдань термінали, які також вводять дані, щоб виконати завдання та отримати результат. Завдяки мультиплексорам передачі даних головні комп’ютери взаємодіють з терміналами (рис. 2.4).

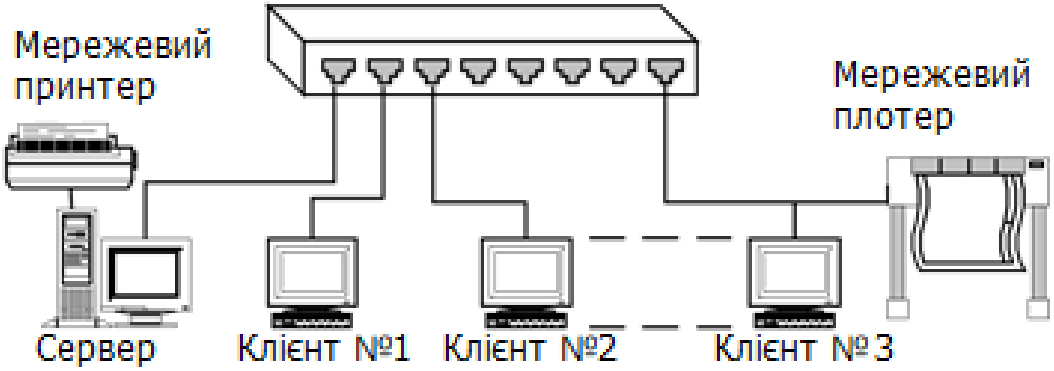


Рисунок 2.4 — Застосування периферійного обладнання [16]

Стандартним прикладом архітектури системи з головними комп'ютерами є системна мережева архітектура (англ. System Network Architecture - SNA) [19].

Архітектурою типу клієнт – сервер являється концепція інформаційної мережі, де головна частина ресурсів повністю зосереджена в серверах, які займаються обслуговуванням своїх клієнтів [20]. Основними компонентами цієї архітектури є клієнт та сервер (рис. 2.5).

Сервером називають об'єкт, який надає сервіс за запитами іншим об'єктам у мережі, а сервісом називають процес обслуговування клієнтів.

Рисунок 2.5 — Архітектура типу клієнт – сервер [16]

Сервер працює згідно завдань, які видавались клієнтами, а також займається управлінням виконання поставлених завдань. Після кожного виконаного завдання

сервер надсилає результати, отриманні в процесі виконання, клієнтові. Сервісною функцією в архітектурі даного типу є опис серверу комплексом прикладних програм. Згідно з цим комплексом виконуються різні прикладні процеси.

Клієнтом називається процес, що здатен викликати сервісну функцію, використовуючи певну операцію; ним також можу бути користувач чи програма. На рисунку 2.6 зображено перелік сервісів в архітектурі даного типу.

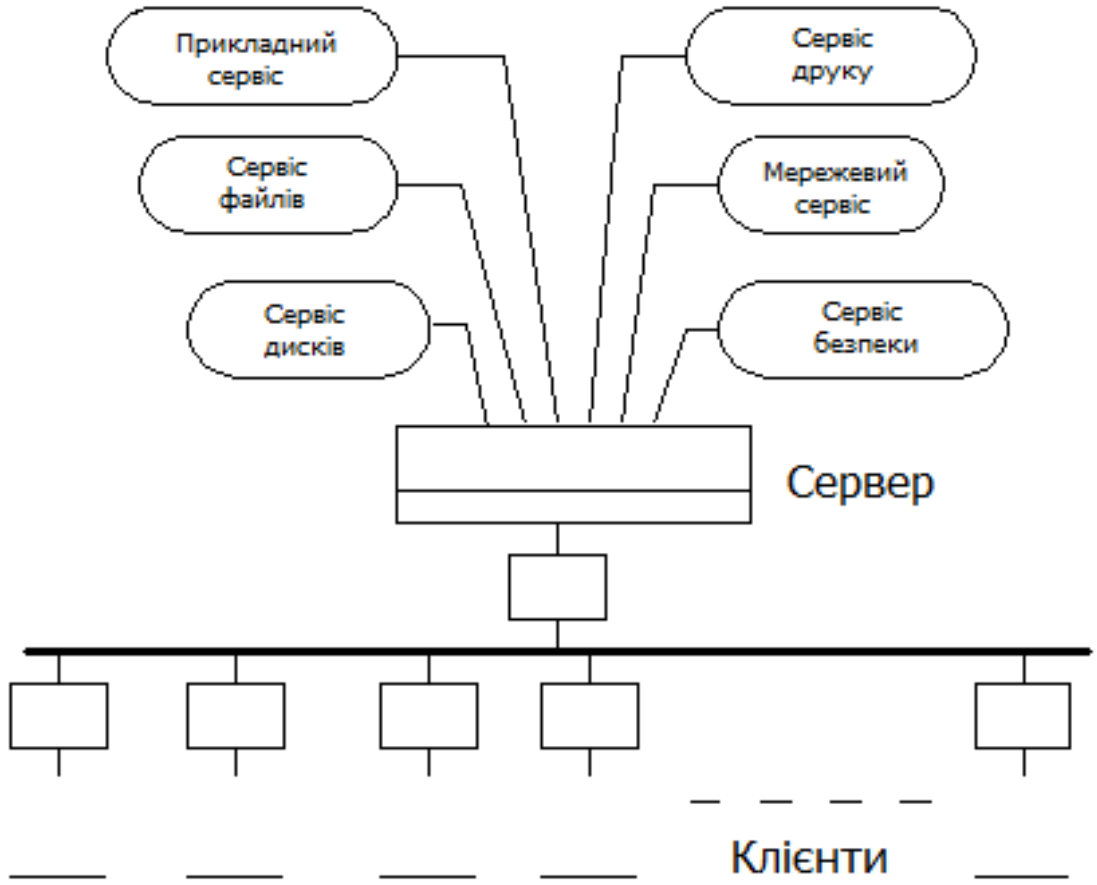


Рисунок 2.6 — Модель клієнт – сервер [16]

Клієнтами називають робочі станції, що користуються ресурсами серверів та мають змогу надавати зручні інтерфейси для користувачів. Також, інтерфейсами користувачів називають процедуру взаємодії того ж самого користувача з мережею чи системою.

Ініціатором використання електронної пошти чи інших сервісів є клієнт. При цьому він може затребувати певний вид обслуговування, отримати необхідні для нього результати, встановити сеанс та повідомляти про кінець роботи.

У деяких мереж є виділений файловий сервер. При цьому на конкретному автономному персональному комп'ютері відбувається встановлення мережевої серверної операційної системи, тому цей персональний комп'ютер перетворюється на сервер. Завдяки встановленому на робочій станції програмному забезпеченню є можливим обмін даних з сервером.

У моделі тонкого клієнта вся обробка додатків і керування даними здійснюється сервером. Клієнт просто відповідає за запуск програмного забезпечення для презентацій.

Вона використовується, коли застарілі системи переносяться на архітектуру клієнтського сервера, в якій застаріла система діє як сервер самостійно з графічним інтерфейсом, реалізованим на клієнті.

У моделі товстого клієнта сервер відповідає лише за керування даними. Програмне забезпечення на клієнті реалізує логіку програми та взаємодію з користувачем системи.

Найбільш підходить для нових систем C/S, де можливості клієнтської системи відомі заздалегідь. Складніша, ніж модель тонкого клієнта, особливо для управління. Нові версії програми мають бути встановлені на всіх клієнтах.

Основним недоліком є те, що це створює велике навантаження на обробку як на сервер, так і на мережу.

Багаторівнева архітектура — це архітектура клієнт-сервер, в якій такі функції, як презентація, обробка додатків і керування даними, фізично розділені. Розділивши програму на рівні, розробники отримують можливість змінити або додати певний шар замість того, щоб переробляти всю програму. Він забезпечує модель, за допомогою якої розробники можуть створювати гнучкі та багаторазові додатки.

Найбільш загальним використанням багаторівневої архітектури є трирівнева архітектура. Трирівнева архітектура зазвичай складається з рівня презентації, рівня програми та рівня зберігання даних і може виконуватися на окремому процесорі.

Рівень презентації — це найвищий рівень програми, за допомогою якого користувачі можуть отримати прямий доступ, наприклад, веб-сторінку або графічний інтерфейс операційної системи (графічний інтерфейс користувача).

Основною функцією цього рівня є переведення завдань і результатів у те, що користувач може зрозуміти. Він взаємодіє з іншими рівнями, щоб розміщувати результати на рівні браузера/клієнта та на всіх інших рівнях мережі.

Рівень програми координує роботу програми, обробляє команди, приймає логічні рішення, оцінює та виконує обчислення. Він контролює функціональність програми, виконуючи детальну обробку. Він також переміщує та обробляє дані між двома навколишніми шарами.

На цьому рівні інформація зберігається та витягується з бази даних або файлової системи. Потім інформація передається назад для обробки, а потім назад користувачеві. Він включає в себе механізми збереження даних (сервери баз даних, спільні файли тощо) і надає API (інтерфейс прикладного програмування) для рівня програми, який надає методи керування збереженими даними.

Перевагами являються: краща продуктивність, ніж підхід з тонким клієнтом, і простіше керувати, ніж підхід з товстим клієнтом, покращує можливість повторного використання та масштабованість – у міру зростання вимог можна додавати додаткові сервери, забезпечує підтримку багатопотокової роботи, а також зменшує мережевий трафік і забезпечує ремонтопридатність і гнучкість.

Недоліками є: незадовільна тестованість через відсутність інструментів тестування, більш критична надійність і доступність сервера.

Однак, також потрібні мережеві прикладні програми, що допомагають реалізувати переваги, які в свою чергу надала мережа. Мережі, які базуються на серверах, володіють кращими характеристиками та вищою надійністю.

Сучасна серверна архітектура має чотири групи об’єктів, а саме сервери, клієнти, дані та мережеві служби. На робочих місцях користувачів в системах розташовані клієнти. В загальному дані зберігаються у серверах. Дані та сервери разом використовують мережеві служби. Також процедури обробки даних керуються мережевими службами.

Системи з архітектурою типу клієнт – сервер мають такі переваги:

1. Ефективний доступ до мережевих ресурсів.
2. Лише один пароль для входу та одержання доступу до всіх ресурсів.
3. Централізоване управління обліковими записами.

4. Спрощене мережеве адміністрування.
5. Організація мереж з багатьма робочих станцій.

Однак, вона також має і наступні недоліки:

1. Висока вартість мережевого устаткування і самих мереж.
2. Несправність сервера може призвести до непрацездатності системи.
3. Можлива втрата мережевих ресурсів.
4. Потрібен кваліфікований персонал.

Архітектурний стиль брокера — це архітектура проміжного програмного забезпечення, яка використовується в розподілених обчисленнях для координації та забезпечення зв’язку між зареєстрованими серверами та клієнтами. Тут об’єктний зв’язок відбувається через систему проміжного програмного забезпечення, яка називається брокером запитів на об’єкт (програмна шина).

Клієнт і сервер не взаємодіють один з одним безпосередньо. Клієнт і сервер мають пряме підключення до свого проксі, який спілкується з посередником-брокером.

Сервер надає послуги шляхом реєстрації та публікації своїх інтерфейсів у брокера, а клієнти можуть запитувати послуги у брокера статично або динамічно шляхом пошуку. CORBA (Common Object Request Broker Architecture) є хорошим прикладом реалізації архітектури брокера.

Брокер відповідає за координацію зв'язку, наприклад, пересилання та пересилання результатів і винятків. Це може бути служба, орієнтована на виклик, документ або посередник, орієнтований на повідомлення, якому клієнти надсилають повідомлення.

Він відповідає за посередництво запитів на послуги, визначення місцезнаходження належного сервера, передачу запитів і відправку відповідей назад клієнтам.

Він зберігає реєстраційну інформацію серверів, включаючи їх функціональні можливості та послуги, а також інформацію про місцезнаходження.

Він надає API для запитів клієнтів, серверів для відповіді, реєстрації або скасування реєстрації компонентів сервера, передачі повідомлень і визначення місцезнаходження серверів.

Заглушки генеруються під час статичної компіляції, а потім розгортаються на стороні клієнта, яка використовується як проксі для клієнта. Проксі на стороні клієнта виступає посередником між клієнтом і брокером і забезпечує додаткову прозорість між ними та клієнтом; віддалений об'єкт виглядає як локальний.

Проксі-сервер приховує IPC (міжпроцесний зв’язок) на рівні протоколу та виконує маршалізацію значень параметрів та скасування маршалювання результатів із сервера.

Скелет створюється компіляцією інтерфейсу служби, а потім розгортається на стороні сервера, який використовується як проксі для сервера. Проксі-сервер на стороні сервера інкапсулює низькорівневі системні мережеві функції та надає високорівневі API для посередництва між сервером і брокером.

Він отримує запити, розпаковує запити, демаршалізує аргументи методу, викликає відповідну службу, а також маршалізує результат перед відправкою його назад клієнту.

Міст може з'єднати дві різні мережі на основі різних протоколів зв'язку. Він є посередником різних брокерів, включаючи брокерів DCOM, .NET Remote і брокерів Java CORBA.

Мости є додатковим компонентом, який приховує деталі реалізації, коли два брокери взаємодіють і беруть запити та параметри в одному форматі та перекладають їх в інший формат.

Послуга – це компонент бізнес-функціональності, який є чітко визначеним, автономним, незалежним, опублікованим і доступним для використання через стандартний інтерфейс програмування. Зв'язки між службами здійснюються за допомогою загальних і універсальних протоколів, орієнтованих на повідомлення, таких як протокол веб-служби SOAP, який може вільно доставляти запити та відповіді між службами.

Сервісно-орієнтована архітектура — це дизайн клієнт/сервер, який підтримує IT-підхід, орієнтований на бізнес, у якому програма складається з програмних служб і споживачів програмних послуг (також відомих як клієнти або запитувачі послуг).

Архітектура, орієнтована на обслуговування, надає наступні функції:

1. Розподілене розгортання — розкриття корпоративних даних та бізнес-логіки як нещільно пов’язані, доступні, структуровані, стандартні, грубозернисті одиниці функціональності без стану, які називаються службами.

2. Можливість компонування — збирання нових процесів з існуючих служб, які доступні з бажаною детальністю через чітко визначені, опубліковані та стандартні інтерфейси скарг.

3. Взаємодія — поширення можливостей та повторне використання спільних послуг в мережі незалежно від базових протоколів або технології впровадження.

4. Можливість повторного використання — вибір постачальника послуг і доступ до наявних ресурсів, які відображаються як послуги.

Перевагами є:

1. Поганий зв’язок орієнтації на обслуговування забезпечує велику гнучкість для підприємств у використанні всіх доступних сервісних ресурсів, незалежно від платформи та технологічних обмежень.

2. Кожен компонент служби незалежний від інших служб завдяки функції служби без стану.

3. Реалізація служби не вплине на застосування служби, доки відкритий інтерфейс не буде змінено.

4. Клієнт або будь-яка служба може отримати доступ до інших служб незалежно від їхньої платформи, технології, постачальників або мовних реалізацій.

5. Можливість повторного використання активів і сервісів, оскільки клієнтам сервісу потрібно лише знати його загальнодоступні інтерфейси, склад сервісу.

6. Розробка бізнес-додатків на основі SOA набагато ефективніша з точки зору часу та витрат.

7. Підвищує масштабованість і забезпечує стандартне з’єднання між системами.

8. Ефективне та ефективне використання «Бізнес-послуг».

9. Інтеграція стає набагато простішою та покращує внутрішню сумісність.

10. Абстрактна складність для розробників і активізація бізнес-процесів ближче до кінцевих користувачів.

У загальному вибір архітектури повністю залежить від того, для чого призначена мережа та скільки робочих станцій вона в собі містить.

Якщо кількість користувачів не більше десяти, немає потреби у сервері спеціалізованого призначення, всі машини знаходяться на невеликій відстані одне від одного та невисокі фінансові можливості, то варто зосередити свою увагу на одноранговій архітектурі.

Якщо є необхідним розділити ресурси на рівні користувача, кількість користувачів більше десяти, потрібен спеціалізований сервер і доступ до глобальної мережі та необхідне централізоване управління чи резервне копіювання, то варто обрати архітектуру типу клієнт – сервер.

2.3 Архітектура розподілених об’єктів

В архітектурі клієнт-сервер, що належить розподіленій системі, є певні відмінності. Наприклад, запити клієнта на рахунок сервісів можливі лише у самого сервера, клієнти зобов’язані володіти знаннями про сервіси, що надаються і про взаємодію певних серверів, а також сервери можуть виконувати певні функції клієнтів, запитуючи сервіси у інакших серверів. Дана модель є чудовим вибором для додатків різноманітного типу, однак у ній присутні певні обмеження для розробників, які у свою чергу, повинні виразити підтримку працездатності та масштабованості засобів, які займаються включенням клієнтів у розподілену систему.

Побудова архітектури системи як архітектури розподілених об’єктів та зменшення різниці між сервером та клієнтом являються основним підходом, що часто використовується у проектуванні розподілених систем. У цій архітектурі головними компонентами даної системи являються об’єкти, що дають список сервісів, використовуючи свої інтерфейси (рисунок 2.7).

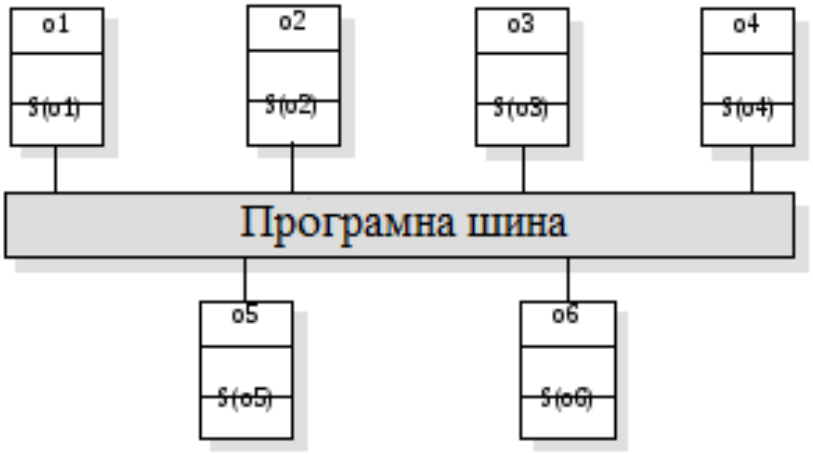


Рисунок 2.7 — Архітектура розподілених об’єктів [21]

Не акцентуючи увагу на різниці між користувачем сервісу та тим, хто його надає, інакші об’єкти мають змогу викликати ті ж сервіси.

Об’єкти можуть знаходитися на різних комп’ютерах у даній мережі, використовуючи проміжне програмне забезпечення для своєї взаємодії. Якщо зазначити аналогію системної шини, що надає дозвіл на поєднання різноманітних пристроїв та підтримку певної взаємодії між апаратними засобами, то можна сказати, що проміжне програмне забезпечення також може бути використаний у якості шини програмного забезпечення. Вона представляє собою певний набір сервісів, що дають змогу іншим об’єктам встановлювати зв’язок один з одним, додавати та видаляти їх з цієї системи. Крім того, у проміжного програмного забезпечення також є інша назва: брокер запитів до об’єкта, завданням якого являється забезпечення інтерфейсу між даними об’єктами [22].

Перевагами архітектури розподілених об’єктів є:

1. Розробникам не потрібно швидко вирішувати місце та спосіб надання сервісів. Об’єкти знаходяться у довільному мережевому вузлі. Іншими словами можна сказати, що відмінність між тонким та товстим клієнтом є незначною, так як відсутня потреба у плануванні розміщенні об’єктів.

2. Дана системна архітектура являється доволі відкритою, тому це надає змогу додати у цю систему потрібні ресурси. Також, можна зазначити, що вдосконалення стандартів даної програмної шини є частим, тому це оновлення дає дозвіл написаним на різноманітних мовах програмування об’єктам надавати сервіси та здійснювати взаємодію між собою.

3. Система являється гнучкою та масштабованою. Створення системних екземплярів зі схожими сервісами допомагає впоратися із системними навантаженнями. Чим більше навантаження, тим більше нових об’єктів можна додати у систему, під час чого інші об’єкти мережі не зупиняють свою роботу.

4. Присутність можливості системної динамічної переконфігурації, що здійснюється мігрувальними у мережі об’єктами. У об’єктів, що здатні надавати сервіси, є можливість мігрування на той самий процесор, що також використовується об’єктами, що ці сервіси запитують. Таким чином продуктивність системи збільшується.

Під час здійснення проектування певних систем архітектура розподілених об’єктів може бути використана двома способами:

1. У вигляді логічної моделі. Вона дає змогу для розробників здійснити структурування та планування системи. При цьому функціональність додатку може бути описана лише за допомогою термінів та різних комбінацій сервісів. Надалі способи присвоєння сервісів розробляються, використовуючи кілька розподілених об’єктів. На цьому рівні здійснюється проектування великомодульних об’єктів, які у свою чергу можуть надати сервіси, які здатні відобразити специфічність кожної області у додатку.

2. У вигляді гнучкого підходу до реалізації. У такій ситуації логічною моделлю системи являється модель клієнт-сервер, де клієнт та сервер представляють собою розподілені об’єкти, що здійснюють взаємодію за допомогою засобів програмної шини. У такому випадку замінна системи є дуже легкою. При цьому і клієнт, і сервер не мають можливості реалізації в одному об’єкті, але вони можуть скласти множину малих об’єктів.

Система, для якої архітектура розподілених об’єктів є підходящою, являється системою обробки даних, які у свою чергу містяться у різноманітних базах даних (рис. 2.8).

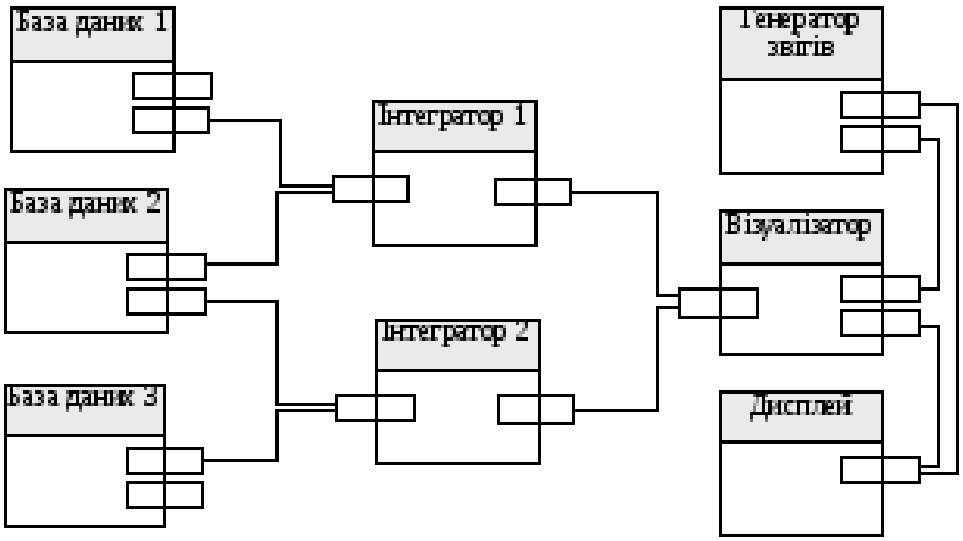


Рисунок 2.7 — Архітектура розподілених системи обробки даних [21]

У такому випадку довільна база даних може бути подана у вигляді об’єкта, що має інтерфейс. Цій базі дані будуть доступні лише у вигляді «читання». Кожен об’єкт-інтегратор має визначений тип залежності між даними, якими він і займається. Об’єкт-інтегратор збирає необхідну інформацію з баз даних, щоб здійснити спробу відслідкувати дані залежності [23].

Об’єкти-візуалізатори взаємодіють разом з об’єктами-інтеграторами, щоб надавати дані у графічному вигляді або складати за цими даними звіти.

Якщо даний додаток є такого типу, то архітектура розподілених об’єктів буде найкращим рішенням по трьох причинах:

1. У системах такого типу єдиний поставщик сервісів, де зосереджене усе управління даними, відсутній.

2. Завдяки додаванню нових об’єктів-інтеграторів стає можливим відслідковування нових типів взаємних залежностей між даними.

3. Можливість збільшення кількість вільних баз даних, при цьому не перервати роботу системи.

Однак, основним недоліком архітектури розподілених об’єктів є складне проектування. Іншими словами, така система має менш природний підхід, щоб створити розподілену систему.

3 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ

3.1 Вимірювання пропускної здатності Tor

На початку Tor, управління довідкою (англ. Directory Authorities) довіряло тільки значенню пропускної здатності, про яке повідомляли реле до тих пір, поки воно було нижчим за публічно відоме значення. Однак, цей підхід був вразливий до атак, що були засновані на помилкових звітах. Проект TorFlow представив управління пропускною здатністю (англ. Bandwidth Authorities), що вимірює, а потім обирає на дійсній пропускній здатності реле. Сценарій вимірювання під назвою спідрейсер (англ. speedracer) поділяє мережу на частини реле з подібною пропускною здатністю, а вже потім ще раз отримує великий файл за допомогою схеми з двома стрибками, яка була створена з реле у цьому фрагменті.

Відношення середньої ємності потоку конкретного реле до решти фрагмента пізніше використовується для регулювання пропускної здатності, що повідомляється самостійно. Цей підхід має недоліки, хоча все й використовується в Tor. Вимірювання за двома ланцюгами стрибків спричиняє вплив на результат вимірювання від реле однакового розміру, які розміщені в одному зрізі. Крім того, через життєвий цикл реле, час релейного вузла, зареєстрований в управлінні довідкою, також впливає на результат вимірювання.

Довірене середовище виконання (англ. Trusted Execution Environments) захищає виконання критичного для безпеки коду програми від зловмисника з повним контролем над системою. Вони дають змогу виконувати програму ізольовано від решти системи, а також вони оснащені апаратним захищеним секретом, який дозволяє атестувати та герметизувати.

Герметизація описує процес зберігання даних у постійному сховищі таким чином, що дозволяє лише захищеній програмі отримати повторний доступ до цих даних. Це здійснюється шляхом отримання ключа запечатування з апаратного захищеного секрету, який є унікальним для кожної захищеної програми. Атестація використовується для перевірки того, що програма дійсно працює в справжньому середовищі. Для цього довірене середовище виконання створює дані атестації, які

описують його стан (наприклад, завантажений програмний код), а також можуть включати користувацькі дані, наприклад ключі, створені захищеною програмою. Ці дані потім підписуються за допомогою ключа атестації, отриманого з секретного апаратного забезпечення, що дозволяє перевірити ці дані за допомогою пов'язаного відкритого ключа. Цей підпис може бути зв'язаним, що дозволяє об'єктам виявляти, чи були створені два підписи одним довіреним середовищем виконання, не розкриваючи додаткової ідентифікаційної інформації.

Хоча ці середовища можуть бути створені за допомогою різних апаратних платформ, таких як TrustZone для процесорів ARM або Sanctum для платформ RISC-V, було створено екземпляр нашої системи за допомогою Intel SGX.

Блокчейн — це децентралізована структура даних, яка зберігає інформацію в незмінній узгодженій послідовності, що поширюється на всю мережу.

Інформація розділена на блоки, що складаються із записів даних і заголовків, з'єднаних разом, оскільки кожен заголовок містить хеш заголовка попереднього блоку, запобігаючи змінам у блоках. Доступ до даних можуть отримати користувачі, які володіють гаманцями у вигляді адрес, представлених хешем відкритого ключа. Поки дані доступні безкоштовно, вставка даних за допомогою транзакцій (підписані ключем гаманця) зазвичай призводить до комісії за транзакцію.

Майнери перевіряють транзакції та включають їх у нові блоки. Вони дотримуються протоколу консенсусу на основі однорангового зв'язку, щоб досягти згоди щодо єдиної історії записів даних. Більшість протоколів консенсусу спираються на алгоритми Proof-of-Work, які змушують їх вирішувати. Для досягнення консенсусу розглядається найдовший ланцюг як дійсний, і кожен майнер працює на найдовшому доступному ланцюжку. Таким чином, в кінцевому підсумку з'являються різні версії блокчейну.

Смарт-контракти — це програми, що зберігаються на блокчейні. Вони можуть викликатися для виконання дії, відправивши транзакцію їх адреса. Виконання коду є витратним з точки зору обчислень виконується кожним, хто перевіряє відповідний блок. Отже, багато систем вимагають плати за виконання коду смарт-контрактів. До обмежень смарт-контрактів можна віднести відсутність

більш складних структур даних або операції з плаваючою комою навіть у смарт-контрактів високого рівня мови, наприклад Solidity. Крім того, вони не можуть займати синхронне спілкування і не мають реального поняття про час окремо від позначки часу в заголовку блоку, роздільна здатність якого обмежена до секунд або навіть хвилин.

3.2 Модель системи

Модель системи зображена на рисунку 3.1. Вона включає наступні компоненти інфраструктури Tor: релейні вузли, приховані в Tor веб-сервіси і клієнти Tor. У SmarTor функціонал управління довідкою та управління пропускною здатністю розподіляються між двома типами суб'єктів: смарт-контракт та вимірювач пропускної здатності.

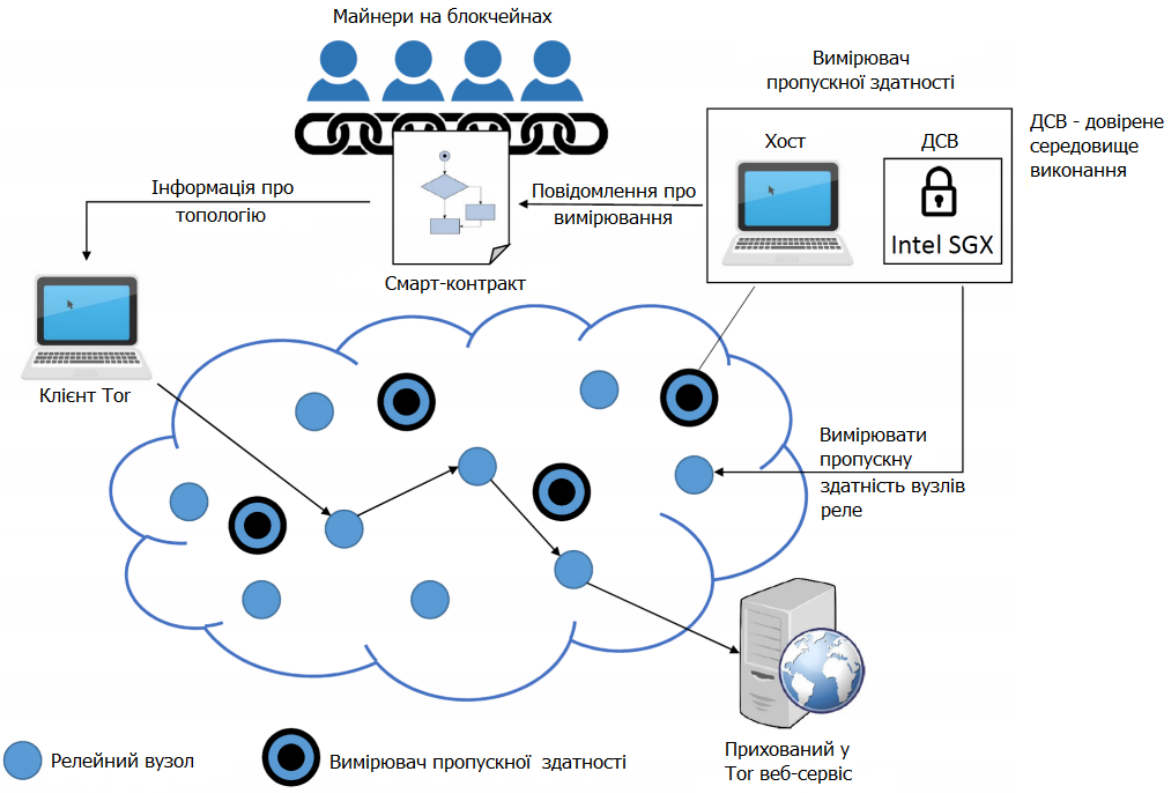


Рисунок 3.1 — Модель системи SmarTor

Смарт-контракти — це обчислювальна програма, що зберігається в блокчейні, тоді як вимірювач пропускної здатності керується волонтерами,

наприклад, особами, які вже керують реле Tor, валідаторами блокчейну або будь-якими іншими користувачами. Можна припустити, що платформи вимірювача пропускної здатності оснащені підтримуваним апаратним забезпеченням довіреним середовищем виконання, де програми можуть виконувати чутливі до безпеки операції (наприклад, шифрування, підписання) ізольовано від решти системи.

Припущення про довіру щодо релейних вузлів, прихованого у Tor веб-сервісу і Tor клієнта успадковані від поточної системи Tor – Tor клієнт і прихований у Tor веб-сервіс вважаються надійними, тоді як деякі (але не більшість) релейних вузлів можуть бути шкідливими. Як і поточний браузер Tor, код, що виконується клієнтом Tor, смарт-контрактом або довіреним середовищем виконання, розробляється довіреною сутністю, і кожен може переглядати його та отримувати доступ після його публікації. Як зазвичай, довіри до смарт-контракту немає, але можна припустити, що жоден зловмисник не може здійснити 50%-атаку на базовий блокчейн.

Крім того, можна зробити наступні припущення щодо вимірювача пропускної здатності. Платформа хоста вимірювача пропускної здатності не є надійною. Проте, можна довірити постачальнику довіреного середовища виконання, що постачає відповідне безпечне обладнання та підтримує його за допомогою інфраструктури відкритих ключів. Після введення в експлуатацію деякі з вимірювачів пропускної здатності у довіреному середовищі виконання можуть бути скомпрометовані, тобто їх апаратний секрет може бути розкритий. Однак, можна припустити, що кількість скомпрометованих вимірювачів пропускної здатності у довіреному середовищі виконання обмежена.

Потрібно зауважити, що апаратні атаки на захищене обладнання є технічно складними та дорогими. Пов’язані атаки коштують до 1 000 000 доларів США при роботі з захищеними від несанкціонованого доступу пристроями, такими як смарт-картки. Хоча досі не повідомлялося про успішні апаратні атаки на Intel SGX і пов’язані з цим витрати, однак існує можливість атак з боку каналів на анклави Intel SGX. Ці атаки є менш витратними, ніж компрометація обладнання, але вони технічно задіяні та часто націлені на конкретні програми, не захищені від них.

Однак, все одно необхідно враховувати, що певна частина довіреного середовища виконання може бути скомпрометована.

Навіть для сильного зловмисника неможливо порушити безпечну апаратну інфраструктуру відкритих ключів або базовий блокчейн. Перш за все, ці атаки скомпрометують усі рішення, засновані на цій технології, руйнуючи основні бізнес-моделі. Крім того, якщо одне технологічне рішення скомпрометовано, SmarTor можна перенести на іншу систему. Можна зауважити, що описані атаки є набагато складнішими, ніж скомпрометувати три управління пропускною здатністю, не захищені безпечним обладнанням, що достатньо для компрометації поточної системи. Зловмиснику з такими можливостями легше написати експлойт, спрямований безпосередньо на програмне забезпечення клієнта Tor, змусивши його розкрити свою ідентичність.

Наш суперник прагне вплинути на інформацію про мережеву інфраструктуру Tor з метою, наприклад, запустити атаки відмови в обслуговуванні або сприяти деанонімізації користувачів. Зокрема, зловмисник може ввести «вузли-привиди» ретрансляції, які фізично не існують, але вказані як доступні вузли в інформації про топологію мережі. Крім того, зловмисник може спробувати збільшити шанси зловмисного релейного вузла бути обраним для пересилання, налаштувавши свої власні вимірювачі пропускної здатності і повідомляючи про високу пропускну здатність для релейних вузлів, які він контролює, і погану пропускну здатність або статус «недоступний» для будь-якого іншого релейного вузла. Під час експлуатації власного вимірювача пропускної здатності, супротивник має повний контроль над хостом вимірювача пропускної здатності і може маніпулювати будь-якими входами та виходами, що надходять до/з довіреного середовища виконання вимірювача пропускної здатності. Наприклад, він може спробувати відстрочити мережеві пакети, щоб штучно зменшити вимірювану пропускну здатність, відтворити результати вимірювань з попередніх раундів вимірювань або спробувати маніпулювати звітами, створеними вимірювачем пропускної здатності у довіреному середовищі виконання. Противник також може скомпрометувати вимірювач пропускної здатності у довіреному середовищі виконання на своїй

платформі, але обмежений у кількості вимірювачів пропускної здатності, які він повністю контролює.

3.3 Дизайн системи

Ідея високого рівня SmarTor полягає в тому, щоб реалізувати функціональні можливості управління довідкою у смарт-контрактах, щоб далі розподілити довіру на основні компоненти інфраструктури та змінити основні припущення довіри.

Функціональність, яку можна реалізувати в смарт-контрактах, обмежена. Зокрема, неможливо реалізувати функціональні можливості управління пропускною здатністю, які перевіряють рекламовану пропускну здатність, оскільки це вимагатиме синхронного зв’язку по мережі, точного таймера та можливості захисту криптографічних ключів за контрактом. Крім того, необхідно враховувати той факт, що обчислення на основі смарт-контрактів є досить дорогими. Тому необхідно зробити їх максимально простими.

Підхід до вирішення цих проблем полягає в тому, щоб передати функціональність управління пропускною здатністю третім сторонам, які ми називаємо інструментами вимірювання пропускної здатності. Вимірювач пропускної здатності виконують вимірювання в циклах вимірювань, організованих смарт-контрактами, і звітують про них для агрегації. Оскільки можна припустити, що вимірювачі пропускної здатності є недовіреними, можна ввести надлишковість у систему та виконати статистичний аналіз звітів, наданих багатьма вимірювачами пропускної здатності, щоб виявити неправильну поведінку. Цей аналіз дозволяє смарт-контракту підтримувати цінність репутації для кожного вимірювача пропускної здатності і виключати вимірювачі пропускної здатності, що погано працюють. Крім того, можна представити безпечне обладнання, яке значно підвищує планку атак на вимірювачі пропускної здатності. Такий підхід дозволяє зменшити надмірність і спростити обчислення.

Зображення огляду високого рівня системи міститься на рисунку 3.2, який ілюструє залучені сутності та їх взаємодію.

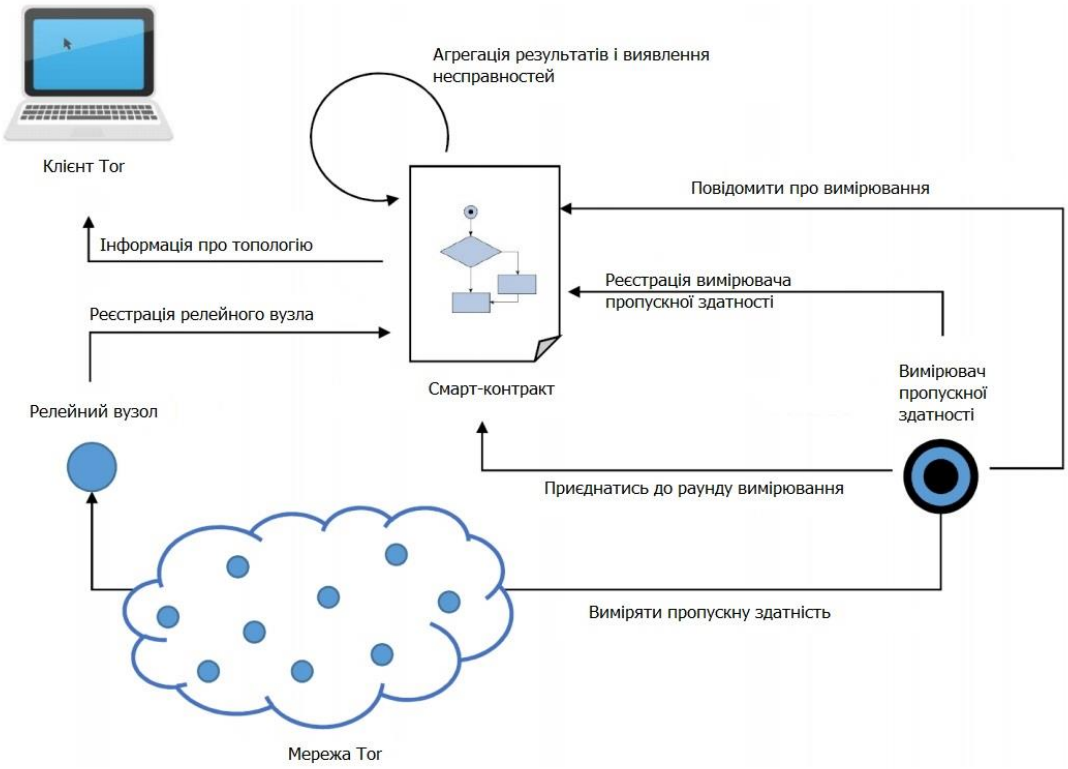


Рисунок 3.2 — Функціональність системи SmarTor

Як згадувалося раніше, хост вимірювача пропускної здатності не є надійним. Проте вимірювач пропускної здатності у довіреному середовищі виконання не має доступу до мережі, і він не може взаємодіяти з іншими об’єктами (наприклад, з смарт-контрактом) самостійно.

Через це вся комунікація вимірювача пропускної здатності у довіреному середовищі виконання із зовнішнім світом має здійснюватися за допомогою хоста вимірювача пропускної здатності.

Кожна сутність має доступний набір ключів. Їх позначення та використання показано у таблиці 3.1.

Ключі сеансу Tor є тимчасовими і що хост вимірювача пропускної не має доступу до ключів, які зберігаються у вимірювачі пропускної здатності у довіреному середовищі виконання.

Таблиця 3.1 — Доступні ключі об’єктів з їх позначеннями, використанням і криптосистемою

Назва ключа	Позначки	Криптосистема	Підтримується	Використання
Гаманцевий ключ	secw pubw	EC(secp256r1)	Вимірювач пропускної здатності у довіреному середовищі виконання, релейні вузли	Підпис транзакції до смарт-контракту
Ключ ідентифікації Tor	sect pubt	RSA2048	Вимірювач пропускної здатності у довіреному середовищі виконання, релейні вузли	Обмін ключами для створення secc
Сесійний ключ Tor	secc	AES128-GCM	Вимірювач пропускної здатності у довіреному середовищі виконання, релейні вузли	Захист комунікації сеансу Tor
Ключ ущільнення	secs	AES256-GCM	Вимірювач пропускної здатності у довіреному середовищі виконання	Шифрує дані в постійне сховище

Кінець таблиці 3.1 – Доступні ключі об’єктів з їх позначеннями, використанням і криптосистемою

Ключ атестування	(seca, puba)	EPID	Вимірювач пропускної здатності у довіреному середовищі виконання	Підпис даних атестації

Трансакції від вимірювача пропускної здатності до смарт-контакту створюються вимірювачем пропускної здатності у довіреному середовищі виконання і підписуються ключем гаманця secw. Хост вимірювача пропускної здатності пересилає ці транзакції майнерам, які потім включають їх у блокчейн. Аналогічно, хост вимірювача пропускної здатності надає поточну версію блокчейну вимірювача пропускної здатності у довіреному середовищі виконання, який перевірить її. Повідомлення між вимірювачем пропускної здатності і релейним вузлом шифруються та аутентифікуються за допомогою ключа сеансу секунд. Хост вимірювача пропускної здатності пересилає будь-які вхідні повідомлення до вимірювача пропускної здатності у довіреному середовищі виконання і будь-які вихідні повідомлення на вказаний релейним вузлом.

Мета реєстрації релейних вузлів - повідомити смарт-контракт про те, що новий релейний вузол бажає приєднатися до мережі, і надати свою контактну інформацію користувачам Tor. Для цього новий релейний вузол надсилає реєстраційну транзакцію τRN, підписану гаманцевим ключем, до смарт-контракту. Ця транзакція включає, як і в поточній системі, інформацію про доступ релейних вузлів, як-от ip, tor_identity_key та exit_policy. Після отримання смарт-контракт зберігає ці дані в списку релейних вузлів, що дозволяє користувачам Tor використовувати це реле в ланцюгах. Це також гарантує, що лише транзакції, підписані одним і тим же ключем гаманця, можуть змінити цю інформацію пізніше.

Перш ніж брати участь у вимірюванні, вимірювач пропускної здатності має зареєструватися в смарт-конракту, як показано на рисунку 3.3. Реєстраційний

вимірювач пропускної здатності видає реєстраційну транзакцію τBM і відправляє її в смарт-контракт. Далі один із вже зареєстрованих вимірювачів пропускної здатності перевіряє дані атестації, щоб переконатися, що його платформа має справжне довірене середовище виконання. Це повторено кожним зареєстрованим вимірювачем пропускної здатності, який потім подає свій голос щодо надійності τvote до смарт-контракту, який перевірить та підрахує ці голоси. Якщо більшість зареєстрованих вимірювачів пропускної здатності проголосувала вірно, смарт-контракт додає його до списку всіх зареєстрованих вимірювачів пропускної здатності.

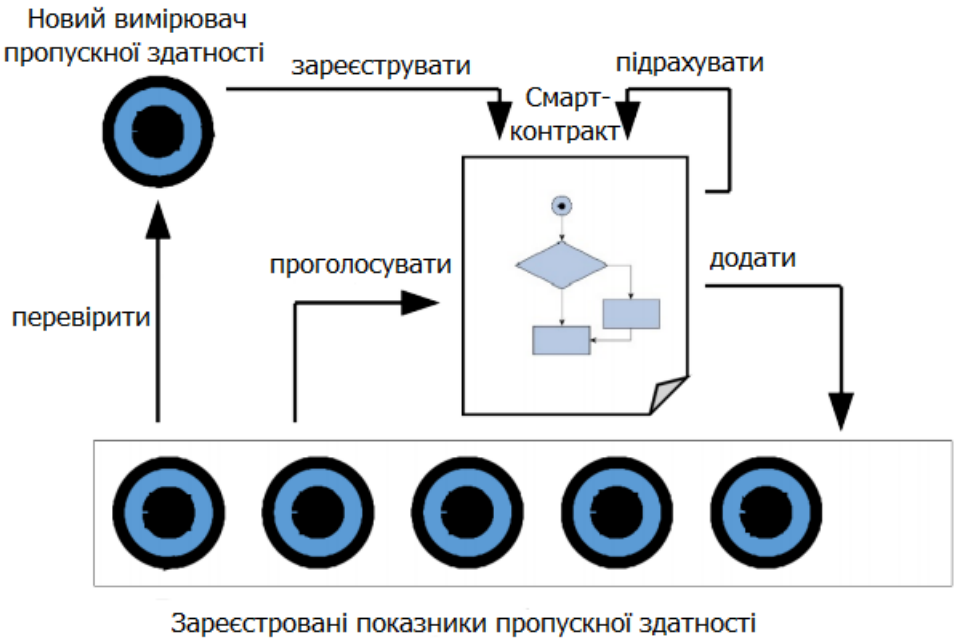


Рисунок 3.3 — Огляд успішного процесу реєстрації

Перед реєстрацією вимірювача пропускної здатності у довіреному середовищі виконання новий вимірювач пропускної здатності створює гаманцевий ключ та запечатує його для постійного зберігання. Після цього він створює τBM, включаючи tor_identity_key новий вимірювач пропускної здатності і пов'язані attestation_data, включаючи pubw.

Ці дані атестації використовуються для підтвердження того, що secw дійсно створено в рамках справжнього довіреного середовище виконання і що жодного

іншого гаманцевого ключа (що представляє вимірювач пропускної здатності) не було створено з цього довіреного середовища виконання. Оскільки ця перевірка є дорогою з точки зору обчислень, смарт-контракт перевіряє лише початкову групу вимірювачів пропускної здатності (фаза завантаження). Після цього (фаза виробництва) цей процес передається вже зареєстрованим вимірювачем пропускної здатності. Кожен вимірювач пропускної здатності у довіреному середовищі виконання повинен періодично запитувати в блокчейн дані атестації нового вимірювача пропускної здатності і перевіряти їх. Після цього він створить τvote, включаючи verification_result і last_blockhash (хеш останнього блоку, прочитаний вимірювачем пропускної здатності у довіреному середовищі виконання), щоб забезпечити свіжість.

Смарт-контракт збереже цей результат, якщо τvote був підписаний зареєстрованим вимірювачем пропускної здатності і last_blockhash є досить недавнім. Крім того, смарт-контракт обчислює відстань між блоками, що містять τBM і τvote. Якщо це перевищує попередньо визначену межу дівтвот (англ. distvote), запускається процес підрахунку голосів. Якщо більшість отриманих голосів була за новий вимірювач пропускної здатності, він додається до списку зареєстрованих вимірювачів пропускної здатності, що дозволяє йому брати участь у раундах вимірювань. Додаткові голоси після цього процесу відхиляються.

Процес вимірювання, як показано на рисунку 3.4, виконується раундами організованими смарт-контрактами. Кожен вимірювач пропускної здатності має активно приєднатися до групи вимірювань, яка починається, як тільки буде доступна достатня кількість вимірювачів пропускної здатності. Смарт-контракт створює різні групи для вимірювачів пропускної здатності з різною пропускною здатністю та призначеними відповідними релейного вузла до нього. Результати вимірювань повідомляються смарт-контрактами, який потім зведе їх до нового консенсусного значення для кожного релейного вузла.

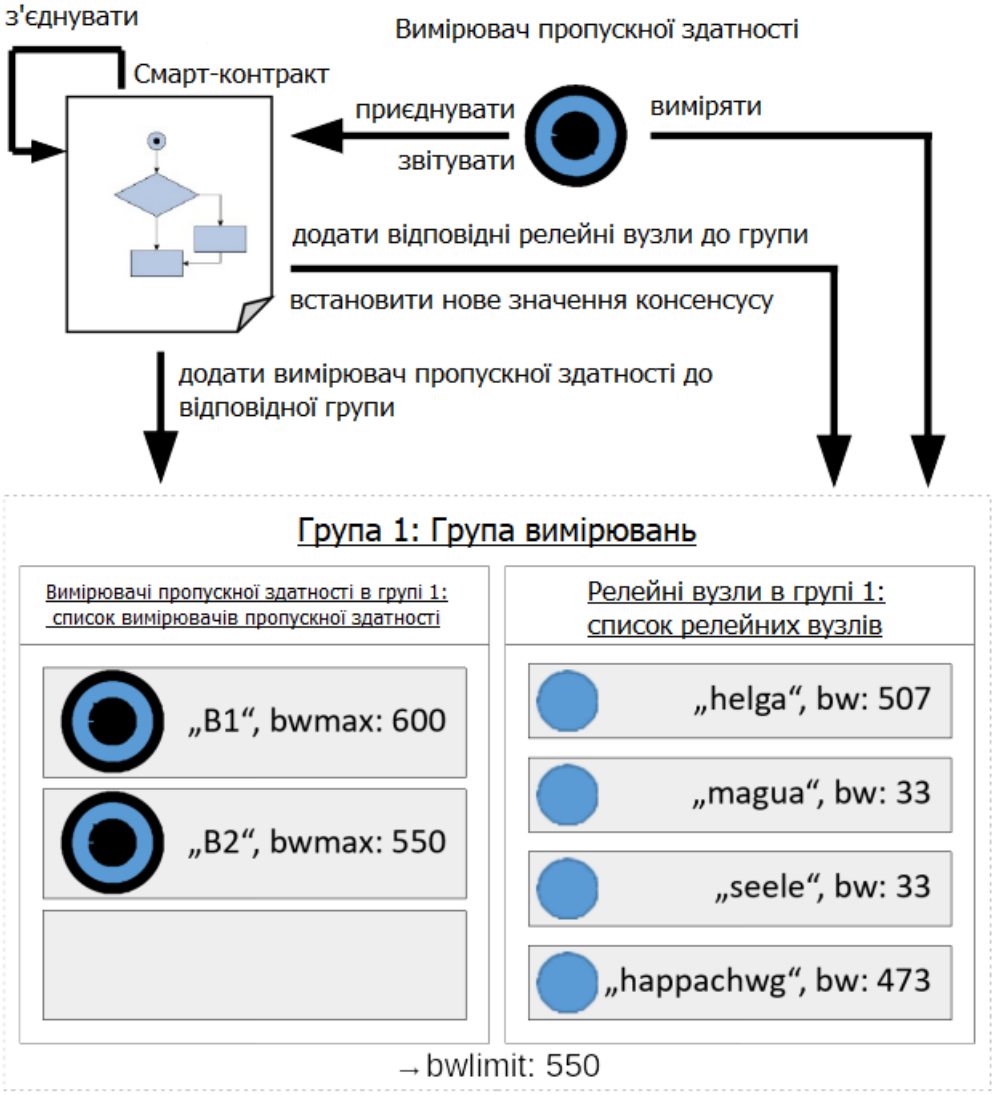


Рисунок 3.4 — Огляд успішного раунду вимірювання

Щоб приєднатися до раунду вимірювання, вимірювач пропускної здатності у довіреному середовищі виконання створює транзакцію приєднання τ_{join}, підписану ключем гаманця. Ця транзакція включає rnd, ctr і bwmax, де rnd — випадкове число, ctr — кількість транзакцій приєднання, створених цим вимірювачами пропускної здатності, а bwmax — максимальна пропускна здатність, яку може виміряти цей вимірювач пропускної здатності.

Після отримання транзакції смарт-контракт перевіряє, чи може цей вимірювач пропускної здатності брати участь у раундах вимірювання і чи монотонно збільшився ctr на одиницю порівняно з попередньою транзакцією. Останній крок має важливе значення для запобігання можливим спробам підбору

певного значення rnd. Якщо перевірка успішна, смарт-контракт випадковим чином поміщає його в одну з груп вимірювань, що очікують на розгляд.

Після того, як n людей приєдналися до відкритої групи вимірювань, смарт-контракт випадковим чином призначає їй r реле. Для цього смарт-контракт випадковим чином вибирає релейний вузол зі списку і додає його до групи, якщо це не порушує властивості групи. Таким чином, кожен вимірювач пропускної здатності повинен мати можливість виміряти цей релейний вузол. Остання виміряна пропускна здатність (bw) цього релейний вузол має бути нижчим за bwsize, що є найнижчим значенням bwmax вимірювача пропускної здатності у цій групі. Якщо релейний вузол не було виміряно раніше або останнє вимірювання було невдалим, bw не встановлюється. У цій ситуації релейний вузол додається до групи, якщо не існує обмеженої кількості u таких релейний вузол. Цей процес повторюється до тих пір, поки група не заповниться.

Після приєднання до групи вимірювач пропускної здатності має запитати стан блокчейн та чекати інформації про призначені реле. Коли вони будуть доступні, розпочнеться процес вимірювання. Під час процесу вимірювання вимірювач пропускної здатності неодноразово отримує файл за користувацькими ланцюгами і виробляє результати транзакції τres, які можуть бути переслані до смарт-контракт.

На відміну від поточного підходу, схеми вимірювання в SmarTor складаються з трьох релейних вузлів. Виміряний релейний вузол є середнім вузлом, і два більш швидкі релейні вузли використовуються як вхід і вихід. Оскільки вимірюваний релейний вузол пропонує найнижчу пропускну здатність, пропускна здатність всієї схеми.

Вимірювач пропускної здатності у довіреному середовищі виконання випадковим чином вибирає релейний вузол входу та виходу. Оскільки пропускна здатність може змінюватися, вхід і вихід повинні мати набагато більшу пропускну здатність, ніж релейного вузла, яку потрібно виміряти. Якщо таких реле немає, як це, то має місце для найшвидших релейних вузів, вимірювач пропускної здатності у довіреному середовищі виконання випадковим чином вибирає релейний вузол з найшвидших доступних релейних вузлів.

Вимірювач пропускної здатності у довіреному середовищі виконання визначає порядок, у якому релейні вузли вимірюються випадковим чином. Після створення схеми вимірювач пропускної здатності підключатиметься через Tor до одного сервера із попередньо визначеного списку. Потім він неодноразово отримує файл за цією ланцюгом; середня пропускна здатність потоку під час цього процесу є результатом вимірювання.

У випадкових точках під час процесу вимірювання вимірювач пропускної здатності у довіреному середовищі виконання також вимірює h релейних вузлів з такою ж пропускною здатністю, як і інші релейні вузли. Ці вимірювання діють як маніпулятори для виявлення проблем із підключенням і пересиланням повідомлень хоста вимірювача пропускної здатності. Вимірювач пропускної здатності у довіреному середовищі виконання зупиняє процес вимірювання, якщо виміряна смуга пропускання цих релейних вузлів не збігається з останнім виміряним значенням.

Після успішного збору всіх вимірювань вимірювач пропускної здатності у довіреному середовищі виконання створює результат транзакції τres. Хост вимірювача пропускної здатності має переслати цю транзакцію до смарт-контракту, який перевіряє ці голоси та призначає нещодавно призначені значення пропускної здатності.

Результатова транзакція τres містить результати вимірювань усіх виміряних реле. Ця транзакція (як і будь-яка інша) є атомарною, що означає, що або кожен результат, або жоден може бути переданий до смарт-контракту. Додаткова інформація в цій транзакції включає лічильник, який описує кількість разів, коли вимірювач пропускної здатності у довіреному середовищі виконання запускав сценарій вимірювання, і блок-хеш останнього блоку в блокчейні, зчитований під час цього процесу. Отримавши τres, смарт-контракт перевіряє, що вимірювач пропускної здатності може повідомити результати для цієї групи, а блокхаш (англ. blockhash) є блоком після блоку, який було запущено групою. Він також підтверджує, що лічильник монотонно збільшується на одиницю, щоб гарантувати, що хост вимірювача пропускної здатності не може впливати на пропускну здатність, запускаючи процес вимірювання кілька разів. Якщо жодних подальших

результатів не бракує, або відстань між поточним блоком та ініціюванням групи перевищує попередньо визначені межі розподілу, смарт-контракт об'єднає результати.

Якщо більше половини призначених вимірювачів пропускної здатності не повідомили результати, результати раунду вимірювання відкидаються. В іншому випадку, як і в поточному методі агрегації, кожному виміряному релейному вузлу призначається (нижнє) середнє значення результатів вимірювання як нове значення консенсусу пропускної здатності.

Можна припустити, що певна частина довіреного середовища виконання може бути скомпрометована. Як контрзахід, потрібно виключити результати вимірювання невірної поведінки вимірювача пропускної здатності в агрегації та змусити їх знову пройти реєстрацію. Під час агрегації смарт-контракт обчислює оцінку репутації rBM для кожного вимірювача пропускної здатності, яка використовується як міра надійності. Чим нижчий показник репутації, тим вище шанс виключення вимірювача пропускної здатності зі списку зареєстрованих вимірювачів пропускної здатності.

Після об'єднання кожен вимірювач пропускної здатності групи отримує бал активності за участь у цьому раунді вимірювання. Крім того, він отримує додатковий бал репутації, якщо його звітні вимірювання узгоджуються з консенсусним значенням. Співвідношення балу репутації до балу репутації можна розглядати як показник того, наскільки схоже вимірювач пропускної здатності поводиться порівняно з іншими вимірювачами пропускної здатності і використовується як rBM. Після повідомлення результату пропускної здатності смарт-контракт має ймовірність виключення вимірювача пропускної здатності з групи зареєстрованих вимірювачів пропускної здатності. Це робиться шляхом випадкового вибору числа $rnd \in [0, 999] \subset Z$. Вимірювач пропускної здатності виключається, якщо $rnd \leqslant f(rBM)$, де $f : [0, 1] \subset R \rightarrow [0, 1000] \subset N$ є монотонна функція, яка використовується для згладжування кривої ймовірності для виключення. Потрібно звернути увагу, що діапазон значень f означає, що будь-який

вимірювач пропускної здатності має шанс виключення 0,1% незалежно від його репутаційного балу.

Це гарантує, що кожен вимірювач пропускної здатності в певний момент повинен пройти повторну перевірку з боку інших. Якщо відомо, що довірене середовище виконання скомпрометований, його ключі відкликаються, що запобігає повторному приєднанню під час етапу перевірки при реєстрації.

Можна запропонувати використовувати f (rBM) := ⌊(1−rBM)2· 1000⌋ як функцію виключення. Імовірність виключення після кількох раундів вимірювання показана на рисунку 3.5. У той час як вимірювач пропускної здатності з репутаційним балом 50% має 90% ймовірність виключення після 8 раундів, чесний вимірювач пропускної здатності (rBM = 95%) має лише 26% шанс виключення після 100 раундів. Зауважте, що цю функцію можна реалізувати за допомогою базових операцій, і rBM = 100% малоймовірний, оскільки пропускна здатність змінюється з часом.

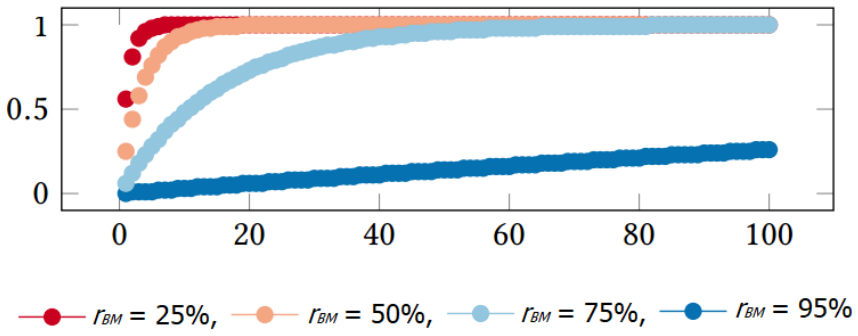


Рисунок 3.5 — Ймовірність виключення (вісь y) після участі в кількох раундах вимірювань (вісь x)

3.4 Аналіз безпеки

Основна мета безпеки даного рішення – перешкодити зловмисникам штучно збільшувати їх пропускну здатність без надання пропорційної кількості ресурсів. Системний параметр, представлений у таблиці 3.2.

Таблиця 3.2 — Використані параметри і запропоновані значення.

Параметр	Позначення	Пропоноване значення
Вимірювачі пропускної здатності на групу	n	9
Релейні вузли на групу	r	15
Максимально нові релейні вузли на групу	u	2
Маніпулятори на раунд	h	2
Час очікування голосування	distvote	256
Час очікування звіту	distres	256

Оскільки вимірювачі пропускної здатності випадковим чином розподіляються до певної групи, існує відмінна від нуля ймовірність дс $\in [0, 1] \subset R$, що зловмисник керує багатьма вимірювачами пропускної здатності та отримує достатню кількість з них в одній групі, щоб представляти більшість у цій групі. Таку групу можна називати скомпрометованою і позначати $p \in [0, 1] \subset R$ для відсотка сутностей під контроль зловмисника. Оскільки кожен вимірювач пропускної здатності має однакову ймовірність бути вибраним для певної групи, дс слід біноміальному розподілу з дс = $\sum_{\lfloor i=n/2 \rfloor}^{i=n} \binom{n}{i} p^i q^i$, q := 1 − p, i j := n − i.

Оскільки дс < p, більшість груп не скомпрометовані, доки зловмисник не контролює більшість сутностей.

На рисунку 3.6 показується ймовірності скомпрометувати певну групу для різних p і n. Наприклад, для p = 10% і n = 9 ймовірність успіху атаки становить 0,09%, що мало. Навіть зловмисник з p = 40% має лише 26,66% шанс обігнати

групу, якщо використовує n = 9. Однак це означатиме, що зловмиснику потрібно буде скомпрометувати 120 = 40% · 300 систем, якщо припустити, що, як у поточній системі існує близько 6000 релейних вузлів, і лише 5% з них беруть участь як вимірювачі пропускної здатності.

n	9	9	9	29	9	49	99
p в %	5	10	25	25	40	40	40
$дс$ в %	0.00	0.09	4.89	0.18	26.66	7.76	2.19

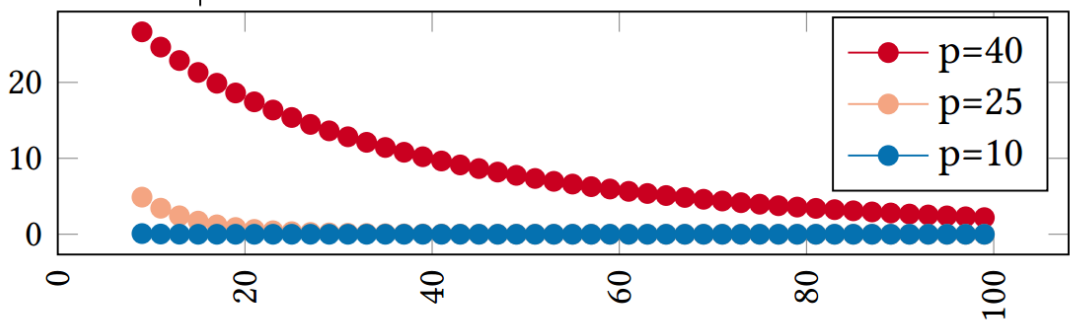


Рисунок 3.6 — Ймовірність *дс* (вісь y) щодо *p* і *n* (вісь x)

Компрометація групи малоймовірна, оскільки дс зазвичай низька для більшості зловмисників і ще більше зменшується, якщо підвищити n або кількість вимірювачів пропускної здатності. Крім того, складніше скомпрометувати групу, ніж скомпрометувати три управління пропускною здатністю. Можливий вплив зловмисника, який скомпрометував групу, залежить від того, чи контролює він хост вимірювача пропускної здатності чи також скомпрометував вимірювач пропускної здатності у довіреному середовищі виконання.

Тут зазначені сценарії атак, коли зловмисник може скомпрометувати хости вимірювача пропускної здатності, які він контролює, але не їхні вимірювачі пропускної здатності у довіреному середовищі виконання. Оскільки хости вимірювача пропускної здатності опосередковують зв'язок між вимірювачем пропускної здатності у довіреному середовищі виконання і смарт-контрактом, вони можуть затримувати або відкидати переслані пакети та транзакції. Хост вимірювача пропускної здатності може контролювати використання пропускної здатності системи та визначення вимірювань. Таким чином, зловмисник може

вирішити відкласти пересилання посилок зменшення вимірюваної пропускної здатності. Однак, оскільки вимірювач пропускної здатності у довіреному середовищі виконання вирішує порядок вимірювань і кожен вимірювач пропускної здатності у групі схожий, зловмисник не знає, який вимірювач пропускної здатності вимірюється зараз. Зловмиснику все одно може бути корисно затримувати пакети протягом усього процесу вимірювання, наприклад, якщо він це робить, то не контролює жодного з призначених релейних вузлів. Однак вимірювач пропускної здатності у довіреному середовищі виконання може виявити цю атаку шляхом спостереження розбіжності між очікуваною та виміряною пропускною здатністю для маніпуляторів релейних вузлів. В цьому випадку, це не створить результат транзакції.

Якщо використовувати параметри n = 9, r = 15, h = 2 і випадково затримувати вимірювання одного релейного вузла в кожному вимірюванні, зловмисник має лише 11,92% шанс бути захопленим вимірюванням маніпулятора. Проте це не впливає на консенсус, якщо він не зробив це принаймні на п’яти вимірювачах пропускної здатності у групі вимірювання. Навіть тоді він повинен відкласти той самий релейний вузол принаймні п’ять разів, щоб вплинути на консенсус. Вибираючи релейний вузол для випадкової затримки, це має невелику ймовірність 0,01%. Якщо йому не вдається вплинути на консенсус, його спроба атаки призводить до зниження репутаційної цінності всіх вимірювачів пропускної здатності, залучених до маніпуляції. Підвищення n, r, h або кількості вимірювачів пропускної здатності ще більше знижує ймовірність успіху такої атаки.

Зловмисник, який контролює хост вимірювача пропускної здатності, може не мати доступу до результатів транзакції τ_{res} або вирішити відмовитися від них замість того, щоб пересилати їх до смарт-контракту. У цьому випадку його вимірювач пропускної здатності не призначають бали репутації, що знижує його оцінку репутації. Це в кінцевому підсумку призводить до того, що смарт-контракт виключає його з групи зареєстрованих вимірювачів пропускної здатності. Оскільки зловмисник не скомпрометував вимірювач пропускної здатності у довіреному середовищі виконання у цьому сценарії, вимірювач пропускної здатності може знову приєднатися через певний проміжок часу. Однак це не скидає його рейтинг

репутації. Подібна поведінка призводить до подальшого зниження оцінки його репутації, поки смарт-контракт зрештою не виключає вимірювача пропускної здатності після кожного вимірювання. Враховуючи той факт, що зловмисник повинен платити за газ для повторного приєднання, дотримання цієї стратегії атаки може призвести до значних витрат.

Якщо зловмисник також може скомпрометувати поточну групу, він може запобігти створенню нових значень консенсусу, оскільки це робиться лише в тому випадку, якщо більшість вимірювачів пропускної здатності повідомили про результат. Однак, щоб запустити DoS-атаку та запобігти виміру релейних вузлів (знову ж таки), зловмиснику доведеться скомпрометувати кожну групу, що містить ці релейні вузли, що ще менше ймовірно відбуватиметься постійно.

Хоча компрометація безпечного обладнання коштує дорого і вимагає нетривіальних навичок, у мотивованого зловмисника може виникнути спокуса застосувати цей вектор атаки, якщо дохід від атаки переважає вкладені зусилля. Отже, ми детально розглянемо сценарії, коли зловмисник скомпрометує вимірювач пропускної здатності у довіреному середовищі виконання, і обговоримо наслідки для безпеки.

У першому сценарії зловмисник має під контролем кілька вимірювачів пропускної здатності у довіреному середовищі виконання, але він не скомпрометував групу. Такий вимірювач пропускної здатності може повідомляти про довільні вимірювання і, наприклад, призначати високу пропускну здатність контрольованим зловмисником релейного вузла і низьку пропускну здатність в іншому випадку. Однак ця атака виявляється під час процесу агрегації, оскільки смарт-контракт виключає значення викидів під час консенсусної агрегації. Крім того, такі спроби знижують рейтинг репутації контрольованого зловмисником вимірювача пропускної здатності, що призводить до виключення вимірювача пропускної здатності з групи зареєстрованих вимірювачів пропускної здатності, змушуючи його пройти процес повторного приєднання. Як і раніше, навіть якщо скомпрометовані ключі атестації довіреного середовища виконання будуть (ще не) відкликані, повторна реєстрація накладає на зловмисника додатковий фінансовий тягар. До тих пір, поки зловмисник не може скомпрометувати групу, смарт-

контракт ігнорує всі дані вимірювання скомпрометованих вимірювачів пропускної здатності і знижує їх репутацію.

У нашому другому сценарії зловмисник скомпрометував достатню кількість вимірювачів пропускної здатності у довіреному середовищі виконання, щоб скомпрометувати групу вимірювань. Для таких груп зловмисник може впливати на призначене значення пропускної здатності всіх релейних вузлів. Крім того, його репутаційні бали збільшуються, а репутаційні бали чесних вимірювачів пропускної здатності знижуються. Отже, існує вагома необхідність, щоб цей сценарій був настільки малоймовірним, наскільки це можливо. Однак ця атака набагато складніша, ніж скомпрометувати більшість (наразі: три) управлінь пропускною здатністю в поточній системі. Крім того, його вплив обмежений, оскільки некомпромісні групи все ще дають дійсні результати вимірювань і підвищують оцінку репутації чесних вимірювачів пропускної здатності. Нарешті, згідно з нашими як припущеннями, більшість груп ніколи не буде скомпрометованою, оскільки можна очікувати, що дс · кількість скомпрометованих груп і дс < p < 0,5. Тому ми вважаємо SmarTor значним покращенням у порівнянні з поточною системою.

Зловмисник може спробувати здійснити DoS-атаку на SmarTor, зареєструвавши неіснуючі реле-привиди. Оскільки перед першим вимірюванням значення пропускної здатності встановлено на 0, така атака не вплине на користувачів Tor. Крім того, зловмиснику доведеться заплатити газ за додавання кожного релейного вузла до блокчейну, що накладає величезні витрати. У той же час збільшення кількості даних релейних вузлів, що зберігаються в блокчейні, підвищить вартість виконання смарт-контракту, оскільки смарт-контракту потрібно створити більше випадкових чисел, поки не знайде достатньо «старих» релейних вузлів для розміщення в групі. Одним із рішень для запобігання такої атаки є збереження реле, останнє вимірювання яких не було успішним (або існуючим), в окремому списку та додавання їх до списку перевірених релейних вузлів після дійсних вимірювань. У цьому випадку до кожної групи додається постійна кількість релейних вузлів зі списку невиміряних, що гарантує, що кожен знову зареєстрований релейний вузол вимірюється в певний момент. Для

запобігання подібної атаки на систему реєстрації вимірювача пропускної здатності можна ввести депозит за реєстрацію, який повертається лише після успішної реєстрації.

Смарт-контракт лише додає релейні вузли з відповідним розміром bwsize до групи. Щоб обмежити кількість релейних вузлів, яку може вибрати смарт-контракт, вимірювач пропускної здатності може повідомити про необґрунтовано низький bwmax.

Введення порогу пропускної здатності для вимірювач пропускної здатності запобігає такій атаці.

3.5 Реалізація

З міркувань сумісності ключі зв’язку створюються за допомогою базової криптосистеми поточного розгорнутої системи.

Вона зберігає інформацію, пов’язану з вимірювачами пропускної здатності, групами вимірювань і релейними вузлами. Усі змінні, які не встановлені явно з параметром конкретної транзакції за замовчуванням на 0.

Смарт-контракт зберігає змінну стану для кожного вимірювача пропускної здатності.

Поточний стан вимірювача пропускної здатності обмежує дозволену взаємодію з смарт-контрактом, так що, наприклад, вимірювач пропускної здатності, який не зареєструвався, не може голосувати за нові вимірювачі пропускної здатності.

Стани та події переходу показані на рисунку 3.7.

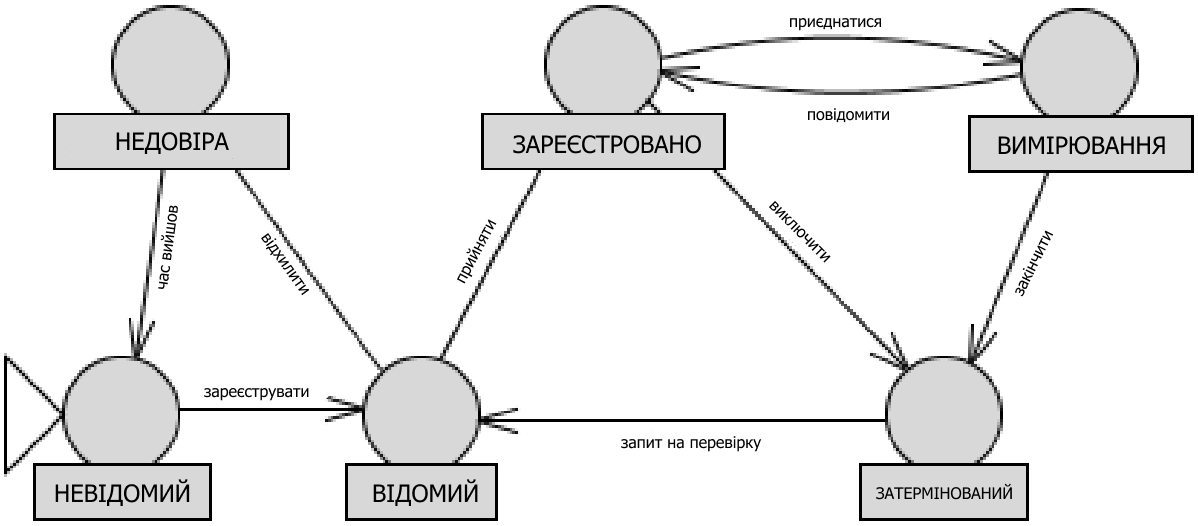


Рисунок 3.7 — Стани вимірювачів пропускної здатності та події переходу

На даний момент дані атестації SGX мають бути перевірені Службою атестації Intel. Хоча цей вибір дизайну гарантує, що перевірки та відкликання обробляються правильно, він також вимагає синхронного каналу зв'язку, недоступного всередині смарт-контракту. Як обхідний шлях, можна запропонувати, що дев'ять поточних управлінь довідкою взяти на себе роль початкової групи вимірювачів пропускної здатності, оскільки їм уже довіряють у поточній системі. Можна зауважити, що довіра до цієї початкової групи обмежена фазою завантаження, оскільки більше вимірювачів пропускної здатності приєднуються та перевіряють дані атестації.

Хоча мови розумних контрактів теоретично повні за Тьюрингом, існує кілька простих операцій, які на практиці вимагають складних обхідних шляхів. Це також вірно для Solidity, який, наскільки нам відомо, є найпотужнішою мовою смарт-контракту. Через ліміт газу для виконання смарт-контракту можливий обсяг обхідних шляхів обмежений, оскільки цей ліміт не можна підвищити довільно.

На практиці перед впровадженням повної версії SmarTor в Solidity слід ввести наступні функції: числа з фіксованою точкою та обчислення, ітерація над наборами ключів/значень карт, видалення карт, повернути номер блоку для хешу блоку, генератори псевдовипадкових чисел (PRNG). Через ці обмеження дана реалізація Proof-of-Concept забезпечує повну функціональність лише для однієї

групи вимірювань. Однак цього достатньо, щоб оцінити вартість газу такої системи. Смарт-контракти не мають рідного доступу до випадковості. Як створити випадковість у смарт-контракті – це ортогональна проблема, яка досліджується окремо і не є специфічною для даної програми. Можна використовувати псевдовипадковість (наприклад, із заголовків блоків), але цей механізм повинен бути загартований проти можливого впливу майнерів. У даній реалізації смарт-контракт вибирає релейний вузол для групи вимірювань, використовуючи реальну апаратну випадковість, надану вимірювачів пропускної здатності у довіреному середовищі виконання під час приєднання до групи. Смарт-контракт об'єднує його на початку групи, хешуючи об'єднані випадкові значення. Це хеш-значення потім заповнює PRNG.

Оскільки ця випадковість доступна лише після запуску групи, цей підхід не можна використовувати для призначення вимірювачів пропускної здатності групам, а інші підходи потрібно досліджувати, якщо розширити реалізацію на кілька груп.

Сценарій вимірювання пропускної здатності впроваджено як замкнута група Intel SGX. Реалізація написана на C++ з використанням Intel SGX SDK 1.9 і складається з 3018 LoC. Отримана бібліотека пропонує ECalls, щоб запустити функцію реєстрації (create_wallet), приєднання раундів вимірювань (join_group), звіт про результати вимірювань (measure_bandwidth) і закінчення (expire). Кожен ECcall призводить до необробленої транзакції, підписаної гаманцевим ключем, створеним під час create_wallet. Деякі криптографічні функції були переписані за допомогою бібліотеки mbedtls для SGX і Portable C++ Hashing Library. Ця підписана транзакція зберігається у файлі, звідки хост вимірювача пропускної здатності може переслати її до смарт-контракту. Наприклад, create_wallet призводить до транзакції реєстрації τBM, яка в даній реалізації також може використовуватися для запиту повторного приєднання. Якщо ECall зазнає невдачі (наприклад, під час спроби завершити термін дії перед створенням гаманця), замкнута група генерує виняток, закінчуючи без здійснення транзакції. Система була розроблена, використовуючи режим попереднього випуску, який, на відміну від режиму випуску, використовує непідписані замкнуті групи, і вони не можуть

створювати засвідчення. На практиці розповсюджувач програмного коду (тобто Tor Foundation) мав би підписати кожен опублікований анклав перед розгортанням ключем підпису анклаву, запитаним у Intel. Функція sgx_read_rand використовується для створення справді випадкових чисел (наприклад, для створення ключа), sgx_create_monotonic_counter для лічильників транзакцій і sgx_get_trusted_time як джерело часу. Хоча роздільна здатність цього джерела часу обмежена секундами, все ще можна досягти точних результатів вимірювань.

Було використано звичайні клієнти Tor і Geth для Windows для підтримки зв'язку з мережами Tor і Ethereum відповідно. Дана реалізація хоста складається з 1112 LoC, який відповідає за пересилання транзакцій до смарт-контракту, запуск замкнутих груп та виконання запитуваних викликів. Виклики, необхідні для роботи в мережі, реалізовані за допомогою системної бібліотеки winsock для Windows. Додатковий сценарій перевіряє, чи зберегла замкнута група будь-яку нову транзакцію в папці. Якщо так, він пересилає її в мережу Ethereum за допомогою функції sendRawTransaction EthereumJS.

Крім того, клієнт Tor повинен запропонувати свій проксі-сервер Socks 5, який використовується для пересилання повідомлень на інше реле, на порт 9050, і порт керування на 9051. Вимірювач пропускної здатності у довіреному середовищі виконання використовує керуючий порт для створення схем і виконання реконфігурації, необхідної для процесу вимірювання. Зокрема, схеми створюються не автоматично, а за допомогою керуючих команд з замкнутої групи. Можна зауважити, що зазвичай зв'язок з Tor і ключі Tor також мають бути захищені Intel SGX. Цього можна досягти, використовуючи безпечну реалізацію Tor, як-от SGX-Tor.

3.6 Висновок

У цьому розділі було представлено SmarTor, підхід до підвищення стійкості мережі анонімності Tor проти атак на основні компоненти інфраструктури та маніпулювання інформацією про топологію. SmarTor переміщує довіру з управління довідкою та пропускною здатністю, якими керує невелика кількість осіб, до блокчейну, який набагато важче скомпрометувати, оскільки його безпека забезпечена за рахунок обчислювальної потужності мільйонів майнерів по всьому світу. Крім того, SmarTor покладається на сучасні безпечні апаратні технології, вимірювання маніпулятором та систему репутації, щоб захистити процес вимірювання пропускної здатності від маніпуляцій. Було надано підтверджену концепцію реалізацію та показано, що дане рішення дає більш надійні та достовірні вимірювання пропускної здатності, ніж поточна система.

ВИСНОВКИ

У даній роботі було представлено розробку розподіленої комп’ютерної системи спеціалізованого призначення згідно топології “тор”.

У ході роботи було зазначено важливість проектування даної системи та описано головні типи архітектур. Внаслідок порівняння декількох системних архітектур, було взято за основу архітектуру розподілених об’єктів тому, що вона володіє великою кількістю переваг перед іншими видами архітектур. Крім того, було описано вже існуючі рішення та опрацьовано головні проблеми таких систем, тому було сформовано вимоги до розробки системи.

Тор представляє собою топологію, що володіє найвищими показниками швидкості та продуктивності. Також, основною перевагою цієї топології є можливість підтримки великої кількості вузлів та легкого масштабування.

Спочатку було побудовано модель системи SmarTor, що містила релейні вузли, приховані в Tor веб-сервіси і клієнти Tor. Крім того, було враховано можливість атаки на дану систему, а щоб запобігти цьому, було розглянуто можливі шляхи нанесення атак, при яких система може постраждати.

Також було оглянуто систему високого рівня, щоб продемонструвати взаємодію між залученими сутностями. Використовуючи функцію виключення, було продемонстровано ймовірність виключення вимірювача пропускної здатності після кількох раундів вимірювання.

Сама ж розподілена система працює доволі коректно та її можна використовувати у подальшому виконанні різноманітних завдань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1) Bernaschi, Massimo, et al. Spiders like onions: on the network of tor hidden services. In: *The World Wide Web Conference*. 2019. p. 105-115.

2) Топології багатопроцесорних систем. URL: https://cutt.ly/PFnkQos (дата звернення: 15.02.2022)

3) Kelley, John L. *General topology*. Courier Dover Publications, 2017.

4) Sierpinski, Waclaw. General topology. In: *General Topology*. University of Toronto press, 2020.

5) Edelsbrunner, Herbert; harer, John L. *Computational topology: an introduction*. American Mathematical Society, 2022.

6) Torus interconnect. URL: https://en.wikipedia.org/wiki/Torus_interconnect (дата звернення: 15.02.2022).

7) Nakahara, Mikio. *Geometry, topology and physics*. CRC press, 2018.

8) Torus. URL: https://en.wikipedia.org/wiki/Torus (дата звернення 15.02.2022)

9) Eilenberg, Samuel; steenrod, Norman. Foundations of algebraic topology. In: *Foundations of Algebraic Topology*. Princeton University Press, 2015.

10) Munkres, James R. *Elements of algebraic topology*. CRC press, 2018.

11) Scullin, William; scovel, Adam. Lessons from the ibm blue gene series of supercomputers. In: *Proceedings of the HPC Systems Professionals Workshop*. 2017. p. 1-7.

12) Yoshida, Toshio. Fujitsu high performance CPU for the Post-K Computer. In: *Hot Chips*. 2018. p. 1-22.

13) Varshney, Mukul. Problems in Task Scheduling in Multiprocessor System.

14) Marty, Michael, et al. Snap: A microkernel approach to host networking. In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 2019. p. 399-413.

15) Kammerer, Klaus, et al. Ambalytics: A scalable and distributed system architecture concept for bibliometric network analyses. *Future Internet*, 2021, 13.8: 203.

16) Огляд і архітектура обчислювальних мереж. URL: https://naurok.com.ua/rozrobka-uroku-na-temu-oglyad-i-arhitektura-obchislyuvalnih-merezh-144464.html (дата звернення 10.03.2022)

17) Ahad, Mohd Abdul; Biswas, Ranjit. PPS-ADS: a framework for privacy-preserved and secured distributed system architecture for handling big data. *International Journal on Advanced Science, Engineering and Information Technology*, 2018, 8.4: 1333-1342.

18) Jafari Navimipour, Nima; Sharifi Milani, Farnaz. A comprehensive study of the resource discovery techniques in peer-to-peer networks. *Peer-to-Peer networking and applications*, 2015, 8.3: 474-492.

19) Yu, Quan, et al. Cybertwin: An origin of next generation network architecture. *IEEE Wireless Communications*, 2019, 26.6: 111-117.

20) Sallow, Amira B., et al. Client/Server remote control administration system: Design and implementation. *Int. J. Multidiscip. Res. Publ*, 2020, 3.2: 7.

21) Багатопроцесорна архітектура. URL: https://studfile.net/preview/5203877/page:6/ (дата звернення 18.02.2022)

22) Dhieb, Najmeddine, et al. Scalable and secure architecture for distributed iot systems. In: *2020 IEEE Technology & Engineering Management Conference (TEMSCON)*. IEEE, 2020. p. 1-6.

23) Roehrs, Alex; Da Costa, Cristiano André; Da Rosa Righi, Rodrigo. OmniPHR: A distributed architecture model to integrate personal health records. *Journal of biomedical informatics*, 2017, 71: 70-81.

24) Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2018. SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution. arXiv preprint arXiv:1802.09085v1.

25) Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. IACR Cryptology ePrint Archive 2016, 86.

26) Maxence Delong, Eric Filiol, Clément Coddet, Olivier Fatou, and Clément Suhard. 2018. Technical and OSINT Analysis of the TOR Foundation. In ICCWS 2018 13th International Conference on Cyber Warfare and Security. Academic Conferences and publishing limited, 164.

27) Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In European Workshop on Systems Security (EuroSec).

28) Aaron Johnson, Rob Jansen, Nicholas Hopper, Aaron Segal, and Paul Syverson. 2017. PeerFlow: Secure Load Balancing in Tor. Proceedings on Privacy Enhancing Technologies, 2017, 74–94.

29) Simon Johnson, Vincent Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. 2016. Intel software guard extensions: EPID provisioning and attestation services. ser. Intel Corporation.

30) Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In 26th USENIX Security Symposium (USENIX Security).

31) Peter Mell, John Kelsey, and James Shook. 2017. Cryptocurrency Smart Contracts for Distributed Consensus of Public Randomness. In International Symposium on Stabilization, Safety, and Security of Distributed Systems. Springer, 410–425.

32) Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA).

33) F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. 2015. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In IEEE Symposium on Security and Privacy.

34) Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town crier: An authenticated data feed for smart contracts. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 270–282.

35) Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper 151 (2014).

36) Jesse Victors, Ming Li, and Xinwen Fu. 2017. The Onion Name System. Proceedings on Privacy Enhancing Technologies 2017, 1 (2017), 21–41.

37) Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In Proceedings of the 27th USENIX Security Symposium. USENIX Association.

38) Chia-Che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A practical library OS for unmodified applications on SGX. In Proceedings of the 2017 USENIX Annual Technical Conference.

39) Yogesh Swami. 2017. Intel SGX Remote Attestation is not sufficient. (2017).

40) Chris Mills. 2016. Judge Confirms Carnegie Mellon Hacked Tor and Provided Info to FBI. http://gizmodo.com/judge-confirms-carnegie-mellon-hacked-tor-and-provided-1761191933.

41) Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. CacheZoom: How SGX Amplifies The Power of Cache Attacks. Technical Report. arXiv:1703.06986 [cs.CR]. https://arxiv.org/abs/1703.06986.

42) Paschos, Georgios S., et al. The role of caching in future communication systems and networks. *IEEE Journal on Selected Areas in Communications*, 2018, 36.6: 1111-1125.

43) Kehagias, Dionysios, et al. Investigating the interaction between energy consumption, quality of service, reliability, security, and maintainability of computer systems and networks. *SN Computer Science*, 2021, 2.1: 1-6.

44) Ahmad, Tanveer; Zhang, Dongdong. Using the internet of things in smart energy systems and networks. *Sustainable Cities and Society*, 2021, 68: 102783.

45) Sharples, Mike; Domingue, John. The blockchain and kudos: A distributed system for educational record, reputation and reward. In: *European conference on technology enhanced learning*. Springer, Cham, 2016. p. 490-496.

46) Zoss, Brandon M., et al. Distributed system of autonomous buoys for scalable deployment and monitoring of large waterbodies. *Autonomous Robots*, 2018, 42.8: 1669-1689.

47) JIANG, Wenbo, et al. PTAS: Privacy-preserving thin-client authentication scheme in blockchain-based PKI. *Future Generation Computer Systems*, 2019, 96: 185-195.

48) ZHANG, Yin Sheng. A Fat Client OS Architecture Supported by Semi-network Resources. *International Journal of Computing Sciences Research*, 2022, 6: 857-876.

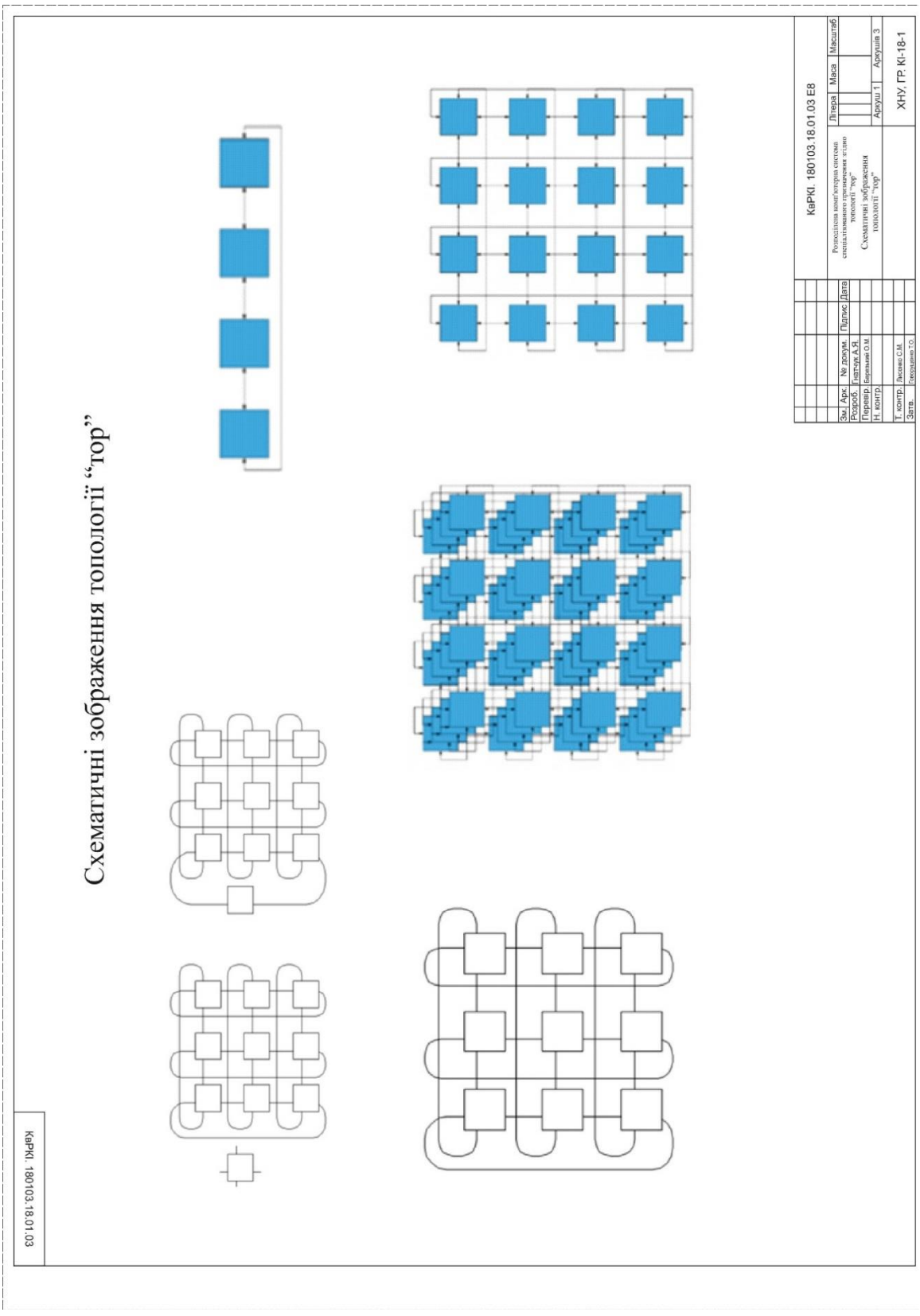
49) Tan, Boon Seng; Low, Kin Yew. Blockchain as the database engine in the accounting system. *Australian Accounting Review*, 2019, 29.2: 312-318.

50) García-Valdez, José-Mario; Merelo-Guervós, Juan-Julián. A modern, event-based architecture for distributed evolutionary algorithms. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2018. p. 233-234.

Додаток А

(обов’язковий)

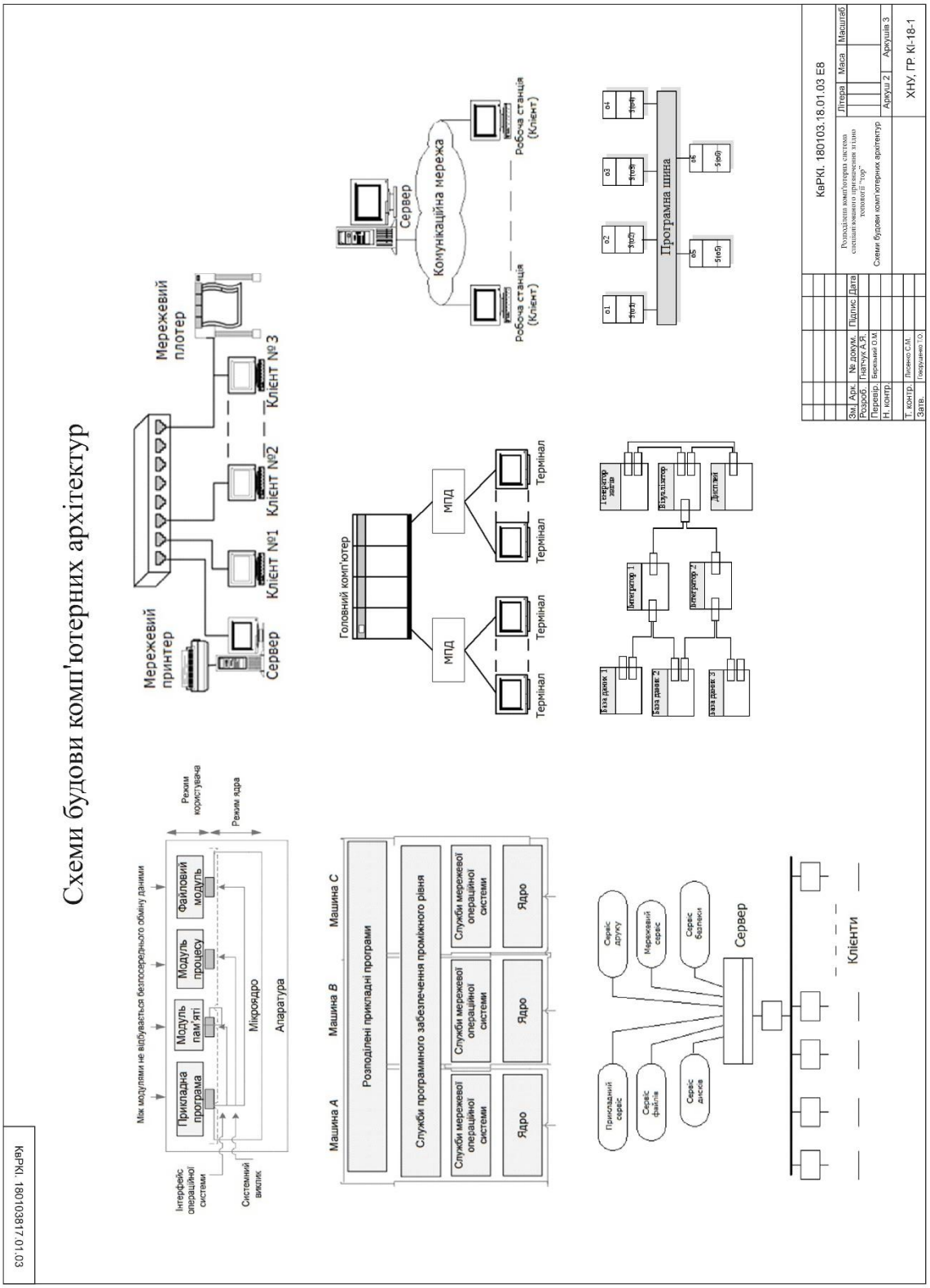
Копія «Схематичні зображення топології “тор”»



Додаток Б

(обов’язковий)

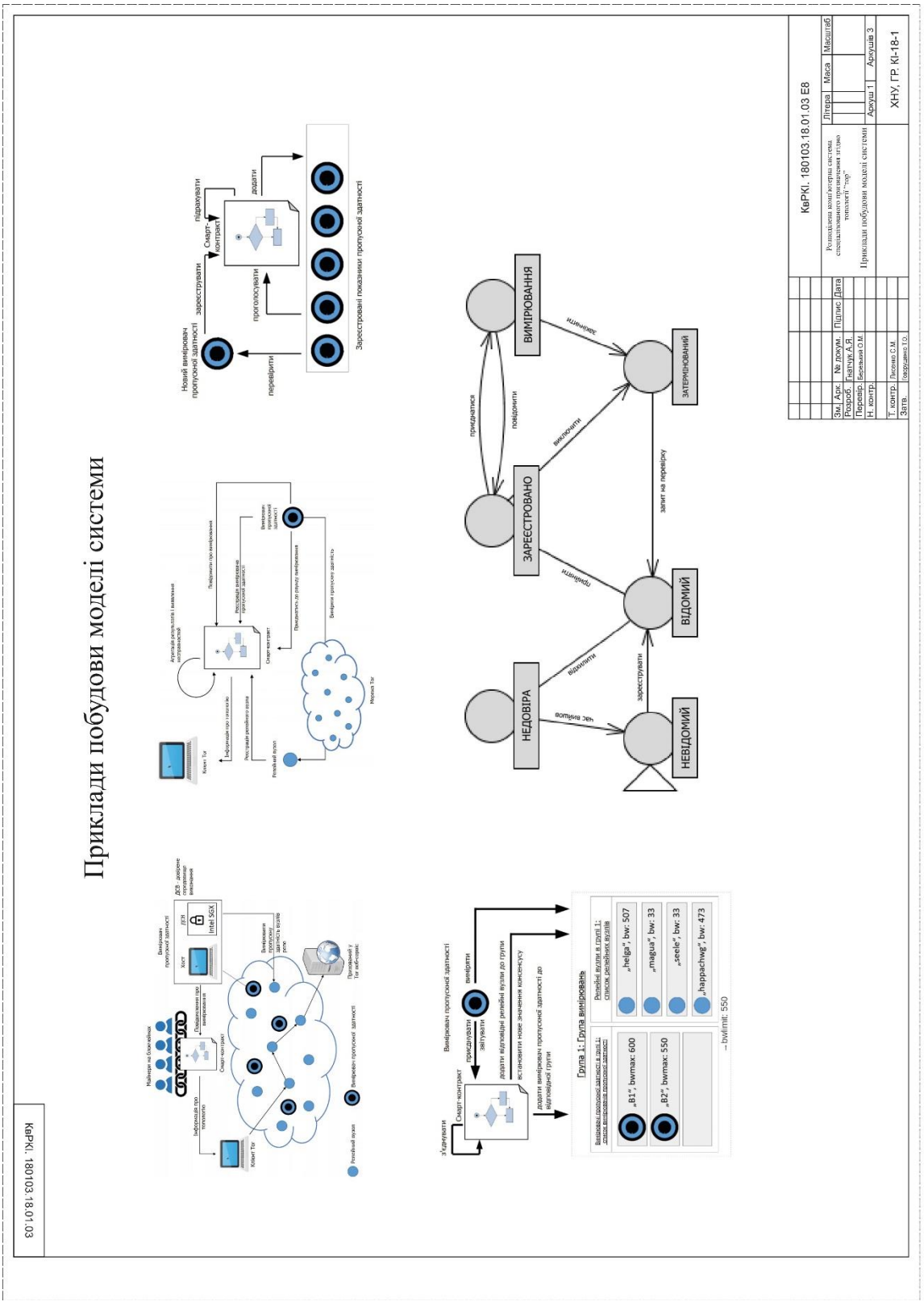
Копія «Схеми будови комп’ютерних архітектур»



Додаток В

(обов’язковий)

Копія «Приклади побудови моделі системи»



UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1011478457

Дата перевірки:
06.06.2022 22:03:12 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
06.06.2022 22:03:30 EEST

ID користувача:
100005591

Назва документа: **Гнатчук_Розподілена комп'ютерна система спеціалізованого призначення згідно топології...**

Кількість сторінок: **69** Кількість слів: **11555** Кількість символів: **91477** Розмір файлу: **1.05 MB** ID файлу: **1011356052**

0.83%
Схожість

Найбільша схожість: **0.6% з джерелом з Бібліотеки** (ID файлу: **1011242507**)

0.1% Джерела з Інтернету	1	Сторінка 71
0.73% Джерела з Бібліотеки	75	Сторінка 71

0% Цитат

Не знайдено жодних цитат

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	55

Mon Jun 06 21:15:07 EEST 2022, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 0.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 8%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 104605 Название: Розподілена комп'ютерна система спеціалізованого призначення згідно топології "тор" Добавлено в БД: 2022-06-06 Авторы: А.Я. Гнатчук Руководители: О.М. Березький Консультанты: Оппоненты:	Символы	Лексемы	Символы	Лексемы
	84633	666	340 (0%)	5 (1%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Розподілена комп'ютерна система спеціалізованого призначення згідно топології "тор"

Автор: Гнатчук Аліна Ярославівна

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Березький Олег Миколайович, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення мають належним чином оформленні посилання;
2) окремі виявлені збіги є загальновживаними фразами або виразами;
3) всі зафіксовані системою модифікації тексту є комбінацією латинських символів зі українськими, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 0.83% і адресується до 76 першоджерел, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи О. М. Березький

Гарант ОП С. М. Лисенко

Завідувач кафедри КІСП Т. О. Говорущенко

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Гнатчук Аліна Ярославівна

Тема: Розподілена комп'ютерна система спеціалізованого призначення згідно топології "тор"

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 58

1. Короткий зміст роботи та прийнятих рішень: Розробка розподіленої комп'ютерної системи спеціалізованого призначенні згідно топології "тор".

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У цій роботі було побудовано модель системи SmarTor, що містить релейні вузли, приховані в Tor веб-сервіси і клієнти Tor, також було враховано можливість атаки на дану систему, а щоб запобігти цьому, було розглянуто можливі шляхи нанесення атак, при яких система може постраждати. Також було оглянуто систему високого рівня, щоб продемонструвати взаємодію між залученими сутностями. Використовуючи функцію виключення, було продемонстровано ймовірність виключення вимірювача пропускної здатності після кількох раундів вимірювання.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: - .

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на високому інженерно-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно (5.00/A)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Яшина О.М. к.ф.-м.н., доцент кафедри інженерії програмного забезпечення

"8" 06 2022 р. _____ (підпис)

Завідувачу кафедри КІІС
д-ру техн.наук, проф. Говорущенко Т. О.

Гнатчук А.Я.
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ-18-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність плагіату ознайомлений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

08 червня 2022 — дата

підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ

Направляється студент Гнатчук Аліна Ярославівна на захист дипломного проєкту (роботи)
(прізвище, ім'я, по батькові)

за спеціальністю 123 - Комп'ютерна інженерія

На тему: Розподілена комп'ютерна система спеціалізованого призначення згідно топології "тор"

Дипломний проєкт (робота), рецензія і довідка про перевірку на плагіат додаються.

Декан факультету _____ О. Сосненко
(підпис) (ім'я, прізвище)

ДОВІДКА УСПІШНОСТІ

Гнатчук А. Я. за період навчання на факультеті інформаційних технологій з 2018 по 2022 роки повністю виконав навчальний план спеціальності з таким розподілом оцінок за:
національною шкалою: відмінно 100,00 %, добре 0,00 %, задовільно 0,00 %.
шкалою ЄКТС: A 100,00, B 0,00 %, C 0,00 %, D 0,00 %, E 0,00 %.

Методист факультету _____ Т. Вагнер
(підпис) (ім'я, прізвище)

ВИСНОВОК КЕРІВНИКА ДИПЛОМНОГО ПРОЄКТУ (РОБОТИ) ТА ОБГРУНТУВАННЯ ОЦІНКИ

Студент Гнатчук Аліна Ярославівна виконала всі поставлені завдання на високому інтелектуально-технічному рівні під час виконання кваліфікаційної роботи та заслуговує оцінки відмінно

Оцінка дипломного проєкту (роботи) відмінно А (5,00)

Керівник дипломного проєкту _____ Березький О. М.
(підпис) (ім'я, прізвище)

"____" _____ 2022 р.

ВИСНОВОК КАФЕДРИ ПРО ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Дипломний проєкт (роботу) розглянуто. Студент Гнатчук А. Я. допускається до захисту цього проєкту (роботи) в екзаменаційній комісії.

Завідувач кафедри Кіїс
(назва)

Т. Говорущенко "08" 06 2022 р.
(підпис, ім'я, прізвище)