ЗМІСТ

[ВСТУП 4](#_Toc467434732)

[1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ 6](#_Toc467434733)

[2 ФІЗИЧНА ТОПОЛОГІЯ 8](#_Toc467434734)

[3 ЛОГІЧНА ТОПОЛОГІЯ 10](#_Toc467434735)

[4 МУЛЬТИСЕРВІСНА ТЕХНОЛОГІЯ 12](#_Toc467434736)

[4.1 Протокол HTTP 12](#_Toc467434737)

[4.2 Протокол HTTPS 18](#_Toc467434738)

[4.3 Протокол WebSocket 19](#_Toc467434739)

[ВИСНОВКИ 25](#_Toc467434740)

[ПЕРЕЛІК ПОСИЛАНЬ 26](#_Toc467434741)

# ВСТУП

Сучасні технології передачі даних надають своїм користувачам широкі можливості по організації різних видів послуг і сервісів:

* організація електронного документообігу й ведення загальних архівів документів;
* організація корпоративної телефонної мережі з єдиним планом нумерації;
* організація систем конференц-зв’язку, у тому числі відеоконференцій;
* побудова розподілених систем відеоспостереження з єдиним центром зберігання даних;
* організація дистанційного доступу до файлів і серверів з базами даних;
* підключення до мережі Інтернет з можливістю організації єдиної корпоративної політики інформаційної безпеки і т. ін.

Перераховані можливості дозволяють спростити процедури по керуванню внутрішніми бізнес-процесами й надають компанії ряд конкурентних переваг, а саме забезпечуються: простота керування компанією, прозорість роботи компанії, оперативний контроль діяльності всіх служб і структурних підрозділів, доступ до всіх інформаційних ресурсів підприємства в реальному часі, оперативний зв’язок; економія засобів на міжнародних й міжміських дзвінках.

Більшість інформаційних рішень, що застосовуються сьогодні, носить розподілений характер і вимагає наявності на підприємстві або в організації високопродуктивної корпоративної мережі передачі даних.

Основним завданням корпоративної мережі є забезпечення взаємодії системних додатків та організація доступу до них віддалених користувачів.

Таким чином, корпоративна мережа – це складна система, що включає тисячі найрізноманітніших компонентів: комп’ютери різних типів, системне й прикладне програмне забезпечення, мережеві адаптери, концентратори, комутатори й маршрутизатори, кабельна система. Основне завдання полягає в тому, щоб ця громіздка й досить дорога система якнайкраще справлялася з обробкою потоків інформації, що циркулюють між співробітниками підприємства й дозволяла приймати їм своєчасні й раціональні рішення.

Основною метою даної курсової роботи являється проектування комп’ютерної мережі із підтримкою протоколу WebSocket.

# 1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

Локальна мережа будується на основі логічної топології „зірка”. Вибір даної топології серед всіх інших пояснюється більшою стійкістю всієї мережі до неполадок окремої станції, та швидкістю виявлення та усунення несправностей.Локальна мережа є мережею типу клієнт-сервер. Даний вибір пояснюється більш широкими можливостями, які надає мережа вказаного типу. Це перш за все: простота адміністрування великих мереж, підвищений рівень безпеки всієї мережі, можливість спільного використання мережевих ресурсів.

Призначення розробки:

На даний момент у закладі склалася ситуація, коли :

1. Є певна кількість комп’ютерів, які працюють окремо і не мають змоги обмінюватися інформацією з іншими комп’ютерами.
2. Відсутність можливості поділу комп’ютерів у відповідності з приналежністю до конкретної групи (відділу).
3. Неможливість спільного використання ресурсів (спільного для всіх мережевого принтера і т.д.).

Для вирішення даної проблеми потрібно створити єдину інформаційну

мережу організації. Дана мережа повинна виконувати наступні функції:

1. Створення єдиного інформаційного простору, який буде об’єднувати всі робочі станції мережі.
2. Забезпечення можливості обміну даними засобами мережі.
3. Надання можливості об’єднання робочих станцій у групи, які будуть характеризувати їх у відповідності з виконуваними задачами.
4. Забезпечення можливості спільного використання ресурсів (мережевого принтера і т.д.).

Реалізація проекту по проектуванню локальної мережі дозволить скоротити документообіг, підвищити продуктивність праці, скоротити час на обробку інформації. Процес створення локальної обчислювальної мережі повинен містити в собі не тільки рішення загальних задач побудови й організації мережі на організації, але і комплексно забезпечувати рішення проблем, які виникають з появою мережі в організації і її експлуатації. До таких проблем можна віднести доступ до інформації і її захист, забезпечення взаємодії робочих станцій між собою. При проектуванні мережі враховується також можливість її розширення в майбутньому без значних витрат. Це досягається шляхом прокладання додаткових кабельних сегментів та розеток.

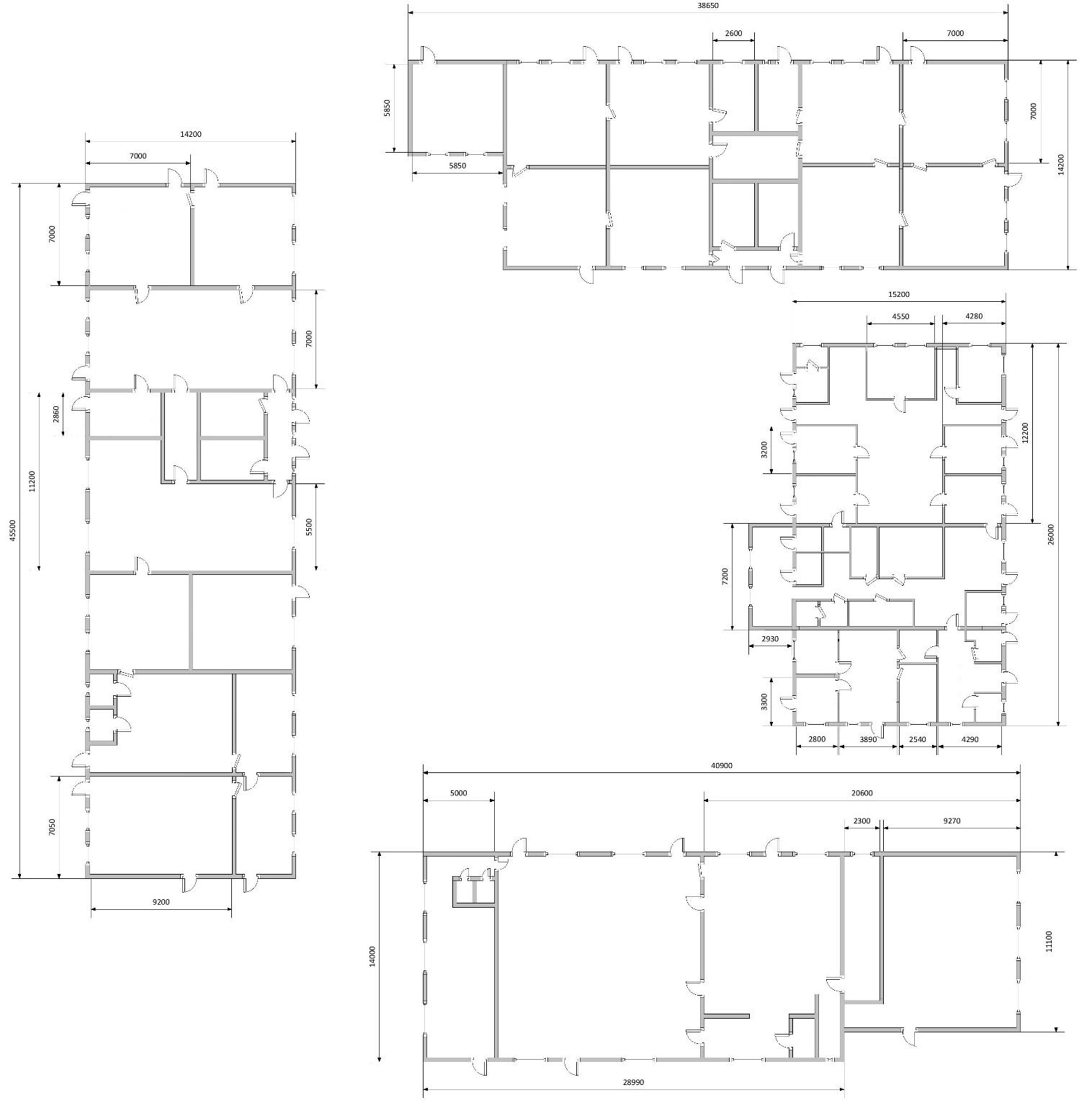


Рисунок 1.1 – План будівлі, для якої проводиться розробка локальної комп’ютерної мережі

# 2 ФІЗИЧНА ТОПОЛОГІЯ

Для побудови мережі було обрано топологію розширена зірка, в якій кожен вузол, що виходить з центрального є центром іншої зірки. Вибрана топологія є найбільш оптимальною та поширеною для побудови великих комп’ютерних мереж.

Структура комп’ютерної мережі

Основними логічними частинами мережі є:

- мережа для комп’ютерів, які знаходяться в 1 корпусі (кафедра А);

- мережа для комп’ютерів, які знаходяться в 2 корпусі (кафедра Б);

- мережа для комп’ютерів, які знаходяться в 3 корпусі (електронно-навчальні ресурси);

- мережа для комп’ютерів, які знаходяться в 4 корпусі (адміністративний підрозділ);

- мережа для серверів публічного доступу;

- мережа для серверів загального доступу;

Також всі ці 6 основних компонентів локальної мережі повинні мати доступ до мережі інтернет, який буде захищений міжмережевим екраном для захисту від атак ззовні мережі.

На рисунку 2.1 зображено загальну схему мережі. Дана мережа повністю відповідає поставленій задачі.



Рисунок 2.1 – Схема комп’ютерної мережі

На рисунку 2.2 представлено фізичну топологію комп’ютерної мережі.

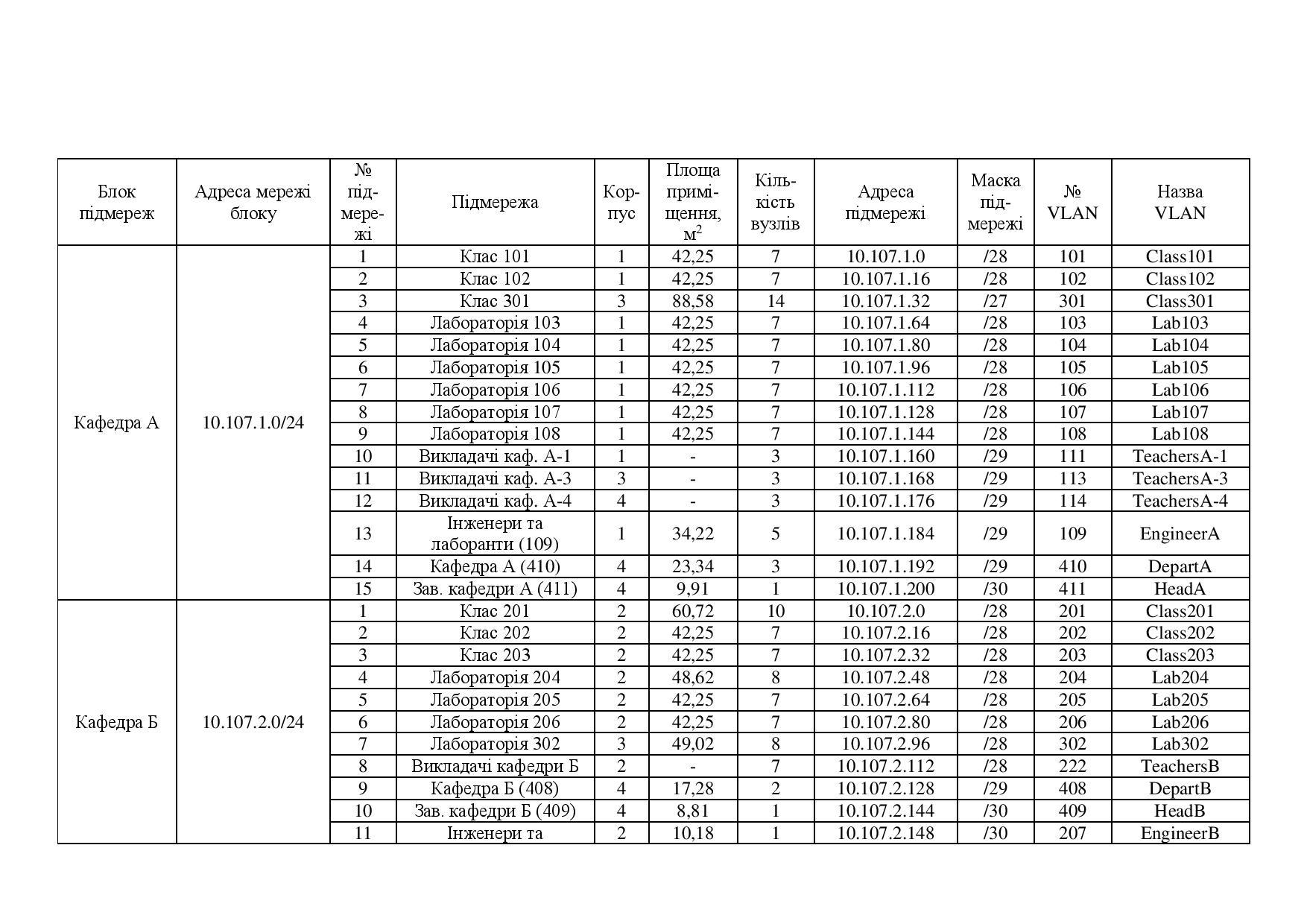


Рисунок 2.2 – Фізична топологія мережі

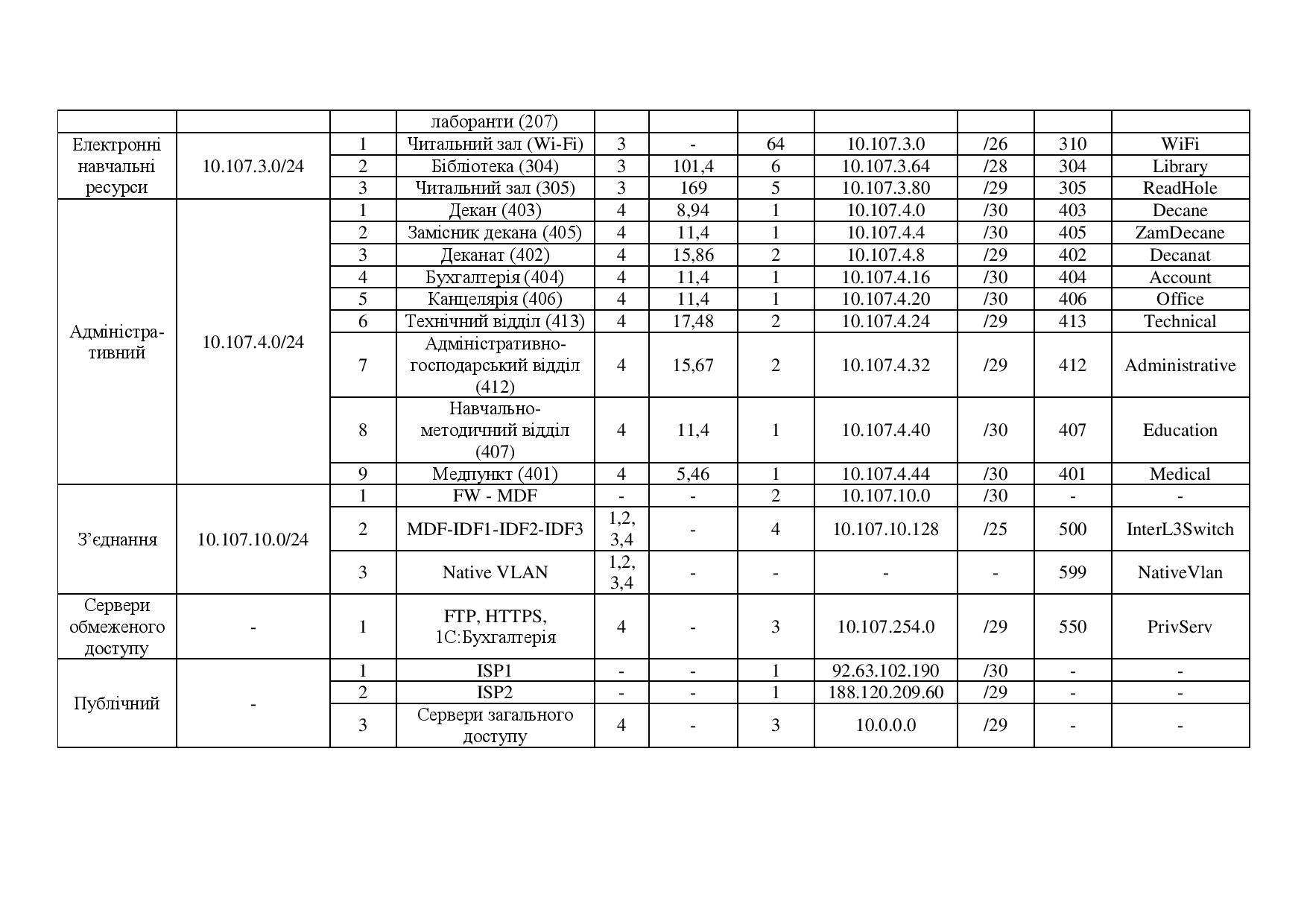
# 3 ЛОГІЧНА ТОПОЛОГІЯ

Відповідно до функціональності підрозділів факультету, розподілу на кафедри, наявності в кафедрах ієрархічної організації персоналу, а також з цілями безпеки, логічну мережу 10.107.0.0/16 сегментовано на функціональні блоки під мереж мережу 10.107.1.0/18, 10.107.2.0/18, 10.107.3.0/18, 10.107.1.0/18 відповідно до корпусів факультету. Ці блоки включать в себе ряд під мереж. Також сегментовано і додаткові та службові підмережі. В таблиці 1 представлено логічну сегментацію мережі.

Таблиця 3.1 – Логічна сегментація мережі



Продовження таблиці 3.1



На рисунку 3.1 представлено логічну топологію мережі.

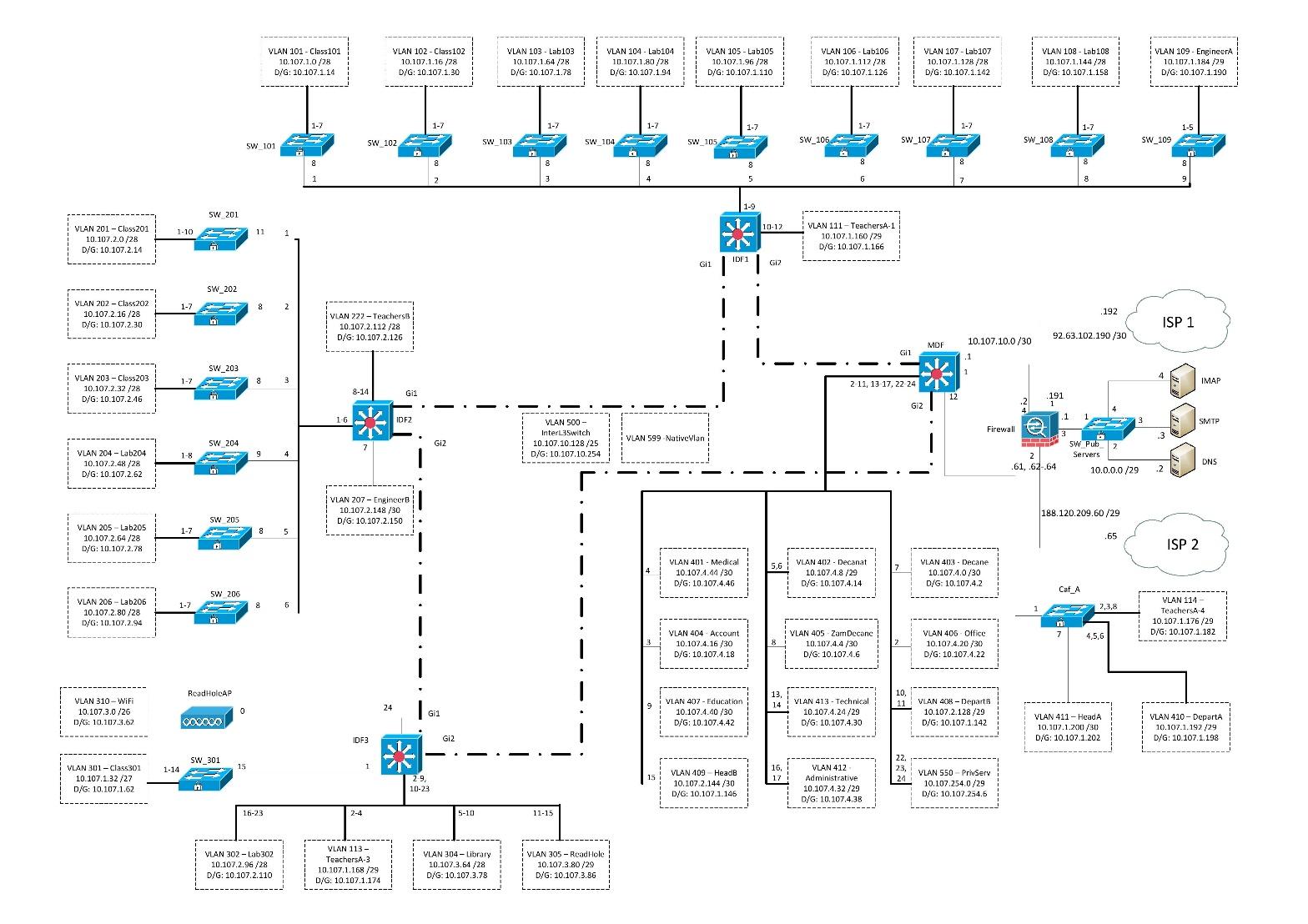


Рисунок 3.1 – Логічна топологія мережі

# 4 МУЛЬТИСЕРВІСНА ТЕХНОЛОГІЯ

4.1 Протокол HTTP

Як мультисервісну технологію було обрано сервери HTTP, HTTPS та WebSocket.

Протокол HTTP (HyperText Transfer Protocol - протокол передачі гіпертексту) - це протокол рівня додатки, що здійснює зв'язок додатків в межах розподілених, спільних або різнорідних інформаційних систем. Протокол дозволяє додаткам обмінюватися даними, представленими в зрозумілому для сприйняття людиною вигляді.

Як випливає з його назви, спочатку HTTP призначався для передачі між додатками так званого гіпертексту (hypertext), що представляє собою особливий вид даних, створений у відповідності зі стандартом HTML (HyperText Markup Language - мова розмітки гіпертексту). Гіпертекстовий документ складається з даних, розмічених за допомогою тегів (tag) мови HTML, і являє собою комбінацію тексту, зображень, гіперпосилань та інших засобів подання даних. Гіперпосилання - одна з найважливіших складових HTML-документа - представляють собою інтерактивні області, вплив на які призводить до отримання пов'язаних з гіперпосиланням даних. Це дозволяє користувачеві, який працює з гіпертекстової інформацією, здійснювати навігацію в межах набору документів чи навіть всієї мережі Інтернет, отримуючи потрібну йому інформацію за допомогою контекстних гіперпосилань.

Протокол HTTP є надбудовою над протоколом TCP і є засобом контролю вмісту переданих даних. На відміну від TCP, який не враховував структуру переданих пакетів, HTTP впроваджує в дані метаінформацію, що дозволяє одержувачу коректно інтерпретувати отримані дані. На основі HTTP функціонує глобальна мережа Інтернет (звана також World Wide Web або WWW). Перша версія протоколу - HTTP/0.9 - володіла достатньо обмеженими можливостями, але з активним розвитком всесвітньої мережі з'явилися нові версії: HTTP/1.0 та HTTP/1.1, що дозволяють контролювати передачу з обчислювальних мереж не тільки гіпертекстової інформації, а й довільні бінарні файли: звукові, графічні, архівні та ін

У силу того, що HTTP є надбудовою над протоколом TCP, при передачі даних також виділяються дві сторони: клієнт і сервер.

Клієнт є ініціатором з'єднання і запитує у сервера деякі дані або послуги. Клієнтом, як правило, є програма, яка називається браузером (browser), що дозволяє як відображати гіпертекстове інформацію, так і приймати файли інших форматів. Щоб отримати деяку інформацію, клієнт посилає серверу запит (request), що містить опис запитуваної інформації.

Сервер при передачі даних через HTTP називають веб-сервером (web-server). Ця програма здійснює обробку запитів від клієнтів, передаючи запитані дані у вигляді відповідей (response), що містять крім шуканих даних метаінформацію, їх описує.

Отримання користувачем цікавлячих його даних складається з наступних етапів:

1. Користувач вводить в рядку браузера адресу цікавить його ресурсу.

2. Браузер на основі інформації, отриманої від користувача, а також з огляду на свої налаштування і конфігурацію операційної системи, формує запит.

3. Браузер підключається до сервера, розташованому, можливо, на віддаленому комп'ютері, і передає йому запит.

4. Сервер, аналізуючи запит, виконує необхідні дії: формує відповідь, включаючи в нього тіло запитаного документа. Якщо це гіпертекстовий документ, він завантажується з файлу або ж генерується динамічно за допомогою сценарію. У відповідь також включається інформація про його даних.

5. Сервер передає відповідь браузеру.

6. Браузер аналізує відповідь і або зберігає отримані дані у файл, або, у разі гіпертекстового документа, аналізує теги HTML і відображає документ на екрані.

Слід зауважити, що клієнтською програмою може бути не тільки браузер, тим не менш, всі кроки, за винятком, може бути, першого, виконуються в будь-якому випадку.

Слід зауважити, що тут розглядається безпосереднє підключення клієнта до сервера, який містить цікаву інформацію, однак, це не завжди можливо в силу різних обставин. У такому випадку підключення може здійснюватися за допомогою однієї чи більше проміжних точок підключення. Можна розділити ці проміжні точки на три групи:

* Проксі-сервери (proxy-server) - програма-посередник, що виконує функції як клієнта, так і сервера з метою створення запитів від імені інших клієнтів. Запити обслуговуються проксі-сервером, або пересилаються їм з внесенням до них змін (у цьому випадку проксі-сервер має назву непрозорим) або без змін (у цьому випадку проксі-сервер має назву прозорим). Проксі-сервер дозволяє групі комп'ютерів виступати в якості одного клієнта або сервера, що часто застосовується при підключенні до Інтернету локальних мереж.
* Шлюз (gateway) - як і проксі-сервер, здійснює трансляцію запитів, однак, не ставлю їх зміни. Шлюз отримує від клієнта запит, як до сервера, який містить шуканий ресурс. Таким чином, клієнт не може визначити, підключається він через шлюз або безпосередньо до містить ресурс сервера.
* Тунель (tunnel) - програма-посередник, що підтримує з'єднання. Хоча після установки з'єднання тунель не розглядається в якості елемента передачі через протокол HTTP, з'єднання, як правило, ініціюється саме HTTP-запитом. Тунель перериває свою роботу, якщо хоча б один з учасників обміну даними закриває з'єднання.

Для збереження функціональності передачі даних при підключенні через проміжні точки не потрібно внесення змін в запити і відповіді, за винятком випадку проксі-сервера: в цьому випадку в клієнтському запиті повинно міститися додаткові поля. Однак, з точки зору сервера, обмін даними здійснюється безпосередньо з клієнтом, отже, ніяких змін в запитах не відбувається. Поетом при розробці програми можливість підключення через проміжні точки не враховувалася.

Запит, що відправляється клієнтом серверу, служить для точної ідентифікації запитуваного ресурсу, а також містить відомості, необхідні для коректної обробки запиту.

За своєю структурою запит складається з трьох частин:

* Рядок запиту
* Блок заголовків
* Об'єкт

Рядок запиту складається з трьох полів, розділених символами пробілу (ASCII-код 20h, далі SP), і закінчується комбінацією з двох символів: повернення каретки (ASCII-код 0Dh, далі CR) і переклад рядка (ASCII-код 0Ah, далі LF) . Елементи рядка запиту представлені наступними полями:

* Метод (method) - визначає метод обробки, застосовуваний до запитуваного ресурсу. У залежності від зазначеного методу формат запиту може бути різним. Допустимі методи:
  + OPTIONS
  + GET
  + HEAD
  + POST
  + PUT
  + DELETE
  + TRACE
* URI (Universal Resource Identifier) ​​ресурсу (resource URI) - вказує місце розташування запитуваного ресурсу в стандартизованому форматі, тобто є адресою ресурсу. При використанні методу GET даний рядок може містити в собі набір параметрів, що передаються серверу у вигляді рядків формату «імя\_параметра = значеніе\_параметра», розділених символами амперсанда '&'. Блок параметрів знаходиться в кінці рядка URI і відокремлюється від адреси символом знаку питання '?'.
* Версія протоколу HTTP - при розробці програми була реалізована підтримка прийому запитів, відповідних версіями 1.0 та 1.1, яким відповідають рядки «HTTP/1.0» і «HTTP/1.1» відповідно.

Блок заголовків, наступний за рядком запиту, може складатися з одного або більше заголовків:

* Тема запиту - містить поля, службовці модифікаторами запиту і містять інформацію про запит і про конфігурацію клієнтської машини.
* Тема об'єкта - у разі, якщо запит включає в себе деякий об'єкт (довільний набір даних), поля цієї таблиці описують об'єкт, вказуючи його формат, кодування та інші параметри.
* Загальний заголовок - містить службові параметри, необхідні для забезпечення коректності передачі та включення додаткових послуг, таких, як кешування.

Розділ заголовків закінчується двома парами символів CR і LF, що дозволяє легко визначити факт закінчення прийому запиту в силу того, що сам запит подібну комбінацію символів містити не може.

Відповідь, що відправляється сервером клієнтові, може бути створений лише в результаті обробки клієнтського запиту. Він містить опис результатів виконання запиту і, якщо були запитані дані, включає в себе запитаний ресурс.

За своєю структурою відповідь складається з наступних частин:

* Рядок стану
* Блок заголовків
* Об'єкт

Рядок стану складається з трьох полів, розділених символами SP, і містить в кінці послідовність символів CR, LF. Елементи рядка стану:

* Версія протоколу HTTP - розроблена програма завжди використовує рядок «HTTP/1.1».
* Код стану (status code) - трьохсимвольний цифровий код, який ідентифікує результат виконання запиту. Хоча стандартом визначений достатньо великий набір кодів стану, у програмі використовуються наступні коди:
  + 200 - успішне виконання;
  + 400 - некоректний запит;
  + 401 - несанкціонований доступ;
  + 404 - ресурс не знайдений;
  + 405 - непридатний метод;
  + 505 - непідтримувана версія HTTP.
* Фраза стану (reason phrase) - коротка фраза, яка пояснює код стану виконання запиту. Стандартом запропонований стандартний набір фраз, проте в програмі цей набір був кілька модифікований.

Блок заголовків, наступний за рядком стану, може складатися з одного або більше заголовків:

* Тема запиту
* Тема об'єкта
* Загальний заголовок

Розділ заголовків закінчується двома парами символів CR і LF, після чого слід довільний набір символів - об'єкт. При роботі програми такими об'єктами можуть бути тільки гіпертекстові документи у форматі HTML, динамічно генеруються модулями, що підключаються.

4.2 Протокол HTTPS

HTTPS— схема URI, що синтаксично ідентична http: схемі яка звичайно використовується для доступу до ресурсів Інтернет. Використання https:URL указує, що протокол HTTP має використовуватися, але з різним типовим портом (443) і додатковим шаром шифрування/автентифікації між HTTP і TCP. Ця схема була винайдена у компанії Netscape Communications Corporation для забезпечення автентифікації і шифрування комунікацій і широко використовується на Інтернет для чутливих для безпеки комунікацій як наприклад платіжні операції і корпоративні логіни.

Власне кажучи, HTTPS не окремий протокол, а комбінація нормальної взаємодії HTTP через SSL або TLS. Це гарантує помірний захист від підслуховування і від нападу «людина-посередині» (man-in-the-middle) (якщо це здійснюється належним чином і уповноважені на видачу сертифікатів верхнього рівня роблять свою роботу належним чином).

Типовим TCP портом HTTPS є 443 (для HTTP типове значення — 80).

Щоб підготовити веб-сервер для прийняття https транзакцій адміністратор повинен створити сертифікат з відкритим ключем для веб-сервера. Ці сертифікати можуть бути створені на UNIX сервері такими програмами, як наприклад OpenSSL ssl-ca. Цей сертифікат повинен бути підписаним уповноваженим на видачу сертифікатів (certificate authority) який засвідчує, що утримувач сертифікату — той самій, що стверджується у сертифікаті. Браузери розповсюджуються з сертифікатами уповноважених на видачу сертифікатів верхнього рівня, таким чином браузери можуть перевірити сертифікати підписані ними.

Організації можуть також мати їх власні уповноважені на видачу сертифікатів, особливо якщо вони відповідальні за конфігурацію браузерів, що мають доступ до їх власних сайтів (наприклад, сайти на внутрішній мережі компанії), оскільки вони можуть тривіально додати свого власного сертифіката до браузера.

Деякі сайти використовують самостійно підписані сертифікати. Їх використання забезпечує захист проти підслуховування але є ризик нападу «людина-посередині». Для запобігання нападу необхідна перевірка сертифікату деяким іншим методом (наприклад подзвонити власнику сертифіката задля перевірки контрольної суми сертифіката).

4.3 Протокол WebSocket

WebSocket - протокол повнодуплексного двостороннього зв'язку поверх TCP-з'єднання, призначений для обміну повідомленнями між браузером і веб-сервером в режимі реального часу. Програмісти і раніше намагалися вирішити проблему «рівноправності» між сервером і клієнтом. Вони експериментували з двостороннім зв'язком, використовуючи запити XMLHttpRequest. Але це було дуже повільно, оскільки кожного разу доводилося створювати нове TCP-з'єднання для кожного повідомлення.

На відміну від XMLHttpRequest WebSockets встановлює одне TCP-з'єднання і підтверджує, що сервер може спілкуватися з WebSocket, роблячи спеціальні перевірки, після чого сервер і клієнт можуть відправляти текстові повідомлення через встановлене з'єднання при необхідності, в результаті чого зв'язок стає швидше. З'єднання постійно тримається відкритим, але не передає зайвих HTTP заголовків. При цьому в веб-сокетах немає обмежень на кількість з'єднань.

В даний час в World Wide Web Consortium (W3C) відбувається стандартизація API Web Sockets, а в Internet Engineering Task Force (IETF) знаходиться на затвердження стандарт протоколу Web Socket.

В тій чи іншій міру WebSocket підтримується в наступних браузерах:

* Google Chrome (починаючи з версії 4.0.249.0);
* Apple Safari (починаючи з версії 5.0.7533.16);
* Mozilla Firefox (починаючи з версії 4);
* Opera (починаючи з версії 10.70 9067);

Різні версії браузерів підтримують різні версії протоколів. Причому саме розвиток протоколу дозволяє припускати, що з часом він може змінитися несумісним чином. Однак, малоймовірно що зміниться WebSocket API.

Протокол працює наступним чином, як тільки сторінка сайту вирішила, що вона хоче відкрити WebSocket на сервер, вона створює спеціальний javascript-об'єкт.

Лістинг 4.1 - Створення об’єкту WebSocket y JavaScript

**var *ws*** = **new** WebSocket(**"ws://127.0.0.1:9000/websocket"**);

***ws***.onmessage = **function** (event) {

*onMessage*(event);

};

***ws***.onopen = **function** (event) {

*logTextToConsole*(**'CONNECTED'**);

};

***ws***.onerror = **function** (event) {

*onError*(event)

};

Все починається так само як в звичайному HTTP-запиті. Браузер підключається по протоколу TCP на 9000-й порт сервера, але дає трохи незвичайний GET-запит, який наведено у лістингу 4.2 і, якщо сервер підтримує WebSocket, то він відповідає, як наведено в лістингу 4.3.

Лістинг 4.2 - HTTP запит для створення WebSocket

GET /websocket HTTP/1.1

Upgrade: WebSocket

Connection: Upgrade

Host: 127.0.0.1

Origin: http://127.0.0.1:9000

Якщо браузер це влаштовує, то він просто залишає TCP-з'єднання відкритим. Все - встановлення зв'язку скоєно, канал обміну даними готовий.

Лістинг 4.3 - Відповідь сервера на створення WebSocket

HTTP/1.1 101 Web Socket Protocol Handshake

Upgrade: WebSocket

Connection: Upgrade

WebSocket-Origin: http://127.0.0.1:9000

WebSocket-Location: ws://localhost:9000/websocket

Як тільки одна сторона хоче передати іншій якусь інформацію, вона відправляє дата-фрейм такого вигляду: 0x00, <рядок в кодуванні UTF-8>, 0xFF. Тобто просто рядок тексту - послідовність байт, до якої спереду приставлений нульовий байт 0x00, а в кінці - 0xFF. Без заголовків і метаданих.

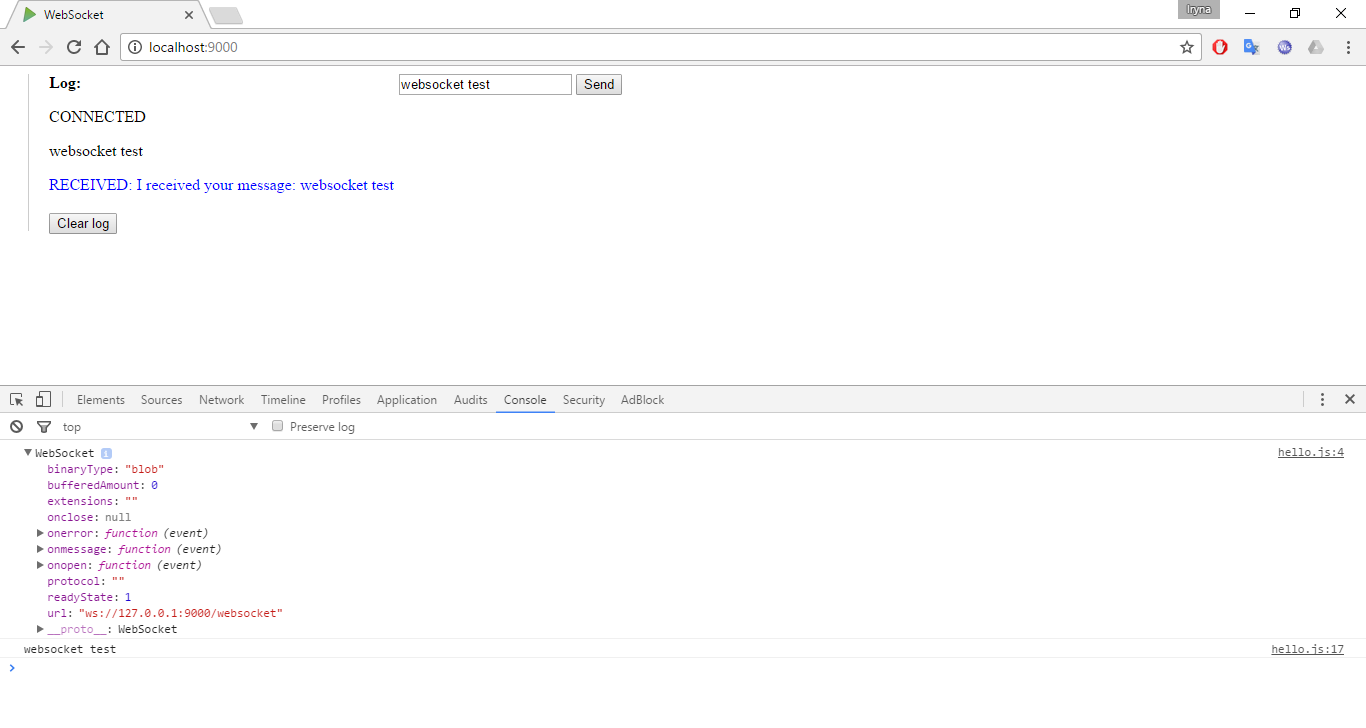


Рисунок 4.1 - Візуальне відображення повідомлення та відповіді WebSocket.

У протоколу є свої переваги, наприклад:

* Швидкість і ефективність передачі забезпечує малий розмір переданих даних, який іноді навіть буде міститися в один TCP-пакет - тут, звичайно ж, все залежить від логіки розробника. При цьому варто врахувати, що з'єднання вже готове - не треба витрачати час і трафік на його встановлення.
* Стандартність, яка усуне потребу в цілій низці технологій: Comet і все що надбудовано над ним (Bayuex, LongPolling, MultiPart і так далі).
* Час життя каналу - WebSockets не мають обмежень на час життя в неактивному стані. Це означає, що більше не треба періодично оновлювати з'єднання, тому що його не має права закривати проксі-сервери. Значить, з'єднання може висіти в неактивному вигляді і не вимагати ресурсів. Проте, на сервері будуть забиватися TCP-сокети. Для цього достатньо використовувати хороший мультиплексор, і тоді сервер легко зможе підтримувати до мільйона відкритих коннектов.
* Комплексні веб-додатки - в HTTP передбачено обмеження на число одночасних відкритих сесій до одного сервера. Тому для безлічі різних асинхронних блоків на сторінці доводиться робити не тільки серверний, а й клієнтський мультиплексор. Це обмеження не поширюється на протокол WebSocket. Відкривається стільки, скільки необхідно.
* Крос-доменні додатки - про обмеження у протоколі вводяться не за принципом «з-того-ж-джерела», а з «дозволеного-джерела», і визначаються не на клієнті, а на сервері.

Для перетворення з’єднання між клієнтом і сервером з HTTP / 1.1 в WebSocket використовується доступний в HTTP / 1.1 механізм зміни протоколу .

Але є складність: оскільки "Upgrade" є hop-by-hop заголовком, то він не передається від клієнта до проксі сервера. При прямому проксінгу клієнти можуть використовувати метод CONNECT, щоб обійти цю проблему. Однак при зворотному проксінгу такий підхід не працює, оскільки клієнт нічого про проксі сервер не знає, і потрібна спеціальна обробка на проксі сервері.

У ранніх реалізаціях WebSocket існувала вразливість, яка називається «отруєний кеш» (cache poisoning). Вона дозволяла атакувати кешуючий проксі-сервера, зокрема, корпоративні.

Атака здійснювалася так:

1. Хакер заманює довірливого відвідувача (далі Жертва) на свою сторінку.
2. Сторінка відкриває WebSocket-з'єднання на сайт хакера. Передбачається, що Жертва сидить через проксі.Власне, на проксі і спрямована ця атака.
3. Сторінка формує спеціального виду WebSocket-запит, який ряд проксі серверів не розуміють.
4. Вони пропускають початковий запит через себе (який містить Connection: upgrade) і думають, що далі йде вже наступного HTTP-запит.
5. Але насправді там дані, що йдуть через вебсокет! І обидві сторони вебсокета (сторінка і сервер) контролюються Хакером. Так що хакер може передати в них щось схоже на GET-запит до відомого ресурсу, наприклад http://code.jquery.com/jquery.js, а сервер відповість «нібито кодом jQuery» з кешованими заголовками.
6. Проксі слухняно проковтне цю відповідь і закешує «нібито jQuery».
7. В результаті при завантаженні наступних сторінок будь-який користувач, який використовує той же проксі, що і Жертва, отримає замість http://code.jquery.com/jquery.js хакерський код.

Тому ця атака і називається «отруєний кеш». Така атака можлива не для будь-яких проксі, але при аналізі уразливості було показано, що вона не теоретична, і вразливі проксі дійсно є. Тому придумали спосіб захисту - «маску». Для того, щоб захиститися від атаки, і придумана маска.

Ключ маски - це випадкове 32-бітове значення, яке варіюється від пакету до пакету. Тіло повідомлення проходить через XOR ^з маскою, а одержувач відновлює його повторним XOR з нею (можна легко довести, що (x ^ a) ^ a == x).

Маска служить двом цілям:

1. Вона генерується браузером. Тому тепер хакер не зможе управляти реальним змістом тексту повідомлення.Після накладання маски воно перетвориться в бінарну мішанину.
2. Одержаний пакет даних вже точно не може бути сприйнятий проміжним проксі як HTTP-запит.

Накладення маски вимагає додаткових ресурсів, тому протокол WebSocket не вимагає її.

Якщо за цим протоколом зв'язуються два клієнта (не обов'язково браузери), довіряють один одному і посередникам, то можна поставити біт Маскав 0, і тоді ключ маски не вказується.

# ВИСНОВКИ

Основною метою даної курсової роботи було проектування комп’ютерної мережі із використанням WebSocket протоколу.

При проектуванні даної мережі були поставлені і успішно вирішені задачі вибору мережної архітектури, спроектовано фізичну та логічну топологію, розглянуті питання переваги WebSocket протоколу та розроблено WebSocket сервер.

Дана мережа має велике значення для функціонування закладу, для якого вона проектувалась. Внаслідок забезпечення безперебійного зв’язку між робочими станціями в мережі та доступу до Інтернет вона забезпечить високу продуктивність роботи працівників закладу.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Планирование и поддержка сетевой инфраструктуры. Учебный курс MSCE / Пер. с англ. М.: Издательско-торговий дом «Русская Редакция», 2005.
2. Методичне забезпечення з предмету «Комп’ютерні мережі».
3. Галушин С. Я. Сети ЭВМ и их защита. – Санкт-Перербург, 2000.
4. Вишневський В. М. Теоретичні основи проектування комп’ютерних мереж. Навчальний посібник. – Техносфера, 2004.
5. Буров Є. М. Комп’ютерні мережі. Вид. Львів: БаК, 2003. – 584с.