МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

М.В. МЕЗЕНЦЕВ, В.І. НОСКОВ

КОМП’ЮТЕРНІ МЕРЕЖІ

Навчально-методичний посібник

Затверджено
редакционно-видавничою
радою університету,
протокол № 2 від 27.06.2024

Харків НТУ «ХПІ»
2024

УДК 681.3.06

М 59

Рецензенти:

В.В. Давидов, д-р техн. наук, доцент, завідувач кафедри інформаційних технологій Приватної установи «Університет науки підприємництва та технологій»

О.О. Сєрков, д-р техн. наук, проф., НТУ «ХПІ»

М 59 **Мезенцев М.В.**

Комп'ютерні мережі. Навч.-метод посібник / М.В Мезенцев, В.І. Носков – Харків: Х.: НТУ "ХПІ", 2024. – 236 с.

Посібник дає цілісне уявлення про класифікацію та топологію комп'ютерних мереж. Розглянуто теоретичні основи передачі даних, кодування даних та лінії зв'язку. Дано поняття протоколів передачі даних та оцінка ефктивності при їх застосуванні. Проаналізовано технології *ETHERNET, TOKEN RING* . Наведено активне устаткування локальних мереж. Розглянуто адресацію в *IP*-мережах, протоколи внутрішньої і зовнішньої маршрутизації. Наведено сучасні засоби тестування комп'ютерних мереж, їхні переваги та недоліки. Розглянуто проєктування системи тестування мереж з використанням протоколів *TCP/IP, POP*3, *FTP* та проєктування засобів діагностики мереж з бібліотеками *ICS, Indy, Winsock.*

Іл.87. Табл. 4. Бібліогр. 10 назв.

ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ЕОМ – електронно-обчислювальна машина
КМ – комп'ютерная мережа
ПК – персональний комп'ютер
СКБД – система керування базой даних
МОС – міжнародна організація зі стандартизації
МВВС – модель взаємодії відкритих систем
ОС – операційна система
КПД – канал передачі даних
IEEE – Institute of Electrical and Electronics Engineers
ASCII – American Standard Code for Information Interchange
NIC – Network Interface Card
IP – Internet Protocol
IPv6 – IP протокол 6 версії
ARP – Address Resolution Protocol
RARP – Reverse Address Resolution Protocol
ICMP – Internet Control Message Protocol
TCP – Trasmission Control Protocol
UDP – User Datagramm Protocol
CIDR – Classless Inter-Domain Routing
RIP – Router Information Protocol
OSPF – Open Shortest Path First
BGP – Border Gateway Protocol
LAN – Local Area Networks
WAN – Wide Area Networks
MAN – Metropolitan Area Networks
CAN – Campus Area Networks
OSI – OpenSystem Interconnect
ISO – International Organization for Standardization
DLL – Data Link Layer
MAC – Media Access Control
CSMA/CD – Carrier Sense Multiple Access / Collision Detection
CSMA/CA – Carrier Sense Multiple Access / Collision Avoidance
FDDI – Fiber Distributed Data Interface
SMTP – Simple Mail Transfer Protocol
SLIP – Serial Line Internet Protocol
HDLC – High-Level Data Link Control
LCP – Link Control Protocol
NCP – Network Control Protocols
PPP – Point-to-Point Protocol
SLIP – Serial Line IP
CSLIP – Compressed SLIP
FTP – File Transport Protocol
WWW – World Wide Web

IPX – Internetwork Packet Exchange
SPX – Sequenced Packet Exchange
NetBIOS – Network Basic Input/Output System
NetBEUI – NetBIOS Extended User Interface
SMB – Server Message Block
ISDN – Integrated Services Digital Network
STP – shielded twisted pair
UTP – unshielded twisted pair
NEXT – Near End CrossTalk
FEXT – Far End CrossTalk
Wi-Fi – Wireless Fidelity
IrDA – Infrared Data Association
NRZ – Non-Return to Zero
NRZI – Non-Return to Zero Inverted
AMI – Bipolar Alternate Mark Inversion
*PAM*5 *– Pulse Amplitude Modulation*
ARQ – Automatic Repeat-reQuest
ACK – Acknowledge
NACK – Negative Acknowledge
SAW - Stop And Wait
ABP – Altarnating Bit Protocol
GBN – Go Back N
SR – Selective Repeat
PDV – Path Delay Value
PW – Path Variability Value
IPG –interpacket gap

ВСТУП

Комп'ютерні мережі (КМ) є природним результатом розвитку обчислювальної техніки. Перші мережі почали з'являтися в 70-х роках, коли треба було налагоджувати обмін даними між якими-небудь потужними супер-ЕОМ і терміналами, на яких працювали користувачі. Термінали могли бути розташовані на багато десятків і сотень кілометрів від ЕОМ, тому зв'язок здійснювався через телефонні лінії за допомогою модемів. Такий обмін інформацією можна віднести до перших прикладів комп'ютерних телекомунікацій.

Для налагодження обміну даними почалася розробка спеціального програмного забезпечення і пристроїв сполучення комп'ютерів. Однак пристрої сполучення розроблялися тільки для конкретних типів комп'ютерів, і це сильно стримувало розвиток обчислювальних мереж.

У середині 80-х років у цій галузі відбувся прорив завдяки розробці мережних стандартів 802.1, 802.2, …, 802.12. Вони були орієнтовані на персональні комп'ютери, які в той час уже стрімко завойовували ринок. Стало можливим з'єднувати в мережі персональні комп'ютери на підприємствах і в підрозділах, щоб поєднувати обчислювальні потужності при роозв'язанні складних завдань, щоб організувати спільний доступ до дискових масивів інформації й т.п. Так з'явилися перші комп'ютерні мережі – прообраз сучасних локальних мереж, які докорінно змінили роботу користувачів і розширили їхні можливості.

З появою комп'ютерних мереж вдалося ефективно розв'язати такі важливі проблеми організації міжмашинної взаємодії, як:

- забезпечення майже необмеженого доступу користувачів до ресурсів усіх комп'ютерів мережі, незалежно від їх територіального знаходження;
- забезпечення відкритості мережі, тобто допустимості включення до неї пристроїв з різною архітектурою та можливостями;
- підвищення ефективності роботи комп'ютерів за рахунок їх спеціалізації;
- забезпечення високої надійності обробки та передачі інформації за рахунок наявності альтернативних та резервних засобів обчислювальної техніки і зв'язку, а також адаптивних алгоритмів керування КМ;
- забезпечення високої економічності обробки інформації за рахунок повного завантаження та комбінованого використання на різних рівнях макро-, міні- і мікрокомп'ютерів;
- надання нових інформаційних послуг користувачам: розподілені бази даних, інтерактивні інформаційні служби, електронні бюлетені, комп'ютерна передача зображень та ін.

Ці переваги КМ роблять їх на сьогоднішній день і в перспективі найбільш прогресивною формою використання засобів обчислювальної техніки.

Задача аналізу та діагностики комп'ютерної мережі завжди була важливою проблемою. У процесі еволюції обчислювальної техніки почали активно розвиватися технології, що використовують комп'ютерну мережу. Тому виникла потреба в програмних засобах, що можуть аналізувати та діагностувати стан мережі. Їхньою метою є розробка систем аналізу комп'ютерної мережі.

Системи аналізу мережі належать до класу інструментальних програмних засобів для моніторингу мережного стану і виявлення деяких типів мережних проблем.

Однією з причин підвищеної уваги до систем аналізу мережі є існування великого кола комерційних і соціальних галузей, що користуються комп'ютерними технологіями. У цих галузях аналіз мережі, необхідний для безперебійної роботи, буде сприйнятий дуже успішно. Очікується, що застосування подібних систем істотно зменшить кількість часу, затрачувану на відновлення працездатності мережі, що істотно полегшить рутинну працю системного адміністратора.

Навчальний посібник «Комп'ютерні мережі. Діагностистика комп'ютерних мереж» призначено для студентів вищих технічних навчальних закладів напрямку «Комп'ютерна інженерія".

У розділах 1–2 розглянуто класифікацію та топологію комп'ютерних мереж, основні моделі взаємозв'язку відкритих систем, стандартні стеки комунікаційних протоколів.

Розділи 3–6 описують теоретичні основи передачі даних, методи доступу і їх класифікацію, способи комутації та лінії зв'язку, які застосовуються в мережах передачі даних, підключення ліній зв'язку, кодування сигналів, методи цифрового кодування, передачу даних на канальному рівні

У розділах 7–11 розглянуто технологію *ETHERNET, FAST ETHERNET, GIGABIT ETHERNET,* адресацію в *IP*-мережах, протоколи *IP, ARP, RARP, IPV6,* протоколи внутрішньої і зовнішньої маршрутизації (*RIP, OSPF, BGP*), протоколи транспортного рівня (*UDP, TCP*).

Розділи 12–20 присв'ячені діагностиці комп'ютерних мереж та проектуванню засобів діагностики мереж з використанням різних бібліотек.

1 КЛАСИФІКАЦІЯ КОМП'ЮТЕРНИХ МЕРЕЖ. ТОПОЛОГІЯ КМ. ВИМОГИ, ЩО СТАВЛЯТЬСЯ ДО КМ

Комп'ютерна мережа це система розподіленої обробки інформації, яка містить як мінімум два комп'ютери, що взаємодіють між собою через канали передачі даних. Або, іншими словами, мережа являє собою сукупність з'єднаних один з одним ПК та інших обчислювальних пристроїв, таких, як принтери, факсимільні апарати і модеми. Комп'ютерні мережі уявляють собою систему масового обслуговування. Мережа дає можливість окремим співробітникам організації взаємодіяти один з одним і звертатися до спільно використовуваних ресурсів; дозволяє їм одержувати доступ до даних, які зберігаються на персональних комп'ютерах у видалених офісах, і встановлювати зв'язок з постачальниками.

1.1 Класифікація комп'ютерних мереж

Для класифікації комп'ютерних мереж використовуються різні ознаки, вибір яких полягає в тому, щоб виділити з існуючого різноманіття такі, які дозволили б забезпечити дану класифікаційну схему такими обов'язковими якостями:

- можливістю класифікації всіх як існуючих, так і перспективних КМ;
- диференціацією істотно різних мереж;
- однозначністю класифікації будь-якої комп'ютерної мережі;
- наочністю, простотою і практичною доцільністю класифікаційної схеми.

Певна невідповідність цих вимог робить завдання вибору раціональної схеми класифікації КМ досить складним, таким, що не знайшло до цього часу однозначного рішення. В основному КМ класифікуються за ознаками структурної і функціональної організації.

За призначенням КМ розподіляють:

- на обчислювальні;
- інформаційні;
- змішані (інформаційно-обчислювальні).

Обчислювальні мережі призначені головним чином для розв'язування завдань користувачів з обміном даними між їхніми абонентами. Інформаційні мережі орієнтовані в основному на надання інформаційних послуг користувачам. Змішані мережі поєднують функції перших двох.

За типом комп'ютерів, які входять до складу КМ, розрізняють:

- однорідні комп'ютерні мережі, які складаються із програмно-спільних ЕОМ;
- неоднорідні, до складу яких входять програмно-несумісні комп'ютери.

Особливе значення має класифікація **за територіальною ознакою**, тобто за величиною території, яку покриває мережа. І для цього є вагомі причини, тому що відмінності технологій локальних і глобальних мереж дуже значні, незважаючи на їхнє постійне зближення.

Класифікуючи мережі за територіальною ознакою, розрізняють:

- локальні (*Local Area Networks – LAN*) мережі;
- глобальні *(Wide Area Networks – WAN*) мережі;
- міські (*Metropolitan Area Networks – MAN*) мережі.

LAN – зосереджені на території не більше 1–2 км; побудовані з використанням дорогих високоякісних ліній зв'язку, які дозволяють, застосовуючи прості методи передачі даних, досягати високих швидкостей обміну даними порядку 1000 Мбіт/с. Послуги, які надаються, відрізняються широкою різноманітністю і звичайно передбачають реалізацію в режимі *on-line*.

WAN – поєднують комп'ютери, розосереджені на відстані сотень і тисяч кілометрів. Часто використовуються вже існуючі не дуже якісні лінії зв'язку. У зв'язку з цим мають більш низькі, чим у локальних мережах, швидкості передачі даних. Для стійкої передачі дискретних даних застосовуються більш складні методи і устаткування, чим у локальних мережах.

MAN – займають проміжне положення між локальними і глобальними мережами. При досить великих відстанях між вузлами (десятки кілометрів) вони мають якісні лінії зв'язку і високі швидкості обміну, іноді навіть більш високі, чим у класичних локальних мережах. Як і у випадку локальних мереж, при побудові *MAN* уже існуючі лінії зв'язку не використовуються, а прокладаються заново.

Також додатково виділяють:

- кампусні мережі (*Campus Area Network – CAN*), які поєднують значно віддалені одна від однієї абонентські системи, або локальні мережі, але ще не вимагають віддалених комунікацій через телефонні лінії і модеми;

Відмітні **ознаки локальної мережі**:

- висока швидкість передачі, більша пропускна здатність;
- низький рівень помилок передачі (або, що те ж саме, високоякісні канали зв'язку). Припустима ймовірність помилок передачі даних повинна мати порядок $10^{-7} – 10^{-8}$;
- ефективний, швидкодіючий механізм керування обміном;
- обмежене число комп'ютерів, що підключаються до мережі.

Глобальні мережі відрізняються від локальних тим, що розраховані на необмежене число абонентів і використовують, як правило, не занадто якісні канали зв'язку і порівняно низьку швидкість передачі, а механізм керування обміном у них у принципі не може бути гарантовано швидким.

У глобальних мережах набагато важливіше не якість зв’язку, а сам факт її існування.

Правда, зараз уже не можна провести чітку і однозначну межу між локальними і глобальними мережами. Більшість локальних мереж має вихід у глобальну мережу, але характер переданої інформації, принципи організації обміну, режими доступу до ресурсів всередині локальної мережі, як правило, сильно відрізняються від тих, що прийнято в глобальній мережі. І хоча всі комп’ютери локальної мережі в даному випадку включені також і в глобальну мережу, специфіки локальної мережі це не скасовує. Можливість виходу в глобальну мережу залишається всього лише одним з ресурсів користувачів локальної мережі.

Наступною, не менш розповсюдженою класифікацією є класифікація КМ за типом топології.

1.2 Топологія КМ

Під топологією комп’ютерної мережі розуміють фізичне розташування комп’ютерів мережі відносно один одного та спосіб їх з’єднання лініями зв’язку. Важливо відзначити, що поняття топології належить, насамперед, до локальних мереж, у яких структуру зв’язків можна легко простежити. У глобальних мережах структура зв’язків зазвичай схована від користувачів і є не занадто важливою, тому що кожний сеанс зв’язку може виконуватися за своїм власним шляхом.

Існують три основних топології мережі:

1) «шина» (*bus*), при якій всі комп’ютери паралельно підключаються до однієї лінії зв’язку і інформація від кожного комп’ютера одночасно передається всім іншим комп’ютерам (рис. 1.1);

2) «зірка» (*star*), при якій до одного центрального комп’ютера приєднуються інші периферійні комп’ютери, причому кожний з них використовує свою окрему лінію зв’язку (рис. 1.2);

3) «кільце» (*ring*), при якій кожний комп’ютер передає інформацію завжди тільки одному комп’ютеру, наступному в ланцюжку, а одержує інформацію тільки від попереднього комп’ютера в ланцюжку, і цей ланцюжок замкнутий в кільце (рис. 1.3).

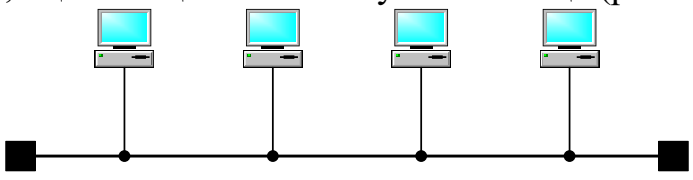

Рисунок 1.1 – Мережна топологія «шина»

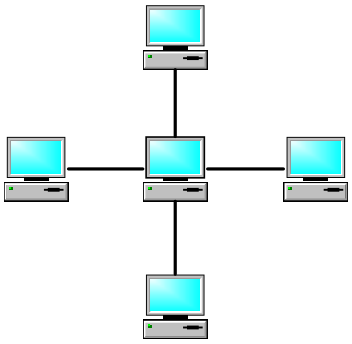

Рисунок 1.2 – Мережна топологія «зірка»

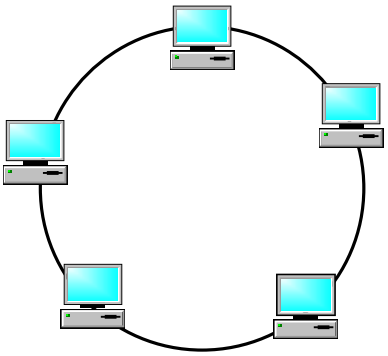

Рисунок 1.3 – Мережна топологія «кільце»

На практиці нерідко використовують і комбінації базових топологій, але більшість мереж орієнтовані саме на ці три. Розглянемо тепер коротко особливості перерахованих мережних топологій.

1.2.1 *Топологія «шина».*

Топологія «шина» (або, як її ще називають, «загальна шина») самою своєю структурою припускає ідентичність мережного устаткування комп'ютерів, а також рівноправність усіх абонентів. При такому з'єднанні комп'ютери можуть передавати інформацію тільки по черзі, тому що лінія зв'язку єдина. У іншому випадку передана інформація буде спотворюватися в результаті накладення(конфлікту (колізії). Таким чином, у топології «шина» реалізується режим напівдуплексного (*half duplex*) обміну (в обох напрямках, але по черзі, а не одночасно).

У топології «шина» відсутній центральний абонент, через який передається вся інформація, що збільшує її надійність (адже при відмові будь-якого центру перестає функціонувати вся керована цим центром система). Додавання нових абонентів у шину досить просте і звичайно можливе навіть під час роботи мережі. У більшості випадків при використанні шини потрібна мінімальна кількість сполучного кабелю в порівнянні з іншими топологіями. Втім необхідно врахувати, що до кожного комп'ютера (крім двох крайніх) підходять два кабелі, що не завжди зручно.

Розв’язання можливих конфліктів, у цьому випадку, лягає на мережне устаткування кожного окремого абонента, апаратура мережного адаптера при топології «шина» виходить складнішою, ніж при інших топологіях. Однак через широке розповсюдження мереж з топологією «шина» (*Ethernet, ArcNet)* вартість мережного устаткування виходить не занадто високою.

Топології «шина» не страшні відмови окремих комп’ютерів, тому що всі інші комп’ютери мережі можуть нормально продовжувати обмін. Може здатися, що «шині» не страшний і обрив кабелю, оскільки в цьому випадку ми одержимо дві цілком працездатні «шини». Однак через особливості поширення електричних сигналів довгими лініями зв’язку необхідно передбачати включення на кінцях «шини» спеціальних пристроїв – термінаторів, показаних на рис. 1.1 у вигляді прямокутників. Без включення термінаторів сигнал відбивається від кінця лінії і спотворюється так, що зв’язок через мережу стає неможливим. Таким чином, при розриві або ушкодженні кабелю порушується узгодження лінії зв’язку і припиняється обмін даними навіть між тими комп’ютерами, які залишилися з’єднаними між собою. Коротке замикання в будь-якій точці кабелю «шини» виводить із ладу всю мережу. Будь-яку відмову мережного устаткування в «шині» дуже важко локалізувати, тому що всі адаптери включені паралельно, і зрозуміти, який з них вийшов з ладу, не так-то просто.

При проходженні інформації лініями зв’язку мережі з топологією «шина» інформаційні сигнали послабляються і ніяк не відновлюються, що накладає тверді обмеження на сумарну довжину ліній зв’язку. Крім того, кожний абонент може одержувати з мережі сигнали різного рівня залежно від відстані до передавального абонента. Це ставить додаткові вимоги до приймальних вузлів мережного устаткування. Для збільшення довжини мережі з топологією «шина» часто використовують кілька сегментів (кожний з яких являє собою «шину»), з’єднаних між собою за допомогою спеціальних відновлювачів сигналів – **репітерів** (repeaters).

Однак таке нарощування довжини мережі не може тривати нескінченно, тому що існують ще і обмеження, пов’язані з кінцевою швидкістю поширення сигналів лініями зв’язку.

1.2.2 *Топологія «зірка».*

Топологія «зірка» – це топологія з явно виділеним центром, до якого підключаються всі інші абоненти. Весь обмін інформацією йде винятково через центральний комп’ютер, на який лягає значне навантаження, тому нічим іншим, крім мережі, він займатися не може. Зрозуміло, що мережне устаткування центрального абонента повинне бути істотно більш складним, чим устаткування периферійних абонентів. Про рівноправність абонентів у цьому випадку говорити не доводиться. Як правило, саме центральний комп’ютер є найпотужнішим і саме на нього покладають усі функції керування обміном. Ніякі конфлікти в мережі з топологією «зірка»

у принципі неможливі, тому що керування повністю централізоване, конфліктувати нема чому.

Якщо говорити про стійкість зірки до відмов комп'ютерів, то вихід з ладу периферійного комп'ютера ніяк не відбивається на функціонуванні частини мережі, що залишилася, зате будь-яка відмова центрального комп'ютера робить мережу повністю непрацездатною. Тому треба вживати спеціальних заходів щодо підвищення надійності центрального комп'ютера і його мережної апаратури. Обрив будь-якого кабелю або коротке замикання в ньому при топології «зірка» порушує обмін тільки з одним комп'ютером, а всі інші комп'ютери можуть нормально продовжувати роботу.

На відміну від «шини», у «зірці» на кожній лінії зв'язку знаходяться тільки два абоненти: центральний і один з периферійних. Найчастіше для їхнього з'єднання використовуються дві лінії зв'язку, кожна з яких передає інформацію тільки в одному напрямку. Таким чином, на кожній лінії зв'язку є тільки один приймач і один передавач. Все це істотно спрощує мережне встаткування в порівнянні із «шиною» і не потребує застосування додаткових зовнішніх термінаторів. Проблема загасання сигналів у лінії зв'язку також вирішується в «зірці» простіше, ніж в «шині», адже кожний приймач завжди одержує сигнал одного рівня. Серйозний недолік топології «зірка» полягає у жорсткому обмеженні кількості абонентів.

Звичайно центральний абонент може обслуговувати не більше 8-16 периферійних абонентів. Якщо в цих межах підключення нових абонентів досить просте, то при їхньому перевищенні воно просто неможливо. Правда, іноді в зірці передбачається можливість нарощування, тобто підключення замість одного з периферійних абонентів ще одного центрального абонента (у результаті виходить топологія з декількох з'єднаних між собою "зірок").

Топологія "зірка", що показана на рис. 1.2, називається активною, або "справжньою зіркою". Існує також топологія, яка називається "пасивною зіркою". Вона тільки зовні схожа на зірку (рис. 1.4).

Рисунок 1.4 – Топологія «пасивна зірка»

У центрі мережі з даною топологією міститься не комп'ютер, а концентратор, або хаб (*hub*), що виконує ту ж функцію, що і репітер. Він

відновлює сигнали, що надходять, і пересилає їх в інші лінії зв'язку. Хоча схема прокладки кабелів подібна справжній або активній зірці, фактично ми маємо справу з шинною топологією, тому що інформація від кожного комп'ютера одночасно передається до всіх інших комп'ютерів, а центрального абонента не існує. Природно, "пасивна зірка" виходить дорожче звичайної «шини», тому що в цьому випадку обов'язково потрібно мати ще і концентратор. Однак вона надає цілий ряд додаткових можливостей, пов'язаних з перевагами зірки. Саме тому останнім часом "пасивна зірка" усе більше витісняє справжню "зірку", що вважається малоперспективною топологією.

Можна виділити також проміжний тип топології між активною і пасивною зіркою. У цьому випадку концентратор не тільки ретранслює сигнали, але і здійснює керування обміном, однак сам в обміні не бере участі.

Велика **перевага "зірки"** (як активної, так і пасивної) полягає в тому, що всі точки підключення зібрані в одному місці. Це дозволяє легко контролювати роботу мережі, локалізувати несправності мережі шляхом простого відключення від центру тих або інших абонентів (що неможливо, наприклад, у випадку «шини»), а також обмежувати доступ сторонніх осіб до життєво важливих для мережі точок підключення. До кожного периферійного абонента, у випадку "зірки", може підходити як один кабель (по якому йде передача в обох напрямках), так і два кабелі (кожний з них передає в одному напрямку), причому друга ситуація зустрічається частіше.

Загальним недоліком для всіх топологій типу «зірка» є значно більша, ніж при інших топологіях, витрата кабелю. Наприклад, якщо комп'ютери розташовані в одну лінію (як на рис. 1.1), то при виборі топології «зірка» знадобиться в кілька разів більше кабелю, чим при топології «шина». Це може істотно вплинути на вартість усієї мережі в цілому.

1.2.3 *Топологія «кільце».*

Топологія «кільце» – це топологія, у якій кожний комп'ютер з'єднаний лініями зв'язку тільки з двома іншими: від одного він тільки одержує інформацію, а іншому тільки передає. На кожній лінії зв'язку, як і у випадку зірки, працює тільки один передавач і один приймач. Це дозволяє відмовитися від застосування зовнішніх термінаторів. Важлива особливість кільця полягає в тому, що кожний комп'ютер ретранслює (відновлює) сигнал, тобто виступає в ролі репітера, тому загасання сигналу у всьому кільці не має ніякого значення, важливо тільки загасання між сусідніми комп'ютерами кільця. Чітко виділеного центру в цьому випадку не існує, всі комп'ютери можуть бути однаковими. Однак досить часто в кільці виділяється спеціальний абонент, що керує обміном або контролює обмін. Зрозуміло, що наявність такого керуючого абонента знижує

надійність мережі, тому що вихід його з ладу відразу ж паралізує весь обмін.

Точно кажучи, комп'ютери в кільці не є повністю рівноправними (на відміну, наприклад, від шинної топології). Одні з них обов'язково одержують інформацію від комп'ютера, що веде передачу в цей момент, раніше, а інші – пізніше. Саме на цій особливості топології і будуються методи керування обміном мережею, спеціально розраховані на «кільце». У цих методах право на наступну передачу (або, як ще говорять, на захоплення мережі) переходить послідовно до наступного по колу комп'ютера.

Підключення нових абонентів у «кільце» звичайно здійснюється зовсім безболісно, хоча і вимагає обов'язкової зупинки роботи всієї мережі на час підключення. Як і у випадку топології «шина», максимальна кількість абонентів у кільці може бути досить велика (до тисячі і більше). Кільцева топологія звичайно є самою стійкою до перевантажень, вона забезпечує впевнену роботу із самими великими потоками переданої мережею інформації, тому що в ній, як правило, немає конфліктів (на відміну від «шини»), а також відсутній центральний абонент (на відміну від зірки).

Сигнал у кільці проходить через усі комп'ютери мережі, тому вихід з ладу хоча б одного з них (або ж його мережного устаткування) порушує роботу всієї мережі в цілому. При будь-якому обриві або короткому замиканні в кожному з кабелів кільця робота всієї мережі також стає неможливою. Кільце найбільш уразливе до ушкоджень кабелю, тому в цій топології звичайно передбачають прокладку двох (або більше) паралельних ліній зв'язку, одна з яких знаходиться в резерві.

У той же час велика перевага топології "кільце" полягає в тому, що ретрансляція сигналів кожним абонентом дозволяє істотно збільшити розміри всієї мережі в цілому (часом до декількох десятків кілометрів). Кільце щодо цього істотно перевершує будь-які інші топології.

Недоліком кільця (у порівнянні із зіркою) можна вважати те, що до кожного комп'ютера мережі необхідно підвести два кабелі.

Іноді топологія «кільце» виконується на основі двох кільцевих ліній зв'язку, що передають інформацію в протилежних напрямках. Мета подібного рішення – збільшення (в ідеалі удвічі) швидкості передачі інформації. До того ж при ушкодженні одного з кабелів мережа може працювати з іншим кабелем (правда, гранична швидкість зменшиться).

Крім трьох розглянутих основних, базових топологій, нерідко застосовується також мережна топологія **«дерево» (*tree*),** яку можна розглядати як комбінацію декількох зірок. Як і у випадку зірки, дерево може бути активним, або справжнім (рис. 1.5), і пасивним (рис. 1.6). При активному дереві в центрах об'єднання декількох ліній зв'язку знаходяться центральні комп'ютери, а при пасивному – концентратори (хаби).

Застосовуються досить часто і комбіновані топології, наприклад, "зірково-шинна", "зірково-кільцева".

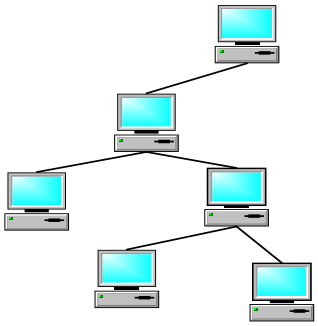

Рисунок 1.5 – Топологія «активне дерево»

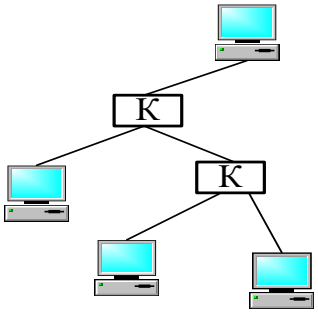

Рисунок 1.6 – Топологія «пасивне дерево». К – концентратори

1.3 Багатозначність поняття топології

Топологія мережі визначає не тільки фізичне розташування комп'ютерів, але, що набагато важливіше, характер зв'язків між ними, особливості поширення сигналів мережею. Саме характер зв'язків визначає ступінь відмовостійкості мережі, необхідну складність мережної апаратури, найбільш підходящий метод керування обміном, можливі типи середовищ передачі (каналів зв'язку), припустимий розмір мережі (довжина ліній зв'язку і кількість абонентів), необхідність електричного узгодження і багато чого іншого.

Коли в літературі згадується про топологію мережі, то можуть мати на увазі чотири зовсім різних поняття, що належать до різних рівнів мережної архітектури:

1. Фізична топологія (тобто схема розташування комп'ютерів і прокладки кабелів). За цим змістом, наприклад, "пасивна зірка" нічим не відрізняється від "активної зірки", тому її нерідко називають просто «зіркою».

2. Логічна топологія (тобто структура зв'язків, характер поширення сигналів мережею). Це, напевно, найбільш правильне визначення топології.

3. Топологія керування обміном (тобто принцип і послідовність передачі права на захоплення мережі між окремими комп'ютерами).

4. Інформаційна топологія (тобто напрямок потоків інформації, переданої мережею).

Наприклад, мережа з фізичною і логічною топологією «шина» може як метод керування використовувати естафетну передачу права захоплення мережі (тобто бути в цьому сенсі кільцем) і одночасно передавати всю інформацію через один виділений комп'ютер (бути в цьому сенсі зіркою).

1.4 Характеристики сучасних обчислювальних мереж

Продуктивність. Існує кілька основних характеристик продуктивності мережі:

- час реакції;
- пропускна здатність;
- затримка передачі.

Час реакції визначається як інтервал часу між виникненням запиту користувача до якої-небудь мережної служби і одержанням відповіді на цей запит.

Очевидно, що значення цього показника залежить від типу служби, до якої звертається користувач, від того, який користувач і до якого серверу звертається, а також від поточного стану елементів мережі – завантаженості сегментів, комутаторів і маршрутизаторів, через які проходить запит, завантаженості серверу і т.д.

Пропускна здатність відображає обсяг даних, переданих мережею або її частиною в одиницю часу.

Пропускна здатність виміряється або в бітах у секунду, або в пакетах у секунду. Пропускна здатність може бути миттєвою, максимальною і середньою.

Середня пропускна здатність обчислюється шляхом ділення загального обсягу переданих даних на час їхньої передачі, причому вибирається досить тривалий проміжок часу – година, день або тиждень.

Миттєва пропускна здатність відрізняється від середньої тим, що для усереднення вибирається дуже маленький проміжок часу – наприклад, 10 мс або 1 с.

Максимальна пропускна здатність – це найбільша миттєва пропускна здатність, зафіксована протягом періоду спостереження.

Затримка передачі визначається як затримка між моментом надходження пакета на вхід якого-небудь мережного пристрою або частини мережі і моментом появи його на виході цього пристрою. Цей параметр продуктивності за змістом близький до реакції мережі, але відрізняється тим, що завжди характеризує тільки мережні етапи обробки даних, без затримок обробки комп'ютерами мережі.

Пропускна здатність і затримка передачі є незалежними параметрами, так що мережа може характеризуватися, наприклад, високою пропускною здатністю, але вносити значні затримки при передачі кожного пакета.

Надійність і безпека. Для оцінки надійності використовується такі характеристики, як коефіцієнт готовності та безпека.

Коефіцієнт готовності означає частину часу, протягом якого система може бути використана. Готовність може бути поліпшена шляхом введення надмірності в структуру системи: ключові елементи системи повинні існувати в декількох екземплярах, щоб при відмові одного з них функціонування системи забезпечували інші.

Іншим аспектом загальної надійності є **безпека** (*security*), тобто здатність системи захистити дані від несанкціонованого доступу.

Ще одною характеристикою надійності є відмовостійкість (*fault wrance*). У мережах під **відмовостійкістю** розуміють здатність системи сховати від користувача відмову окремих її елементів. У відмовостійкій системі відмова одного з її елементів призводить до деякого зниження якості її роботи, а не до повної зупинки.

Розширюваність і масштабованість. Розширюваність (*extensibility*) означає можливість порівняно легкого додавання окремих елементів мережі (користувачів, комп'ютерів, додатків, служб), нарощування довжини сегментів мережі і заміни існуючої апаратури більш потужною. При цьому **принципово** важливо, що легкість розширення системи іноді може забезпечуватися в деяких досить обмежених границях. Масштабованість (*scalability*) означає, що мережа дозволяє нарощувати кількість вузлів і довжину зв'язків у дуже широких межах, при цьому продуктивність мережі не погіршується. Для забезпечення масштабованості мережі доводиться застосовувати додаткове комунікаційне устаткування і спеціальним способом структурувати мережу.

Прозорість. Прозорість (*transparency*) мережі досягається в тому випадку, коли мережа подається користувачам не як множина окремих комп'ютерів, зв'язаних між собою складною системою кабелів, а як єдина традиційна обчислювальна машина із системою поділу часу. Прозорість може бути досягнута на двох різних рівнях – на рівні користувача і на рівні програміста. На рівні користувача прозорість означає, що для роботи з вилученими ресурсами він використовує ті ж команди і звичні йому процедури, що і для роботи з локальними ресурсами. На програмному рівні прозорість полягає в тому, що додатку для доступу до вилучених ресурсів потрібні ті ж виклики, що і для доступу до локальних ресурсів.

Керованість. Керованість мережі має на увазі можливість централізовано контролювати стан основних елементів мережі, виявляти і розв'язувати проблеми, що виникають при роботі мережі, виконувати аналіз продуктивності і планувати розвиток мережі. В ідеалі засоби керування мережами є системою, що здійснює спостереження, контроль і керування кожним елементом мережі – від найпростіших до найскладніших пристроїв. При цьому така система розглядає мережу як єдине ціле, а не як розрізнений набір окремих пристроїв.

Сумісність. Сумісність означає, що мережа здатна містити в собі найрізноманітніше програмне і апаратне забезпечення, тобто в ній можуть співіснувати різні операційні системи, що підтримують різні стеки комунікаційних протоколів, і працювати апаратні засоби і додатки від різних виробників. Мережа, що складається з різнотипних елементів, називається неоднорідною або гетерогенною. Якщо гетерогенна мережа працює без проблем, то вона є інтегрованою. Основний шлях побудови інтегрованих мереж – використання модулів, виконаних відповідно до відкритих стандартів і специфікацій.

Контрольні запитання

1. Що називається комп'ютерною мережею?
2. Які ознаки використовуються для класифікації мереж?
3. Як класифікуються мережі за територіальною ознакою?
4. Що таке топологія комп'ютерної мережі?
5. Які топології використовуються в комп'ютерних мережах?

2 ОСНОВНІ МОДЕЛІ ВЗАЄМОЗВ'ЯЗКУ ВІДКРИТИХ СИСТЕМ. СТАНДАРТИ ГРУПИ. *IEEE* 802. СТАНДАРТНІ СТЕКИ КОМУНІКАЦІЙНИХ ПРОТОКОЛІВ

Міжнародна організація зі стандартизації (МОС, *International Organization for Standardization, ISO*) запропонувала в 1978 р. еталонну **модель взаємодії відкритих систем** (ВВС, *Open System Interconnect, OSI*). На основі цієї моделі був розроблений стек протоколів, що не знайшов широкого розповсюдження, хоча він і був прийнятий як національний стандарт урядом США ще в 1990 році (проект *GOSIP*). Проте модель *OSI* є головною методологічною основою для аналізу і розробки мереж.

Стандартом де-факто для глобальних мереж у цей час є стек протоколів *TCP/IP*, розроблений у середині 70-х років за замовленням Міністерства оборони США. Пізніше була розроблена і модель *TCP/IP*.

У локальних мережах, поряд з *TCP/IP*, застосовуються стеки *IPX/SPX, NetBIOS/SMB, XNS, DECnet* та інші.

Розходження між моделями *OSI* і *TCP/IP* пов'язані з різними цілями і методологією розробки протоколів і послуг. Розробка моделі *OSI* була спрямована на досить амбіційну мету – встановлення механізмів для розподіленої обробки даних в апаратно- і програмно-різнорідних комп'ютерних середовищах. Мета розробки протоколів *TCP/IP* була набагато скромнішою і прагматичнішою: встановлення механізмів для з'єднання мереж і надання користувачам цих мереж набору базових комунікаційних послуг.

Розробкою протоколів *OSI* займалася велика міжнародна організація – МОС. Таким організаціям властиве вповільнене функціонування. Робота над стандартами *OSI* показала як недостатню мобільність таких організацій перед інтенсивністю технологічного розвитку в даній гадузі, так і складності із встановленням балансу між суперечливими інтересами багатьох учасників роботи.

Розробка протоколів *TCP/IP* відбувалася в середовищі, яке орієнтоване на практичне застосування. У центрі уваги були конкретні проблеми, що стосуються зв'язків мереж і обслуговування користувачів. Розробку *TCP/IP* починали заради рішення проблем мережі *ARPANET*, яка була пов'язана з різким ростом кількості підключених комп'ютерів і, відповідно, трафіку, ними виробленого. Протокол *IP* повинен був надати засоби розподілу єдиної мережі, якою була *ARPANET*, на множину підмереж, що ізолюють внутрішній трафік один від одного. Потрібно було створити мережу мереж замість мережі комп'ютерів.

2.1 Еталонна модель *OSI*

При розробці моделі *OSI* виділення рівнів базувалося на таких принципах:

- кожний рівень повинен виконувати окрему функцію;

- потік інформації між рівнями повинен бути мінімізований;
- функції рівнів повинні бути зручні для визначення міжнародних стандартів;
- кількість рівнів повинна бути достатньою для поділу функцій, але не надлишковою.

Модель *OSI* визначає сім рівнів: фізичний, канальний, мережний, транспортний, сеансовий, подання даних, прикладний.

Всі мережні функції в моделі розділені на 7 рівнів (рис. 2.1). При цьому вище розташовані рівні виконують більш складні, глобальні завдання, для чого використовують у своїх цілях нижче розташовані рівні, а також керують ними. Мета нижче розташованого рівня – надання послуг вище розташованому рівню, причому нижче розташованому рівню не важливі деталі виконання цих послуг. Нижче розташовані рівні виконують більш прості, більше конкретні функції. В ідеалі кожний рівень взаємодіє тільки з тими рівнями, які розташовані поруч із ним (вище його і нижче його). Верхній рівень відповідає прикладному завданню (додатку, який працює в цей момент), нижній – безпосередній передачі сигналів каналами зв'язку.

7. Прикладний
6. Подання даних
5. Сеансів
4. Транспортний
3. Мережний
2. Канальний
1. Фізичний

Рисунок 2.1 – Рівні моделі *OSI*

Перераховані функції (показані на рис 2.1 рівні) реалізуються кожним абонентом мережі. При цьому кожний рівень на одному абоненті працює так, начебто він має прямий зв'язок з відповідним рівнем іншого абонента, тобто між однойменними рівнями абонентів мережі існує віртуальний зв'язок. Реальний же зв'язок абоненти однієї мережі мають тільки на самому нижньому, першому фізичному рівні. У передавальному абоненті інформація проходить всі рівні, починаючи з верхнього і закінчуючи нижнім. У приймальному абоненті отримана інформація здійснює шлях назад: від нижнього рівня до верхнього (рис. 2.2).

Модель *OSI* описує тільки системні засоби взаємодії, реалізовані операційною системою, системними утилітами, системними апаратними засобами. Модель не включає засобу взаємодії додатків кінцевих користувачів. Свої власні протоколи взаємодії додатка реалізують, звертаючись до системних засобів. Тому необхідно розрізняти рівень взаємодії додатків і прикладний рівень.

Рисунок 2.2 – Шлях інформації від абонента до абонента

Варто також мати на увазі, що додаток може взяти на себе функції деяких верхніх рівнів моделі *OSI*. Наприклад, деякі СКБД мають убудовані засоби видаленого доступу до файлів. У цьому випадку додаток, виконуючи доступ до видалених ресурсів, не використовує системну файлову службу; він обходить верхні рівні моделі *OSI* і звертається прямо до системних засобів, що відповідають за транспортування повідомлень мережею, які розташовані на нижніх рівнях моделі *OSI*.

2.2 Рівні моделі *OSI*

2.2.1 *Фізичний рівень.*

Фізичний рівень (*Physical layer*) має справи з передачею бітів фізичними каналами зв'язку, такими, як коаксіальний кабель, кручена пара, оптоволоконний кабель або цифровий територіальний канал. До цього рівня мають відношення характеристики фізичних середовищ передачі даних, такі, як смуга пропускання, перешкодозахищеність, хвильовий опір та інші. На цьому ж рівні визначаються характеристики електричних сигналів, що передають дискретну інформацію, таку, як крутість фронтів імпульсів, рівні напруги або струму переданого сигналу, тип кодування, швидкість передачі сигналів. Крім того, тут стандартизуються типи рознімань і призначення кожного контакту.

Функції фізичного рівня:

- передача бітів фізичними каналами;

- формування електричних сигналів;
- кодування інформації;
- синхронізація;
- модуляція.

Реалізується апаратно.

Функції фізичного рівня реалізуються у всіх пристроях, підключених до мережі. З боку комп'ютера функції фізичного рівня виконуються мережним адаптером або послідовним портом.

Прикладом протоколу фізичного рівня може служити специфікація 100*Base-TX* технології *Ethernet*, що визначає за середовище передачі даних неекрановану кручену пару категорії 5 із хвильовим опором 100 Ом, рознімання *RJ*-45, максимальну довжину фізичного сегмента 100 метрів, а також деякі інші характеристики середовища і електричних сигналів.

2.2.2 *Канальний рівень.*

На **фізичному рівні** просто пересилаються **біти**. При цьому не враховується, що в тих мережах, у яких лінії зв'язку використовуються (розділяються) поперемінно декількома парами взаємодіючих комп'ютерів, фізичне середовище передачі може бути зайняте.

Тому **одним із завдань** канального рівня (*Data Link layer*) є перевірка доступності середовища передачі. **Інше завдання** канального рівня – реалізація механізмів виявлення і корекції помилок. Для цього на канальному рівні біти групуються в набори, які називаються ***кадрами*** (***frames***).

Канальний рівень забезпечує коректність передачі кожного кадру, поміщаючи спеціальну послідовність біт на початку і в кінці кожного кадра, для його виділення. Також він обчислює контрольну суму, обробляючи всі байти кадру певним способом, і додає контрольну суму до кадру. Канальний рівень може не тільки виявляти помилки, але і виправляти їх за рахунок повторної передачі ушкоджених кадрів. Необхідно відзначити, що функція виправлення помилок для канального рівня не є обов'язковою, тому в деяких протоколах цього рівня вона відсутня, наприклад в *Ethernet* і *Frame Relay*. Реалізується апаратно.

2.2.3 *Мережний рівень.*

Мережний рівень (*Network layer*) служить для утворення єдиної транспортної системи, що поєднує кілька мереж, причому ці мережі можуть використовувати зовсім різні принципи передачі повідомлень між кінцевими вузлами і мати довільну структуру зв'язків. Функції мережного рівня досить різноманітні.

На мережному рівні сам термін ***мережа*** наділяють специфічним значенням. У цьому випадку під мережею розуміють сукупність комп'ютерів, з'єднаних між собою відповідно до однієї зі стандартних

типових топологій, яка використовує для передачі даних один із протоколів канального рівня, характерний для цієї топології.

Повідомлення мережного рівня прийнято називати ***пакетами (packets).*** При організації доставки пакетів на мережному рівні використовується поняття «номер мережі». У цьому випадку адреса одержувача складається зі старшої частини (номера мережі) і молодшої (номера вузла в цій мережі). Всі вузли однієї мережі повинні мати ту саму старшу частину адреси, тому терміну «мережа» на мережному рівні можна дати і інше, більш формальне визначення: мережа –це сукупність вузлів, мережна адреса яких містить той самий номер мережі.

На мережному рівні визначаються два види протоколів:

1. Мережні протоколи ***(routed protocols)*** – реалізують просування пакетів через мережу. Саме ці протоколи мають на увазі, коли говорять про протоколи мережного рівня. Однак часто до мережного рівня відносять і інший вид протоколів, які називаються протоколами обміну маршрутною інформацією або просто *протоколами маршрутизації (routing protocols)*.

2. Протоколи розв'язання адрес – *Address Resolution Protocol, ARP*, які відповідають за відображення адреси вузла, використовуваного на мережному рівні, у локальну адресу мережі.

3. Прикладами протоколів мережного рівня є протокол міжмережної взаємодії *IP* стека *TCP/IP* і протокол міжмережевого обміну пакетами *IPX* стека *Novell*.

2.2.4 *Транспортний рівень*.

Транспортний рівень *(Transport layer)* забезпечує додаткам або верхнім рівням стека (прикладному і сеансовому) передачу даних з тим ступенем надійності, що їм необхідно. Модель *OSI* визначає п'ять класів сервісу, які надаються транспортним рівнем. Ці види сервісу відрізняються якістю надаваних послуг: терміновістю, можливістю відновлення перерваного зв'язку, наявністю засобів націлити декількох з'єднань між різними прикладними протоколами через загальний транспортний протокол, а головне – здатністю до виявлення і виправлення помилок передачі, таких як перекручування, втрата і дублювання пакетів.

Основні завдання транспортного рівня:

1) розбивка повідомлення сеансового рівня на пакети, їхня нумерація;
2) буферизація прийнятих пакетів;
3) упорядкування пакетів, які надходять;
4) адресація прикладних процесів;
5) керування потоком.

Як правило, всі протоколи, починаючи із транспортного рівня і вище, реалізуються програмними засобами кінцевих вузлів мережі –

компонентами їх мережних операційних систем. Як приклад транспортних протоколів можна навести протоколи *TCP* і *UDP* стека *TCP/IP* і протокол *SPX* стека *Novell.*

2.2.5 *Сеансовий рівень.*

Сеансовий рівень *(Session layer)* забезпечує керування діалогом: фіксує, яка зі сторін є активною в даний момент, надає засоби синхронізації. Останні дозволяють вставляти контрольні точки в довгі передачі, щоб у випадку відмови можна було повернутися назад до останньої контрольної точки, а не починати все спочатку. На практиці деякі додатки використовують сеансовий рівень, і він рідко реалізується у вигляді окремих протоколів, хоча функції цього рівня часто поєднують із функціями прикладного рівня і реалізують в одному протоколі.

Основні завдання сеансового рівня:

1) встановлення способу обміну повідомленнями (дуплексний або напівдуплексний);
2) синхронізація обміну повідомленнями;
3) організація "контрольних точок" діалогу.

2.2.6 *Представницький рівень.*

Представницький рівень (*Presentation layer*) має справу з формою подання переданої мережею інформації, не міняючи при цьому її змісту. За рахунок рівня подання інформація, передана прикладним рівнем однієї системи, завжди зрозуміла прикладному рівню іншої системи. За допомогою засобів даного рівня протоколи прикладних рівнів можуть перебороти синтаксичні розходження в поданні даних або ж розходження в кодах символів, наприклад кодів *ASCII* і *EBCDIC*. На цьому рівні можливо виконувати шифрування і дешифрування даних, завдяки якому таємність обміну даними забезпечується відразу для всіх прикладних служб. Прикладом такого протоколу є протокол *Secure Socket Layer (SSL),* що забезпечує секретний обмін повідомленнями для протоколів прикладного рівня стека *TCP/IP*.

Основні завдання представницького рівня:

1) перетворення даних із зовнішнього формату у внутрішній;
2) шифрування і розшифровка даних.

2.2.7 *Прикладний рівень.*

Прикладний рівень (*Application layer*) – це в дійсності просто набір різноманітних протоколів, за допомогою яких користувачі мережі одержують доступ до ресурсів, що розділяються, таких, як файли, принтери або гіпертекстові *Web*-сторінки, а також організують свою спільну роботу, наприклад, за допомогою протоколу електронної пошти. Одиниця даних, якою оперує прикладний рівень, називається повідомленням (*message*).

Основні завдання прикладного рівня:

1) ідентифікація, перевірка прав доступу;

2) принт- і файл-сервіс, пошта, вилучений доступ і т.д.

Крім моделі *OSI*, існує також модель ***IEEE Project* 802**, прийнята в лютому 1980 року (звідси і число 802 у назві), яку можна розглядати як модифікацію, розвиток, уточнення моделі *OSI*. Стандарти, обумовлені цією моделлю (так звані 802-специфікації), діляться на дванадцять категорій, кожній з яких присвоєний свій номер.

- 802.1 – об'єднання мереж.
- 802.2M – керування логічним зв'язком.
- 802.3 – локальна мережа з методом доступу *CSMA/CD* і топологією «шина» (*Ethernet*).
- 802.4 – локальна мережа з топологією «шина» і маркерним доступом.
- 802.5 – локальна мережа з топологією «кільце» і маркерним доступом.
- 802.6 – міська мережа (*Metropolitan Area Network, MAN*).
- 802.7 – широкомовна технологія.
- 802.8 – оптоволоконна технологія.
- 802.9 – інтегровані мережі з можливістю передачі мови і даних.
- 802.10 – безпека мереж.
- 802.11 – бездротова мережа.
- 802.12 – локальна мережа із централізованим керуванням, доступом за пріоритетними запитами і топологією «дерево» (100*VG-AnyLAN*).

Стандарти 802.3, 802.4, 802.5, 802.12 прямо ставляться до підрівня *MAC* другого (канального) рівня еталонної моделі *OSI*. Інші 802-специфікації вирішують загальні питання мереж.

2.3 Стандартні стеки комунікаційних протоколів

Протокол – це набір правил і процедур, що визначають послідовність та формат повідомлень, якими обмінюються мережні компоненти, що належать одному й тому ж рівню, але розташовані на різних вузлах. Природно, всі комп'ютери, що беруть участь в обміні, повинні працювати за тими самими протоколами, щоб по завершенні передачі вся інформація відновлювалася в первісному вигляді.

Існує кілька стандартних наборів (або, як їх ще називають, стеків) протоколів, що набули зараз найбільш широке поширення:

- набір протоколів *ISO/OSI*;

- *IBM System Network Architecture* (*SNA*);

- *Digital DECnet*;

- *Novell NetWare*;

- *Apple AppleTalk*;

- набір протоколів глобальної мережі *Internet*, *TCP/IP*.

2.3.1 *Стек OSI.*

Варто чітко розрізняти модель *OSI* і стек *OSI*. У той час як модель *OSI* є концептуальною схемою взаємодії відкритих систем, стек *OSI* являє собою набір цілком конкретних специфікацій протоколів. На відміну від інших стеків протоколів стек *OSI* повністю відповідає моделі *OSI*, він включає специфікації протоколів для всіх семи рівнів взаємодії, певних у цій моделі.

На нижніх рівнях стек *OSI* підтримує *Ethernet, Token Ring, FDDI,* протоколи глобальних мереж, *X.*25 і *ISDN*, тобто використовує розроблені протоколи нижніх рівнів, як і всі інші стеки. Протоколи мережного, транспортного і сеансового рівнів стека *OSI* специфіковані і реалізовані різними виробниками, але поширені поки що мало. Найбільш популярними протоколами стека *OSI* є прикладні протоколи. До них належать: протокол передачі файлів *FTAM*, протокол емуляції термінала *VTP*, протоколи довідкової служби *X.*500, електронної пошти *X.*400 і ряд інших.

2.3.2 *Стек TCP/IP.*

Стек *TCP/IP* був розроблений з ініціативи Міністерства оборони США більше 20 років тому для зв'язку експериментальної мережі *ARPAnet* з іншими мережами як набір загальних протоколів для різнорідного обчислювального середовища.

Стек *TCP/IP* на нижньому рівні підтримує всі популярні стандарти фізичного і канального рівнів: для локальних мереж (*Ethernet, Token Ring, FDDI)*, для глобальних (протоколи роботи на аналогових і виділених лініях *SLIP, PPP*, протоколи територіальних мереж *X.*25 і *ISDN)*.

Основними протоколами стека, що дали йому назву, є протоколи *IP* і *TCP*. Ці протоколи в термінології моделі *OSI* належать до мережного і транспортного рівнів відповідно. *IP* забезпечує просування пакета по складній мережі, а *TCP* гарантує надійність його доставки.

За довгі роки використання в мережах різних країн і організацій стек *TCP/IP* увібрав у себе велику кількість протоколів прикладного рівня. До них належать такі популярні протоколи, як протокол пересилання файлів *FTP*, протокол емуляції термінала *Telnet*, поштовий протокол *SMTP*, що використовується в електронній пошті мережі *Internet*, гіпертекстові сервіси служби *WWW* і багато інших.

Сьогодні стек *TCP/IP* є одним із найпоширеніших стеків транспортних протоколів обчислювальних мереж. Дійсно, тільки в мережі *Internet* об'єднано близько 10 мільйонів комп'ютерів по усьому світі, які взаємодіють один з одним за допомогою стека протоколів *TCP/IP*.

2.3.3 *Стек IPX/SPX.*

Цей стек є оригінальним стеком протоколів фірми *Novell*, розробленим для мережної операційної системи *NetWare* ще на початку 80-х років. Протоколи мережного і сеансового рівнів *Internetwork Packet Exchange* (*IPX*) і *Sequenced Packet Exchange* (*SPX*), які дали назву стеку, є прямою адаптацією протоколів *XNS* фірми *Xerox*, розповсюджених набагато менше, чим стек *IPX/SPX*. Популярність стека *IPX/SPX* безпосередньо пов'язана з операційною системою *Novell NetWare*.

Стек *IPX/SPX* довгий час обмежувався поширеністю його тільки мережами *NetWare*, тому що він є власністю фірми *Novell* і на його реалізацію потрібно одержувати ліцензію (тобто відкриті специфікації не підтримувалися). Зараз стек *IPX/SPX* реалізований не тільки в *NetWare*, але і у декількох інших популярних мережних ОС, наприклад *SCO UNIX, Sun Solaris, Microsoft Windows*.

2.3.4 *Стек NetBIOS/SMB.*

Цей стек широко використовується в продуктах компаній *IBM* і *Microsoft*. На фізичному і канальному рівнях цього стека використовуються усі найпоширеніші протоколи *Ethernet, Token Ring, FDDI* і інші. На верхніх рівнях працюють протоколи *NetBEUI* і *SMB.*

Протокол *NetBIOS* (*Network Basic Input/Output System*) з'явився в 1984 році як мережне розширення стандартних функцій базової системи введення/виведення (*BIOS*) *IBM PC* для мережної програми *PC Network* фірми *IBM*. Надалі цей протокол був замінений так званим протоколом розширеного користувальницького інтерфейсу *NetBEUI - NetBIOS Extended User Interface*. Для забезпечення сумісності додатків як інтерфейс до протоколу *NetBEUI* був збережений інтерфейс *NetBIOS*. Протокол *NetBEUI* розроблявся як ефективний протокол, що споживає небагато ресурсів і призначений для мереж, які нараховують не більше 200 робочих станцій. Цей протокол містить багато корисних мережних функцій, які можливо віднести до мережного, транспортного і сеансового рівнів моделі *OSI*, однак з його допомогою неможлива маршрутизація пакетів. Це обмежує застосування протоколу *NetBEUI* локальними мережами, не розділеними на підмережі, і унеможливлює його використання в складних мережах. Деякі обмеження *NetBEUI* знімаються реалізацією цього протоколу *NBF* (*NetBEUI Frame*).

Протокол *SMB* (*Server Message Block*) виконує функції сеансового, представницького і прикладного рівнів. На основі *SMB* реалізується файлова служба, а також служби друку і передачі повідомлень між додатками.

Стеки протоколів *SNA* фірми *IBM, DECnet* корпорації *Digital Equipment* і *AppleTalk/AFP* фірми *Apple* застосовуються в основному в операційних системах і мережному устаткуванні цих фірм.

У табл. 2.1 показані рівні моделі *OSI,* та відповідність деяких, найбільш популярних, протоколів цим рівням. Часто ця відповідність

досить умовна, тому що модель *OSI* – це тільки керівництво до дії, причому досить загальне, а конкретні протоколи розроблялися для розв'язування специфічних завдань, причому багато хто з них з'явилися до розробки моделі *OSI*. У більшості випадків розроблювачі стеків віддавали перевагу швидкості роботи мережі на шкоду модульності. Жоден стек, крім стека *OSI*, не розбитий на сім рівнів. Найчастіше в стеці явно виділяються 3-4 рівня: рівень мережних адаптерів, у якому реалізуються протоколи фізичного і канального рівнів, мережний рівень, транспортний рівень і рівень служб, що вбирає в себе функції сеансового, представницького і прикладного рівнів.

Таблиця 2.1 – Відповідність популярних стеків протоколів моделі *OSI*

<table>
<tr><th>Модель OSI</th><th>Стек
IBM /
Microsoft</th><th>Стек
TCP/IP</th><th>Стек
Novell
NetWare</th><th>Стек
OSI</th></tr>
<tr><td>7. Прикладний</td><td rowspan="2">SMB</td><td rowspan="2">Telnet, FTP, SNMP, SMTP, WWW</td><td rowspan="3">NCP, SAP</td><td>X.400, X.500, FTAM</td></tr>
<tr><td>6. Подання даних</td><td>Представницький протокол OSI</td></tr>
<tr><td>5. Сеансовий</td><td rowspan="2">NetBIOS</td><td rowspan="2">TCP, UDP</td><td>Сеансовий протокол OSI</td></tr>
<tr><td>4. Транспортний</td><td>SPX</td><td>Транспортний протокол OSI</td></tr>
<tr><td>3. Мережний</td><td></td><td>IP, RIP, OSPF</td><td>IPX, RIP, NLSP</td><td>ES-ES, IS-IS</td></tr>
<tr><td>2. Канальний</td><td colspan="4">Ethernet, Token Ring, FDDI, SLIP, X.25, ATM, PPP, LAP-B, LAP-D</td></tr>
<tr><td>1. Фізичний</td><td colspan="4">коаксіальний кабель, "кручена пара", оптоволокно, радіохвилі</td></tr>
</table>

Контрольні запитання

1. Які рівні існують в еталонній моделі взаємодії відкритих систем?
2. Що називається протоколом?
3. Що таке стек протоколів?
4. Які основні стеки комунікаційних протоколів існують на сьогоднішній день?
5. Скільки рівнів виділяється у стеку *TCP/IP?*

3 ТЕОРЕТИЧНІ ОСНОВИ ПЕРЕДАЧІ ДАНИХ. МЕТОДИ ДОСТУПУ І ЇХНЯ КЛАСИФІКАЦІЯ. СПОСОБИ КОМУТАЦІЇ. АНАЛОГОВІ КАНАЛИ ПЕРЕДАЧІ ДАНИХ

Будь-який сигнал можна розглядати як функцію часу або як функцію частоти. У першому випадку ця функція показує, як міняються з часом параметри сигналу, наприклад, напруга або струм. Якщо ця функція має безперервний характер, то говорять про **безперервний** сигнал. Якщо ця функція має дискретний вигляд, то говорять про **дискретний** сигнал.

Частотне подання функції основане на тому факті, що будь-яка функція може бути подана у вигляді ряду Фур'є (3.1):

$$g(t) = \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft), \qquad (3.1)$$

де $f = 1/T$ –- частота, a_n ,b_n – амплітуди n-ї гармоніки.

Характеристику каналу, яка визначає спектр частот фізичного середовища лінії зв'язку каналу, при якому сигнал пропускається без істотного зниження потужності, називають **смугою пропускання**.

Максимальну швидкість, з якою канал здатний передавати дані, називають **пропускною здатністю каналу або бітовою швидкістю.**

У 1924 Найквист (3.2) відкрив взаємозв'язок між пропускною здатністю каналу і шириною його смуги пропущення.

3.1 Теорема Найквиста

$$V_{max\ data\ rate} = 2H \log_2 M \text{ (біт/с)}, \qquad (3.2)$$

де $V_{max\ data\ rate}$ – максимальна швидкість передачі; H – ширина смуги пропускання каналу, виражена в Гц; M – кількість рівнів сигналу, що використовуються при передачі. Наприклад, із цієї формули випливає, що канал зі смугою 3 КГц не може передавати дворівневі сигнали швидше 6000 біт/с.

Ця теорема також показує, що, наприклад, безглуздо сканувати лінію частіше, ніж подвоєна ширина смуги пропускання. Дійсно, всі частоти, вище цієї, відсутні в сигналі, а тому вся інформація, необхідна для відновлення сигналу буде зібрана при такому скануванні.

Однак теорема Найквиста не враховує шум у каналі, що вимірюється як відношення потужності корисного сигналу до потужності шуму: *S/N*. Ця величина виміряється в децибелах: $10*\log_{10}(S/N)$ *dB*. Наприклад, якщо відношення *S/N* дорівнює 10, то говорять про шум в 10*dB*, якщо відношення дорівнює 100, то – 20 *dB*.

На випадок каналу із шумом є теорема Шенона (3.3), за якою максимальна швидкість передачі даних по каналу із шумом дорівнює:

$$H \log_2 (1+S/N) \text{ (біт/с)}, \qquad (3.3)$$

де *S/N* – співвідношення сигнал-шум у каналі.

Тут вже не важлива кількість рівнів у сигналі. Ця формула встановлює теоретичну межу, яка рідко досягається на практиці. Наприклад, каналом зі смугою пропущення в 3000 Гц і рівнем шуму 30 *dB* (це характеристики телефонної лінії) не можна передати дані швидше, ніж зі швидкістю 30 000 біт/с.

3.2 Методи доступу і їхня класифікація

Метод доступу (*access method*) – це набір правил, що регламентують спосіб одержання в користування (“захоплення”) середовища передачі. Метод доступу визначає, яким образом вузли одержують можливість передавати дані.

Виділяють наступні класи методів доступу:

- селективні методи;
- змагальні методи (методи випадкового доступу);
- методи, основані на резервуванні часу;
- кільцеві методи.

Усі методи доступу, крім змагальних, утворюють групу методів детермінованого доступу. При використанні **селективних методів** для того, щоб вузол міг передавати дані, він повинен одержати дозвіл. Метод називається **опитуванням** (*polling*), якщо дозволи передаються всім вузлам по черзі спеціальним мережним устаткуванням. Метод називається **передачею маркера** (*token passing*), якщо кожен вузол по завершенні передачі передає дозвіл наступному.

Методи **випадкового доступу** (*random access methods*) основані на “змаганні” вузлів за одержання доступу до середовища передачі. Випадковий доступ може бути реалізований різними способами: базовим асинхронним, з тактовою синхронізацією моментів передачі кадрів, із прослуховуванням каналу перед початком передачі (“слухай, перш ніж говорити”), із прослуховуванням каналу під час передачі (“слухай, поки говориш”). Можуть бути використані одночасно кілька способів з перерахованих.

Методи, основані на **резервуванні часу**, зводяться до виділення інтервалів часу (слотів), які розподіляються між вузлами. Вузол одержує канал у своє розпорядження на всю тривалість виділених йому слотів. Існують варіанти методів, що враховують пріоритети – вузли з більш високим пріоритетом одержують більшу кількість слотів.

Кільцеві методи використовуються у локальних КМ із кільцевою топологією. Кільцевий метод вставки регістрів полягає в підключенні

паралельно до кільця одного або декількох буферних регістрів. Дані для передачі записуються в регістр, після чого вузол очікує міжкадрового проміжку. Потім вміст регістра передається в канал. Якщо під час передачі надходить кадр, він записується в буфер і передається після своїх даних.

Розрізняють клієнт-серверні і однорангові методи доступу.

Клієнт-серверні методи доступу припускають наявність у мережі центрального вузла, який керує всіма іншими. Такі методи розпадаються на дві групи: з опитуванням і без опитування.

Серед **методів доступу з опитуванням** найпоширенішим є "опитування із зупинкою і очікуванням " і "безперервний автоматичний запит на повторення" (ARQ). Кожного разу первинний вузол послідовно передає вузлам дозвіл на передачу даних. Якщо вузол має дані для передачі, він видає їх у середовище передачі, якщо немає, то видає або короткий пакет даних типу "даних немає", або просто нічого не передає.

При використанні **однорангових** методів доступу всі вузли рівноправні. Мультиплексна передача з тимчасовим поділом – це найбільш проста однорангова система без пріоритетів, що використовує твердий розклад роботи вузлів. Кожному вузлу виділяється інтервал часу, протягом якого вузол може передавати дані, причому інтервали розподіляються порівну між усіма вузлами.

3.3 Метод доступу з контролем несучої і визначенням колізій

Множинний доступ з контролем несучої і визначенням колізій (*CSMA/CD, Carrier Sense Multiple Access/Collision Detect*) – найпоширеніший метод випадкового доступу, який застосовують в локальних мережах. Всі вузли мережі постійно прослуховують канал (контроль несучої). Якщо вузол має дані для передачі, він чекає тиші в каналі і починає передачу. При цьому може виявитися так, що інший вузол теж виявив, що канал вільний, і теж почав передачу. Така ситуація називається **колізією**. Оскільки всі вузли, передаючи дані, продовжують прослуховувати канал, вони можуть виявити накладення сигналів від різних джерел. При виявленні колізії вузли, що передають дані, видають у канал спеціальну послідовність бітів – "затор", що служить для оповіщення інших вузлів про колізію. Потім всі передавальні вузли припиняють передачу і планують здійснити її пізніше. Величина паузи вибирається випадково.

3.4 Маркерні методи доступу

Метод передачі маркера належить до селективних детермінованих однорангових методів доступу. Мережі з шинною топологією, які використовують передачу маркера, називаються мережами типу "маркерна шина" (*token bus*), а кільцеві мережі – мережами типу "маркерне кільце" (*token ring*).

У мережах типу **“маркерна шина”** маркер являє собою кадр, який містить поле адреси, куди записується адреса вузла. Це надає право доступу до середовища передачі. Після передачі кадру в маркер записується адреса наступного вузла і маркер видається у канал.

Мережі типу **“маркерне кільце”,** будучи мережами з кільцевою топологією, мають послідовну конфігурацію: кожна пара вузлів зв’язана окремим каналом, а для функціонування мережі необхідне функціонування всіх вузлів. У таких мережах маркер не містить адреси вузла, якому дозволена передача, а містить тільки поле зайнятості, що може мати одне із двох значень: “зайнятий” і “вільний”. Коли вузол, що має дані для передачі, одержує вільний маркер, він міняє стан маркера на “зайнятий”, а потім передає в канал маркер і свій кадр даних. Станція-одержувач, розпізнавши свою адресу в кадрі даних, зчитує призначені їй дані, але не міняє стану маркера. Змінює стан маркера на “вільний” (після повного оберту маркера з кадром даних по кільцю) той вузол, що його зайняв. Кадр даних при цьому видаляється з кільця. Вузол не може повторно використовувати маркер для передачі іншого кадру даних, а повинен передати вільний маркер далі по кільцю і дочекатися його одержання після одного або декількох обертів.

Рівнорангові пріоритетні системи включають пріоритетні слотові системи, системи з контролем несучої без колізій і системи з передачею маркера із пріоритетами.

Пріоритетні слотові системи подібні до систем з мультиплексною передачею з тимчасовим поділом, але видача слотів відбувається з урахуванням пріоритетів вузлів. Критеріями для встановлення пріоритетів можуть бути: попереднє володіння слотом, час відповіді, обсяг переданих даних та ін.

Системи з контролем несучої без колізій (*CSMA/CA, Carrier Sense Multiple Access/Collision Avoidance*) відрізняються від систем з виявленням колізій наявністю у вузлів таймерів, що визначають безпечні моменти передачі. Тривалість таймерів встановлюється залежно від пріоритетів вузлів: станції з більш високим пріоритетом мають меншу тривалість таймера.

Пріоритетні системи з передачею маркера визначають пріоритети вузлів таким чином, що чим менше номер вузла, тим вище його пріоритет. Маркер при цьому містить поле резервування, у яке вузол, що збирається передавати дані, записує своє значення пріоритету. Якщо в кільці зустрінеться вузол з більш високим пріоритетом, що теж має дані для передачі, цей вузол запише своє значення пріоритету в поле резервування, чим перекриє попередню заявку (зберігши старе значення поля резервування у своїй пам’яті). Якщо маркер, що надійшов на вузол, містить у полі резервування значення пріоритету даного вузла, то цей вузол може передавати дані. Після оберту маркера по кільцю і його звільнення вузол, що передавав, повинен відновити в маркері значення поля резервування, збережене в пам’яті.

3.5 Способи комутації

Комутація є необхідним елементом зв'язку вузлів між собою, що дозволяє скоротити кількість необхідних ліній зв'язку і підвищити завантаження каналів зв'язку. Практично неможливо надати кожній парі вузлів виділену лінію зв'язку, тому в мережах завжди застосовується той або інший спосіб комутації абонентів, що використовує існуючі лінії зв'язку для передачі даних різних вузлів.

Мережею, що комутується, називається мережа, у якій зв'язок між вузлами встановлюється тільки за запитом.

Абоненти з'єднуються з комутаторами виділеними (індивідуальними) лініями зв'язку. Лінії зв'язку, що з'єднують комутатори, використовуються абонентами спільно.

Комутація може здійснюватися у двох режимах: динамічно і статично. У першому випадку комутація виконується на час сеансу зв'язку (звичайно від секунд до годин) з ініціативи одного з вузлів, а по закінченні сеансу зв'язок розривається. У другому випадку комутація виконується обслуговуючим персоналом мережі на значно більш тривалий період часу (кілька місяців або років) і не може бути змінена з ініціативи користувачів. Такі канали називаються **виділеними** (*dedicated*) або **орендованими** (*leased*).

Дві групи способів комутації: **комутація каналів** (*circuit switching*) і **комутація із проміжним зберіганням** (*store-and-forward*). Друга група складається із двох способів: **комутації повідомлень** (*message switching*) і **комутації пакетів** (*packet switching*).

При **комутації каналів** між вузлами, яким необхідно встановити зв'язок один з одним, забезпечується організація безперервного складеного каналу, який складається з послідовно з'єднаних окремих каналів між вузлами. Окремі канали з'єднуються між собою комутуючим устаткуванням (комутаторами). Перед передачею даних необхідно виконати процедуру встановлення з'єднання, у процесі якої створюється складений канал.

Під **комутацією повідомлень** розуміють передачу єдиного блоку даних між вузлами мережі з тимчасовою буферизацією цього блоку кожним із транзитних вузлів. Повідомленням може бути текстовий файл, файл із графічним зображенням, електронний лист. Повідомлення має довільний розмір, обумовлений винятково його змістом, а не тими або іншими технологічними міркуваннями.

При **комутації пакетів** всі передані користувачем дані розбиваються передавальним вузлом на невеликі (до декількох кілобайт) частини – **пакети** (packet). Кожний пакет оснащується заголовком, у якому

вказується, як мінімум, адреса вузла-одержувача і номер пакета. Передача пакетів мережею відбувається незалежно один від одного. Комутатори такої мережі мають внутрішню буферну пам'ять для тимчасового зберігання пакетів, що дозволяє згладжувати пульсації трафіку на лініях зв'язку між комутаторами. Пакети іноді називають **дейтаграмами (***datagram***)**, а режим індивідуальної комутації пакетів – **дейтаграмним** режимом.

Мережа з комутацією пакетів уповільнює процес взаємодії кожної конкретної пари вузлів, оскільки їхні пакети можуть очікувати в комутаторах, поки передадуться інші пакети. Однак загальна ефективність (обсяг переданих даних в одиницю часу) при комутації пакетів буде вищою, ніж при комутації каналів. Це пов'язано з тим, що трафік кожного окремого абонента має пульсуючий характер, а пульсації різних абонентів, відповідно до закону більших чисел, розподіляються в часі, збільшуючи рівномірність навантаження на мережу.

3.6 Аналогові канали передачі даних. Аналогова модуляція

Під **каналом передачі даних** (КПД) розуміють сукупність середовища передачі (середовища поширення сигналу) і технічних засобів передачі між канальними інтерфейсами. Залежно від форми інформації, що може передаватися каналом, розрізняють **аналогові** і **цифрові** канали.

Аналоговий канал на вході (і відповідно, на виході) має безперервний сигнал, ті або інші характеристики якого (наприклад, амплітуда або частота) несуть передану інформацію. Цифровий канал приймає і видає дані в цифровій (дискретній, імпульсній) формі.

Оскільки мережі зв'язують цифрові комп'ютери каналом зв'язку, то необхідно передавати дискретні дані. Відповідно, при використанні аналогових сигналів необхідно деяке перетворення (кодування) переданих даних цими сигналами. Таке перетворення називається **аналоговою модуляцією** (або аналоговим кодуванням). В його основі лежить зміна однієї з характеристик синусоїдального несучого сигналу відповідно до послідовності переданих даних. Основні способи аналогової модуляції: амплітудна, частотна і фазова. Можливо також використання комбінованих методів, наприклад, поєднання амплітудної і фазової модуляцій.

При **амплітудній** (рис. 3.1, b) модуляції змінюється тільки амплітуда синусоїди несучої частоти, при передачі логічної одиниці видається синусоїда однієї амплітуди, а при передачі логічного нуля – іншої амплітуди. Цей спосіб у чистому вигляді має низьку помилкостійкість і застосовується рідко.

При **частотній** (рис. 3.1, c) модуляції змінюється тільки частота несучої – для логічної одиниці і логічного нуля вибираються синусоїди двох різних частот. Цей спосіб досить просто реалізувати, і він часто застосовується при низькошвидкісній передачі даних.

При **фазовій** (рис. 3.1, d) модуляції логічній одиниці і логічному нулю відповідають сигнали однакової амплітуди і частоти, але відрізняються за фазою (наприклад, 0^o і 180^o).

З комбінованих методів широко використовуються методи **квадратурної амплітудної модуляції** (*Quadrature Amplitude Modulation, QAM*), що поєднують амплітудну модуляцію з 4-ма рівнями амплітуди і фазову модуляцію з 8-ма значеннями зсуву фази. З 32-х можливих комбінацій амплітуди і зсуву фази для передачі даних у різних модифікаціях методу використовуються всього декілька, у той час, як усі інші комбінації є забороненими, що дозволяє поліпшити розпізнавання помилкових сигналів.

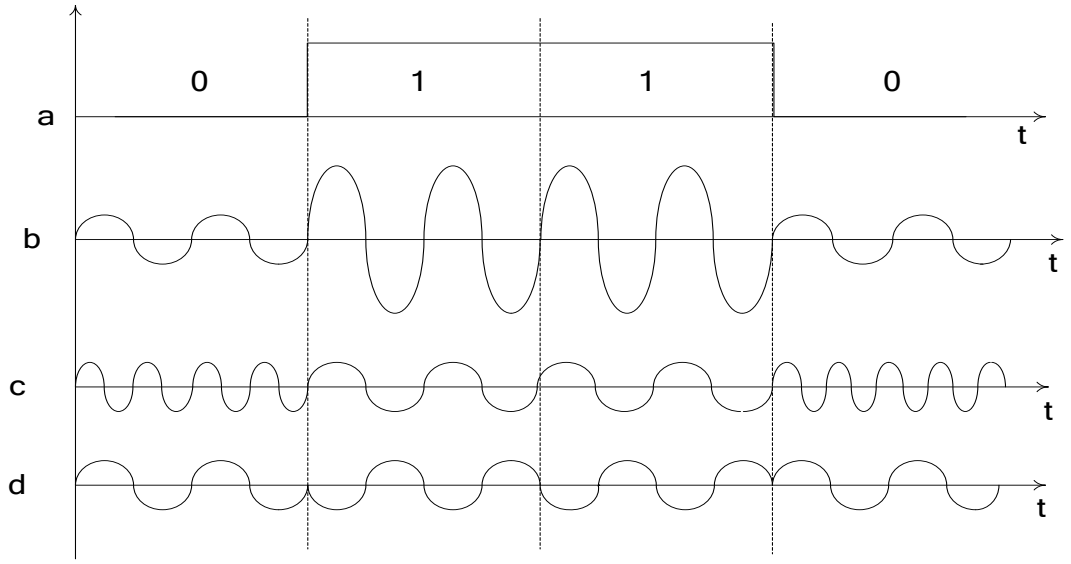

Рисунок 3.1 – Типи модуляції

3.7 Модеми

Пристрої, що виконують модуляцію і демодуляцію (відновлення з модульованого сигналу вихідних даних), називаються **модемами** (модулятор-демодулятор). Модеми класифікують за областю застосування, функціональним призначенням, типом використовуваного каналу, підтримкою протоколів модуляції, виправленням помилок і ущільненням даних, конструктивним виконанням.

За застосуванням модеми можна розділити на такі групи:

- для телефонних каналів, що комутуються;
- для виділених каналів;
- для фізичних ліній:
- вузькосмугові (*baseband*);
- короткого радіуса дії (*short range* або *line driver*);
- для цифрових систем передачі (CSU/DSU);
- для стільникових систем зв’язку;
- для радіоканалів з пакетною передачею;

- для локальних радіомереж.

Модеми **для телефонних каналів**, що комутуються, призначені для широкого кола користувачів і є найпоширенішими. Такі модеми повинні працювати в смузі пропускання 3,1 кГц у голосовому діапазоні (оскільки апаратура АТС не пропустить інші сигнали), вміти взаємодіяти з АТС – набирати номер в імпульсному або тоновому режимі, визначати сигнал "зайнято" і т.д.

Модеми **для виділених орендованих каналів** відрізняються від модемів для ліній, що комутуються, тільки в тому, що їм не потрібно взаємодіяти з апаратурою АТС для встановлення з'єднання. Вони теж повинні працювати у вузькій смузі пропускання.

Модеми **для фізичних ліній** не обмежені вузькою смугою пропускання певної АТС (при цьому діють інші обмеження смуги, пов'язані з довжиною, екрановаістю та іншими характеристиками лінії). Вузькосмугові модеми для фізичних ліній використовують методи модуляції, аналогічні застосовуваним у модемах для ліній, що комутуються, але за рахунок більш широкої смуги пропускання, можуть досягати більш високих швидкостей передачі – 128 Кбіт/с і вище.

Модеми короткого радіуса дії використовують уже не аналогову модуляцію, а цифрові сигнали. Часто використовуються різноманітні методи цифрового кодування, що виключають постійну складову із сигналу.

Модеми для цифрових систем передачі забезпечують підключення до стандартних цифрових каналів (*T1/E1, ISDN*) і підтримують функції канальних інтерфейсів.

Модеми для стільникових систем зв'язку підтримують спеціальні протоколи модуляції і корекції помилок, що дозволяють працювати при параметрах середовища передачі, які часто змінюються, і високому рівні перешкод.

Модеми для радіоканалів з пакетною передачею використовують ту саму смугу частот, у якій організується множинний доступ, наприклад, з контролем несучої. Швидкість передачі, що досягається при цьому, звичайно невисока – до 64 Кбіт/с, але відстань між станціями може становити кілька кілометрів.

Модеми для локальних радіомереж забезпечують передачу даних з високою швидкістю (до 16 Мбіт/с) на невеликі відстані (до 300 м). Для запобігання взаємному впливу декількох одночасно передавальних модемів використовуються різні способи, наприклад, псевдовипадкової перебудови робочої частоти або широкополосну передачу.

За методом передачі модеми діляться на синхронні і асинхронні. Оскільки модем зв'язаний, з одного боку, з комп'ютером, а з іншого боку – через канал з іншим модемом, можливий асинхронно-синхронний режим роботи: модем одержує дані від комп'ютера асинхронно, а передає їх іншому модему в синхронному режимі.

3.8 Протоколи, що підтримуються модемами

Усі модемні протоколи можна розділити на міжнародні і фірмові. Часто фірмовий протокол, розроблений тією або іншою компанією, реалізують і інші виробники модемів, він стає стандартом де-факто, а потім на його основі виробляється міжнародний стандарт.

Міжнародні стандарти в галузі електрозв'язку випускаються Комітетом зі стандартизації комунікацій *ITU-T* (раніше називався Міжнародним консультативним комітетом з телефонії і телеграфії, МККТТ - *Comitet Consultatif Internationale de Telegraphique et Telephonique, CCITT*) у формі рекомендацій. Рекомендації *ITU-T*, що стосуються модемів, належать до серії *V*.

Модемні протоколи можна розбити на кілька груп:

- протоколи, що визначають з'єднання модема і каналу зв'язку: *V*.2, *V*.25 та ін. ;
- протоколи, що визначають з'єднання модема з комп'ютером: *V*.10, *V*.11, *V*.14, *V*.25, *V*.25*bis*, *V*.28 та ін. ;
- протоколи модуляції: *V*.17, *V*.22, *V*.32, *V*.32*bis*, *V*.32*ter*, *V*.34, *V*.90, *HST*, *PEP*, *ZyX* та ін. ;
- протоколи корекції помилок: *MNP*1-*MNP*4, *MNP*6, *MNP*10, *V*.41, *V*.42;
- протоколи ущільнення даних: *V*.42*bis*, *MNP*5, *MNP*7;
- протоколи узгодження параметрів зв'язку: *V*.8;
- протоколи діагностики модемів: *V*.51-*V*.54, *V*.56.

3.9 Режими передачі

Режим передачі визначає спосіб комунікації між двома вузлами.

При **симплексному** (*simplex*) режимі приймач і передавач зв'язується лінією зв'язку, за якою інформація передається тільки в одному напрямку. Передавальний вузол у симплексному режимі повністю займає канал. Приклади: радіомовлення, телемовлення.

Напівдуплексний (*half duplex*) режим допускає передачу у двох напрямках, але в різні моменти часу. Два вузли зв'язуються таким каналом зв'язку, що дозволяє їм по черзі (але не одночасно) передавати інформацію. Для зміни напрямку передачі, як правило, використовується передача спеціального сигналу і одержання підтвердження.

Дуплексний або **повнодуплексний** (*duplex, full duplex*) режими дозволяє одночасно передавати інформацію у двох напрямках. У найпростішому випадку для дуплексного зв'язку використовуються дві лінії зв'язку (пряма і зворотна), але існують рішення, що дозволяють підтримувати дуплексний режим на єдиній лінії (наприклад, обидва вузли можуть одночасно передавати дані, а із прийнятого сигналу віднімати власні дані). Дуплексний режим може бути симетричним (смуга пропускання каналу однакова в обох напрямках) або асиметричним.

3.10 Асинхронна, синхронна, ізохронна і плезиохронна передача

Для послідовної передачі даних досить однієї лінії, якою можуть послідовно передаватися біти даних. Приймач повинен вміти розпізнавати, де починається і де закінчується сигнал, що відповідає кожному біту даних. Інакше кажучи, передавач і приймач повинні вміти синхронізуватися. Якщо якість синхронізації низька (за час передачі одного біта неузгодженість досягає декількох відсотків), використовується **асинхронний** (*asynchronous*) режим: виконується узгодження синхрогенераторів на початку передачі кожного байта. Як правило, передача байта починається зі спеціального старт-біта, потім ідуть біти даних, а за ними, можливо, біт парності. Після всіх бітів даних передається стоп-біт. Старт-біт і стоп-біт завжди мають певне значення: старт-біт кодується логічним нулем, а стоп-біт – логічною одиницею. Між передачею стоп-біта одного байта і старт-біта наступного байта може проходити довільний час. Асинхронний режим сильно залежить від похибок синхрогенераторів, що задає моменти прийому бітів. Чим вища швидкість передачі, тим більша ця похибка. У результаті цих і деяких інших обмежень швидкість передачі в асинхронному режимі обмежена сотнями кілобіт у секунду (стандартні швидкості: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 біт/с).

Якщо синхронізація дуже якісна (наприклад, використовується додаткова лінія, якою передаються синхросигнали), то можливо передавати потік даних без додаткової синхронізації окремих байтів. Такий режим називається **синхронним** (*synchronous*). Передача бітів даних випереджається і закінчується видачею в канал символу синхронізації. При відсутності даних передавач повинен постійно передавати в канал символи синхронізації.

У випадку **ізохронної** (*isochronous*) передачі відправлення кадрів даних відбувається в задані (відомі приймачу і відправникові) моменти часу. При цьому дані, передані одним вузлом з постійною швидкістю, будуть надходити до приймача з тією же швидкістю. Ізохронна передача необхідна, наприклад, для доставки оцифрованого відеозображення або звуку.

Плезиохронна (*plesiochronous*) передача вимагає внутрішньої синхронізації вузлів від джерел з номінально збіжними частотами. Термін “плезиохронна” означає “майже синхронна”, оскільки частоти джерел точно не збігаються, і згодом накопичується розбіжність, що компенсується вставкою фіктивних даних.

Контрольні запитання

1. Що таке смуга пропускання каналу?
2. Що таке пропускна здатність каналу?
3. Що дозволяє визначити теорема Найквиста, теорема Шенона?
4. Що таке метод доступу. Які методи доступу використовуються в комп'ютерних мережах?
5. Які способи комутацій використовуються в комп'ютерних мережах?
6. Що таке канал передачі даних, як вони підрозділяються?
7. Які види модуляції існують?
8. Що таке модем? На які групи підрозділяються модеми за областю застосування?
9. Які існують групи модемних протоколів?
10. Що таке режим передачі і які режими передачі існують в комп'ютерних мережах?
11. Що таке асинхронна, синхронна, ізохронна і плезиохронна передача?

4 ЛІНІЇ ЗВ'ЯЗКУ, ЯКІ ЗАСТОСОВУЮТЬСЯ В МЕРЕЖАХ ПЕРЕДАЧІ ДАНИХ

Середовищем передачі інформації називаються ті лінії зв'язку (або канали зв'язку), якими проходить обмін інформацією між комп'ютерами. У переважній більшості комп'ютерних мереж (особливо локальних) використовуються провідні або кабельні канали зв'язку, хоча існують і бездротові мережі, які зараз набувають все більшого застосування, особливо в портативних комп'ютерах.

Інформація в локальних мережах найчастіше передається в послідовному коді, тобто біт за бітом. Така передача повільніша і складніша, ніж при використанні паралельного коду. Однак треба враховувати те, що при більш швидкій паралельній передачі (декількома кабелями одночасно) збільшується кількість сполучних кабелів у число разів, що дорівнює кількості розрядів паралельного коду (наприклад, в 8 разів при 8-ми розрядному коді). Це зовсім не дріб'язок, як може здатися на перший погляд. При значних відстанях між абонентами мережі вартість кабелю цілком порівнянна з вартістю комп'ютерів і навіть може перевершувати її. Значно дешевше обійдеться також пошук ушкоджень і ремонт кабелю.

Але це ще не все. Передача на більші відстані при будь-якому типі кабелю вимагає складної передавальної і приймальної апаратури, тому що при цьому необхідно формувати потужний сигнал на передавальному кінці і детектувати слабкий сигнал на приймальному кінці. При послідовній передачі для цього потрібен усього один передавач і один приймач. При паралельній же кількість необхідних передавачів і приймачів зростає пропорційно розрядності використовуваного паралельного коду. У зв'язку із цим, навіть якщо розробляється мережа незначної довжини (порядку десятка метрів) найчастіше вибирають послідовну передачу. До того ж при паралельній передачі надзвичайно важливо, щоб довжини окремих кабелів точно дорівнювали одна одній. Інакше в результаті проходження кабелем різної довжини між сигналами на приймальному кінці утвориться часовий зсув, що може призвести до збоїв у роботі або навіть до повної непрацездатності мережі. Наприклад, при швидкості передачі 100 Мбіт/с і тривалості біта 10 нс цей часовий зсув не повинен перевищувати 5–10 нс. Таку величину зсуву дає різниця в довжинах кабелів в 1–2 метри. При довжині кабелю 1000 метрів це становить 0,1–0,2%.

Треба відзначити, що в деяких високошвидкісних локальних мережах все-таки використовують паралельну передачу 2–4-ма парами провідників у одному кабелі, що дозволяє при заданій швидкості передачі застосовувати більш дешеві кабелі з меншою смугою пропускання. Але припустима довжина кабелів при цьому не перевищує сотні метрів. Прикладом може служити сегмент 100 *Base-T*4 *мережі Fast Ethernet*.

Промисловістю випускається величезна кількість типів кабелів, наприклад, тільки одна найбільша кабельна компанія *Belden* пропонує більше 2000 їхніх найменувань.

Усі кабелі можна розділити на три великі групи:

- електричні (мідні) кабелі на основі кручених пар проводів (*twisted pair*), які діляться на екрановані (*shielded twisted pair, STP*) і неекрановані (*unshielded twisted pair, UTP*);
- електричні (мідні) *коаксіальні кабелі* (coaxial cable);
- оптоволоконні кабелі (*fibre optic*).

Кожний тип кабелю має свої переваги і недоліки, так що при виборі треба враховувати як особливості розв'язуваного завдання, так і особливості конкретної мережі, у тому числі і використовуваної топології.

Можна виділити такі основні **параметри кабелів**, принципово важливі для використання в локальних мережах:

1. **Смуга пропускання** кабелю (частотний діапазон сигналів, що пропускаються кабелем) і **загасання сигналу** в кабелі. Два цих параметри тісно пов'язані між собою, тому що з ростом частоти сигналу росте загасання сигналу. Треба вибирати кабель, який на заданій частоті сигналу має прийнятне загасання. Або ж треба вибирати частоту сигналу, на якій загасання ще прийнятне. Загасання виміряється в децибелах і пропорційно довжині кабелю.

2. **Перешкодозахищеність** кабелю і забезпечувана ним **таємність** передачі інформації. Ці два взаємозалежних параметри показують, як кабель взаємодіє з навколишнім середовищем, тобто як він реагує на зовнішні перешкоди і наскільки просто прослухати інформацію, передану кабелем.

3. **Швидкість поширення сигналу** кабелем або зворотний параметр – **затримка сигналу** на метр довжини кабелю. Цей параметр має принципове значення при виборі довжини мережі. Типові величини швидкості поширення сигналу – від 0,6 до 0,8 від швидкості поширення світла у вакуумі. Відповідно типові величини затримок – від 4 до 5 нс/м.

4. Для електричних кабелів дуже важливою величиною є **хвильовий опір** кабелю. Хвильовий опір важливо враховувати при узгодженні кабелю для запобігання відбиттю сигналу від кінців кабелю. Хвильовий опір залежить від форми і взаєморозташування провідників, від технології виготовлення і матеріалу діелектрика кабелю. Типові значення хвильового опору – від 50 до 150 Ом.

У цей час діють такі стандарти на кабелі:

- *EIA/TIA* 568 (*Commercial Building Telecommunications Cabling Standard*) – американський;
- *ISO/IEC IS* 11801 (*Generic cabling for customer premises*) – міжнародний;
- *CENELEC EN* 50173 (*Generic cabling systems*) – європейський.

Ці стандарти описують практично однакові кабельні системи, але відрізняються термінологією і нормами на параметри.

4.1 Кабелі на основі кручених пар

Кручені пари проводів використовуються в дешевих і сьогодні, мабуть, найпопулярніших кабелях. Кабель на основі кручених пар являє собою декілька пар скручених попарно ізольованих мідних проводів у єдиній діелектричній (пластиковій) оболонці. Він досить гнучкий і зручний для прокладки. Скручування проводів дозволяє звести до мінімуму індуктивні наведення кабелів один на одного і знизити вплив перехідних процесів.

Звичайно до кабелю входить дві (рис. 4.1) або чотири кручені пари.

Рисунок 4.1 – Кабель із крученими парами

Неекрановані кручені пари характеризуються слабкою захищеністю від зовнішніх електромагнітних перешкод, а також від підслуховування, що може здійснюватися з метою, наприклад, промислового шпигунства. Причому перехоплення переданої мережею інформації можливо як за допомогою контактного методу (наприклад, за допомогою двох голок, уткнутих у кабель), так і за допомогою безконтактного методу, що зводиться до радіоперехоплення випромінюваних кабелем електромагнітних полів. Причому дія перешкод і величина випромінювання збільшується із зростанням довжини кабелю. Для усунення цих недоліків застосовується екранування кабелів.

У випадку екранованої крученої пари *STP* кожна із кручених пар міститься в металевій обгортці-екрані для зменшення випромінювань кабелю, захисту від зовнішніх електромагнітних перешкод і зниження взаємного впливу пар проводів один на одного (*crosstalk* – перехресні наведення). Для того щоб екран захищав від перешкод, він повинен бути обов’язково заземлений. Звичайно, екранована кручена пара помітно дорожча, ніж неекранована. Її використання вимагає спеціальних екранованих рознімань. Тому зустрічається вона значно рідше, ніж неекранована кручена пара.

Основні переваги неекранованих кручених пар – простота монтажу рознімань на кінцях кабелю, а також ремонту будь-яких ушкоджень у порівнянні з іншими типами кабелю. Всі інші характеристики в них гірші, ніж в інших кабелів. Наприклад, при заданій швидкості передачі загасання сигналу (зменшення його рівня в міру проходження по кабелю) у них більше, ніж у коаксіальних кабелів. Якщо врахувати ще низьку перешкодозахищеність, то зрозуміло, чому лінії зв’язку на основі кручених пар, як правило, досить короткі (звичайно в межах 100 метрів). У цей час кручена пара використовується для передачі інформації на швидкостях до

1000 Мбіт/с, хоча технічні проблеми, що виникають при таких швидкостях, є досить складними.

Відповідно до стандарту *EIA/TIA* 568 існує п'ять основних і дві додаткові категорії кабелів на основі неекранованої крученої пари (*UTP*):

1. Кабель **категорії 1** – це звичайний телефонний кабель (пари проводів некручені), яким може передаватися тільки розмова. Цей тип кабелю має великий розкид параметрів (хвильового опору, смуги пропущення, перехресних наведень).

2. Кабель **категорії 2** – це кабель із кручених пар для передачі даних у смузі частот до 1 Мгц. Зараз він використовується дуже рідко. Стандарт *EIA/TIA* 568 не розрізняє кабелі категорій 1 і 2.

3. Кабель **категорії 3** – це кабель для передачі даних у смузі частот до 16 МГц, що складається із кручених пар з дев'ятьма витками проводів на метр довжини. Кабель тестується на всі параметри і має хвильовий опір 100 Ом. Це найпростіший тип кабелів, рекомендований стандартом для локальних мереж. Ще недавно він був найпоширенішим, але зараз повсюдно витісняється кабелем категорії 5.

4. Кабель **категорії 4** – це кабель, що передає дані в смузі частот до 20 Мгц. Використовується рідко, тому що не занадто помітно відрізняється від категорії 3. Стандартом рекомендується замість кабелю категорії 3 переходити відразу на кабель категорії 5. Кабель категорії 4 тестується на всі параметри і має хвильовий опір 100 Ом. Кабель був створений для роботи в мережах за стандартом *IEEE* 802.5.

5. Кабель **категорії 5** – зараз є найрозповсюдженішим кабелем, розрахованим на передачу даних у смузі частот до 100 Мгц. Складається із кручених пар, що мають не менш 27 витків на метр довжини (8 витків на фут). Кабель тестується на всі параметри і має хвильовий опір 100 Ом. Рекомендується застосовувати його в сучасних високошвидкісних мережах типу *Fast Ethernet* і *TPFDDI*. Кабель категорії 5 приблизно на 30– 50% дорожче, ніж кабель категорії 3.

6. Кабель **категорії 6** – перспективний тип кабелю для передачі даних у смузі частот до 200 (або 250) Мгц.

7. Кабель **категорії 7** – перспективний тип кабелю для передачі даних у смузі частот до 600 Мгц.

Відповідно до стандарту *EIA/TIA* 568 повний хвильовий опір найбільш розповсюджених кабелів категорій 3, 4 і 5 повинен становити 100 Ом ±15 % у частотному діапазоні від 1 МГц до максимальної частоти кабелю. Вимоги не є фіксованими, величина хвильового опору може знаходитись в діапазоні від 85 до 115 Ом. Тут же слід зазначити, що хвильовий опір екранованої крученої пари *STP* за стандартом повинен дорівнювати 150 Ом ±15 %. Для узгодження опорів кабелю і устаткування у випадку їхньої розбіжності застосовують трансформатори (*Balun*). Існує також екранована кручена пара із хвильовим опором 100 Ом, але використовується вона досить рідко.

Другий найважливіший параметр, що задається стандартом, – це максимальне загасання сигналу, переданого по кабелю, на різних частотах. У таблиці 4.1 наведені граничні значення величини загасання в децибелах для кабелів категорій 3, 4 і 5 на відстань 1000 футів (тобто 305 метрів) при нормальній температурі навколишнього середовища 20° С.

Таблиця 4.1 – Максимальне загасання в кабелях

Частота, МГц	Максимальне загасання, дБ		
	Категорія 3	Категорія 4	Категорія 5
0,064	2,8	2,3	2,2
0,256	4,0	3,4	3,2
0,512	5,6	4,6	4,5
0,772	6,8	5,7	5,5
1,0	7,8	6,5	6,3
4,0	17	13	13
8,0	26	19	18
10,0	30	22	20
16,0	40	27	25
20,0	—	31	28
25,0	—	—	32
31,25	—	—	36
62,5	—	—	52
100	—	—	67

З таблиці видно, що величини загасання на частотах, близьких до граничних, для всіх кабелів є дуже значними. Навіть на невеликих відстанях сигнал послаблюється в десятки і сотні разів, що ставить високі вимоги до приймачів сигналу.

Ще один специфічний параметр, обумовлений стандартом, – це величина так званого перехресного наведення на ближньому кінці (*NEXT – Near End CrossTalk*). Вона характеризує вплив різних проводів у кабелі один на одного (рис. 4.2.) Сигнал, переданий однією із кручених пар кабелю (верхня пара), наводить індуктивну перешкоду на іншу (нижню) кручену пару кабелю. Дві кручені пари в мережі звичайно передають інформацію в різні сторони, тому найбільш важливе наведення на ближньому кінці приймальної пари (нижньої на рисунку), тому що саме там знаходиться приймач інформації. Перехресне наведення на далекому кінці (*FEXT – Far End CrossTalk*) не має такого великого значення.

Таблиця 4.2 – Припустимі рівні перехресних наведень *NEXT*

Частота, МГц	Перехресне наведення на ближньому кінці, дБ		
	Категорія 3	Категорія 4	Категорія 5
0,150	- 54	-68	-74
0,772	-43	-58	-64
1,0	-41	-56	-62
4,0	-32	-47	-53
8,0	-28	-42	-48
10,0	-26	-41	-47
16,0	-23	-38	-44
20,0	—	-36	-42
25,0	—	—	-41
31,25	—	—	-40
62,5	—	—	-35
100,0	—	—	-32

У таблиці 4.2. подані значення допустимого перехресного наведення на ближньому кінці для кабелів категорій 3, 4 і 5 на різних частотах сигналу. Природно, більш якісні кабелі забезпечують меншу величину перехресного наведення.

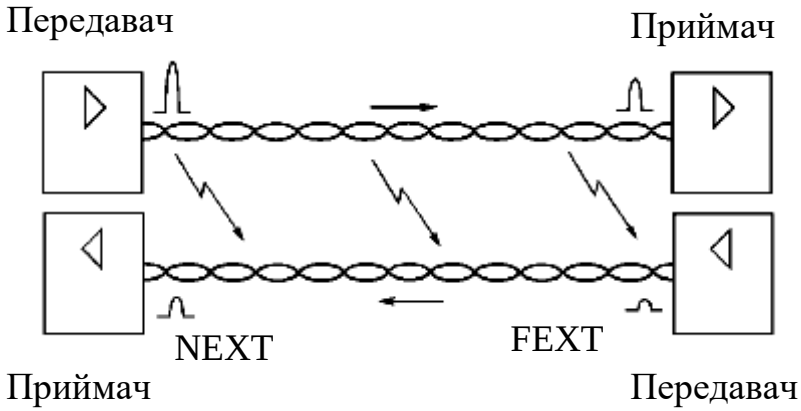

Рисунок 4.2 – Перехресні перешкоди в кабелях на кручених парах

Стандарт визначає також максимально припустиму величину робочої ємності кожної із кручених пар кабелів категорії 4 і 5. Вона повинна становити не більше 17 нФ на 305 метрів (1000 футів) при частоті сигналу 1 кГц і температурі навколишнього середовища 20° С.

Для приєднання кручених пар використовуються рознімання (конектори) типу *RJ*-45, схожі на рознімання, що використовуються в телефонах (*RJ*-11), але трохи більші за розміром. Рознімання *RJ*-45 мають вісім контактів замість чотирьох у випадку *RJ*-11. Приєднуються рознімання до кабелю за допомогою спеціальних обтискних інструментів. При цьому золочені голчасті контакти рознімання проколюють ізоляцію кожного дроту, входять між його жилами і забезпечують надійне і якісне з'єднання. Треба враховувати, що при установці рознімань стандартом допускається розплетіння крученої пари кабелю на довжину не більше одного сантиметра.

Найчастіше кручені пари використовуються для передачі даних в одному напрямку (точка-точка), тобто в топологіях типу "зірка" або "кільце". Топологія «шина» звичайно орієнтується на коаксіальний кабель. Тому зовнішні термінатори, які погоджують непідключені кінці кабелю, для кручених пар практично ніколи не застосовуються.

Кабелі випускаються з двома типами зовнішніх оболонок:

1) кабель у полівінілхлоридній (ПВХ, *PVC*) оболонці дешевший і призначений для роботи в порівняно комфортних умовах експлуатації.

2) кабель у тефлоновій оболонці дорожчий і призначений для більш жорстких умов експлуатації.

Кабель у ПВХ оболонці називається ще *non-plenum,* а в тефлонової – *plenum*. Термін *plenum* означає в цьому випадку простір під фальшпідлогою і над підвісною стелею, де зручно розміщати кабелі мережі. Для прокладки в цих схованих від очей просторах зручнішим є кабель у тефлоновій оболонці. Такий кабель ще і горить набагато гірше, ніж ПВХ-кабель, і не виділяє при цьому отрутних газів у великій кількості.

Ще один важливий параметр будь-якого кабелю, що жорстко не визначається стандартом, але може істотно вплинути на працездатність мережі, – це швидкість поширення сигналу в кабелі або, інакше кажучи, затримка поширення сигналу в кабелі, розраховуючи на одиницю довжини.

Виробники кабелів іноді вказують величину ***затримки*** на метр довжини, а іноді – швидкість поширення сигналу щодо швидкості світла (або *NVP – Nominal Velocity of Propagation*, як її часто називають у документації). Зв'язані ці дві величини простою формулою(4.1):

$$t_з = 1/(3 \times 10^8 \times NVP), \qquad (4.1)$$

де $t_з$ – величина затримки на метр довжини кабелю в наносекундах. Наприклад, якщо *NVP*=0,65 (65 % від швидкості світла), то затримка $t_з$ буде дорівнювати 5,13 нс/м. Типова величина затримки більшості сучасних кабелів становить близько 4—5 нс/м.

Варто також відзначити, що кожний із проводів, які входять до кабелю на основі кручених пар, як правило, має свій колір ізоляції, що істотно спрощує монтаж рознімань, особливо в тому випадку, коли кінці

кабелю знаходяться у різних кімнатах, тому контроль за допомогою приладів є утрудженим.

Прикладом кабелю з екранованими крученими парами може служити кабель *STP IBM* типу 1, що містить у собі дві екрановані кручені пари *AWG* типу 22. Хвильовий опір кожної пари становить 150 Ом. Для цього кабелю застосовуються спеціальні рознімання, що відрізняються від рознімань для неекранованої крученої пари (наприклад, *DB*9). Є і екрановані версії рознімання *RJ*-45.

4.2 Коаксіальні кабелі

Коаксіальний кабель являє собою електричний кабель, що складається із центрального мідного проведення (1) і металевого обплетення – екрана (3), розділених між собою шаром діелектрика – внутрішньої ізоляції (2) і поміщених у загальну зовнішню оболонку (4) (рис. 4.3).

Рисунок 4.3 – Коаксіальний кабель

Коаксіальний кабель донедавна був дуже популярним, що пов’язано з його високою перешкодозахищеністю (завдяки металевому обплетенню), більш широкими, чим у випадку крученої пари, смугами пропущення (понад 1ГГц), а також більшими припустимими відстанями передачі (до кілометра). До нього важче механічно підключитися для несанкціонованого прослуховування мережі, він дає також помітно менше електромагнітних випромінювань зовні. Однак монтаж і ремонт коаксіального кабелю істотно складніший, ніж крученої пари, а вартість його вище (він дорожче приблизно в 1,5 – 3 рази). Складніше і установка рознімань на кінцях кабелю. Зараз його застосовують рідше, ніж кручену пару. Стандарт *EIA/TIA*-568 містить у собі тільки один тип коаксіального кабелю, застосовуваний у мережі *Ethernet*.

В основному коаксіальний кабель застосувується у мережах з топологією типу «шина». При цьому на кінцях кабелю обов’язково повинні встановлюватися термінатори для запобігання внутрішнім відбиттям сигналу, причому один (і тільки один!) з термінаторів повинен бути заземлений. Без заземлення металеве обплетення не захищає мережу від зовнішніх електромагнітних перешкод і не знижує випромінювання переданої мережею інформації в зовнішнє середовище. Але при заземленні обплетення у двох або більше точках з ладу може вийти не тільки мережне устаткування, але і комп’ютери, підключені до мережі. Термінатори повинні бути обов’язково узгодженими з кабелем. Необхідно, щоб їхній

опір дорівнював хвильовому опору кабелю. Наприклад, якщо використовується 50-омний кабель, для нього підходять тільки 50-омні термінатори.

Рідше коаксіальні кабелі застосовуються в мережах з топологією зірка (наприклад, пасивна зірка в мережі Arcnet). У цьому випадку проблема узгодження істотно спрощується, тому що не потребує зовнішніх термінаторів на вільних кінцях.

Хвильовий опір кабелю вказується в супровідній документації. Найчастіше в локальних мережах застосовуються 50-омні (*RG*-58, *RG*-11, *RG*-8) і 93-омні кабелі (*RG*-62). Розповсюджені в телевізійній техніці 75-омні кабелі в локальних мережах не використовуються.

Марок коаксіального кабелю небагато. Він не вважається особливо перспективним. Не випадково в мережі *Fast Ethernet* не передбачено застосування коаксіальних кабелів. Однак у багатьох випадках класична шинна топологія (а не пасивна зірка) є дуже зручною. Як ми вже відзначали, вона не вимагає застосування додаткових пристроїв – концентраторів.

Існує два основних типи коаксіального кабелю:

1) тонкий (*thin*) кабель, що має діаметр близько 0,5 см, більш гнучкий;

2) товстий (*thick*) кабель, діаметром близько 1 см, значно більш твердий. Він являє собою класичний варіант коаксіального кабелю, що уже майже повністю витиснутий сучасним тонким кабелем.

Тонкий кабель використовується для передачі на коротші відстані, чим товстий, оскільки сигнал у ньому загасає сильніше. Зате з тонким кабелем набагато зручніше працювати: його можна оперативно прокласти до кожного комп’ютера, а товстий вимагає твердої фіксації на стіні приміщення.

Підключення до тонкого кабелю (за допомогою рознімань *BNC* байонетного типу) простіше і не вимагає додаткового устаткування. А для підключення до товстого кабелю треба використовувати спеціальні досить дорогі пристрої, що проколюють його оболонки і встановлюють контакт як із центральною жилою, так і з екраном. Товстий кабель приблизно вдвічі дорожче, ніж тонкий, тому тонкий кабель застосовується набагато частіше.

Як і у випадку кручених пар, важливим параметром коаксіального кабелю є тип його зовнішньої оболонки. Точно так само в цьому випадку застосовуються як *non-plenum* (*PVC*), так і *plenum* кабелі. Природно, тефлоновий кабель дорожче полівінілхлоридного. Звичайно тип оболонки можна відрізнити за фарбуванням (наприклад, для *PVC* кабелю фірма *Belden* використовує жовтий колір, а для тефлонового – жовтогарячий).

Типові величини затримки поширення сигналу в коаксіальному кабелі становлять для тонкого кабелю близько 5 нс/м, а для товстого – близько 4,5 нс/м.

Існують варіанти коаксіального кабелю з подвійним екраном (один екран розташований усередині іншого і відділений від нього додатковим

шаром ізоляції). Такі кабелі мають кращу перешкодозахищеність і захист від прослуховування, але вони небагато дорожче звичайних.

Сьогодні вважається, що коаксіальний кабель застарів. У більшості випадків його цілком може замінити кручена пара або оптоволоконний кабель. І нові стандарти на кабельні системи вже не включають його в перелік типів кабелів.

4.3 Оптоволоконні кабелі

Оптоволоконний (він же волоконно-оптичний) кабель – це принципово інший тип кабелю в порівнянні з розглянутими двома типами електричного або мідного кабелю. Інформація з нього передається не електричним сигналом, а світловим. Головний його елемент – це прозоре скловолокно, яким світло проходить на величезні відстані (до десятків кілометрів) з незначним ослабленням.

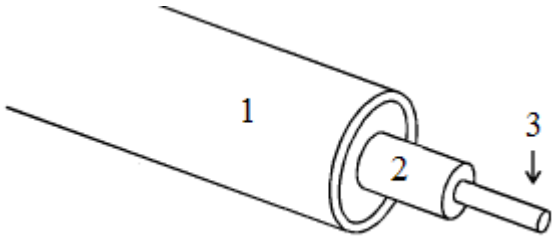

Рис. 4.4 – Структура оптоволоконного кабелю

Структура оптоволоконного кабелю дуже проста і схожа на структуру коаксіального електричного кабелю (рис. 4.4). Тільки замість центрального мідного проводу тут використовується тонкий (діаметром близько 1-10 мкм) провідник з оптично прозорого матерілу (скло, пластик) (3), а замість внутрішньої ізоляції – скляна або пластикова оболонка (2), що не дозволяє світлу виходити за межі центрального провідника. У цьому випадку мова йде про режим так званого повного внутрішнього відбиття світла від границі двох речовин з різними коефіцієнтами переломлення (у скляної оболонки коефіцієнт переломлення значно нижчий, ніж у центрального волокна). Металеве обплетення кабелю відсутнє, тому що екранування від зовнішніх електромагнітних перешкод тут не потрібне. Однак іноді його все-таки застосовують для механічного захисту від навколишнього середовища (такий кабель іноді називають броньовим, він може поєднувати під одною оболонкою декілька оптоволоконних кабелів).

Оптоволоконний кабель має виняткові характеристики за перешкодозахищеністю і таємністю переданої інформації. Ніякі зовнішні електромагнітні перешкоди не здатні спотворити світловий сигнал, а сам сигнал не породжує зовнішніх електромагнітних випромінювань.

Підключитися до цього типу кабелю для несанкціонованого прослуховування мережі практично неможливо, тому що при цьому порушується цілісність кабелю. Теоретично можлива смуга пропущення такого кабелю досягає величини 10^{12} Гц, тобто 1000 ГГц, що незрівнянно вище, ніж в електричних кабелів. Вартість оптоволоконного кабелю постійно знижується і зараз приблизно дорівнює вартості тонкого коаксіального кабелю.

Типова величина загасання сигналу в оптоволоконних кабелях на частотах, що використовуються у локальних мережах, становить від 5 до 20 дБ/км, що приблизно відповідає показникам електричних кабелів на низьких частотах. Але у випадку оптоволоконного кабелю при рості частоти переданого сигналу загасання збільшується дуже незначно, і на більших частотах (особливо понад 200 МГц) його перевага перед електричним кабелем є незаперечною, у нього просто немає конкурентів.

Однак оптоволоконний кабель має і деякі недоліки. Головний з них – висока складність монтажу (при установці рознімань необхідна мікронна точність, від точності відколу скловолокна і ступеня його полірування сильно залежить загасання в рознімanні). Для установки рознімань застосовують зварювання або склеювання за допомогою спеціального гелю, що має такий же коефіцієнт переломлення світла, що і скловолокно. У кожному разі для цього потрібна висока кваліфікація персоналу і спеціальні інструменти. Тому найчастіше оптоволоконний кабель продається у вигляді заздалегідь нарізаних шматків різної довжини, на обох кінцях яких уже встановлені рознімання потрібного типу. Варто пам’ятати, що неякісна установка рознімання різко знижує допустиму довжину кабелю, обумовленою загасанням.

Також треба пам’ятати, що використання оптоволоконного кабелю вимагає спеціальних оптичних приймачів і передавачів, які перетворять світлові сигнали в електричні і навпаки, що часом істотно збільшує вартість мережі в цілому.

Оптоволоконні кабелі допускають розгалуження сигналів (для цього виробляються спеціальні пасивні **розгалужувачі** (*couplers*) на 2—8 каналів), але, як правило, їх використовують для передачі даних тільки в одному напрямку між одним передавачем і одним приймачем. Адже будь-яке розгалуження сильно послабляє світловий сигнал, і якщо розгалужень буде багато, то світло може просто не дійти до кінця мережі. Крім того, при розгалуженні є і внутрішні втрати, так що сумарна потужність сигналу на виході менше вхідної потужності.

Оптоволоконний кабель менш міцний і гнучкий, чим електричний. Типова величина допустимого радіуса вигину становить близько 10 — 20 см, при менших радіусах вигину центральне волокно може зламатися. Погано впливає на кабель і механічне розтягання та роздавлювання.

Чутливий оптоволоконний кабель і до іонізуючих випромінювань, через які знижується прозорість скловолокна, тобто збільшується

загасання сигналу. Різкі перепади температури також негативно позначаються на ньому, скловолокно може тріснути.

Застосовують оптоволоконний кабель тільки в мережах з топологією “зірка” і “кільце”. Ніяких проблем узгодження і заземлення в цьому випадку не існує. Кабель забезпечує ідеальну гальванічну розв’язку комп’ютерів мережі. У майбутньому цей тип кабелю, напевно, витисне електричні кабелі або, у всякому разі, сильно потіснить їх. Запаси міді на планеті виснажуються, а сировини для виробництва скла цілком достатньо.

Існують два різних типи оптоволоконного кабелю*:*

1. **багатомодовий** або **мультімодовий** кабель, більш дешевий, але менш якісний;

2. **одномодовий** кабель, більш дорогий, але має кращі характеристики в порівнянні з першим.

Суть розходження між цими двома типами зводиться до різних режимів проходження світлових променів у кабелі.

В одномодовом кабелі практично всі промені проходять той самий шлях, у результаті чого вони досягають приймача одночасно, і форма сигналу майже не спотворюється (рис. 4.5). Одномодовий кабель має діаметр центрального волокна близько 1,3 мкм і передає світло тільки з такою ж довжиною хвилі (1,3 мкм). Дисперсія і втрати сигналу при цьому дуже незначні, що дозволяє передавати сигнали на значно більшу відстань, чим у випадку застосування багатомодового кабелю. Для одномодового кабелю застосовуються лазерні приймачі-передавачі, які використовують світло винятково з необхідною довжиною хвилі. Такі приймачі-передавачі поки ще порівняно дорогі і недовговічні. Однак у перспективі одномодовий кабель повинен стати основним типом завдяки своїм

Рисунок 4.5 – Поширення світла в одномодовому кабелі

прекрасним характеристикам. До того ж лазери мають більшу швидкодію, чим звичайні світлодіоди. Загасання сигналу в одномодовому кабелі становить близько 5 дБ/км і може бути навіть знижене до 1 дБ/км.

У багатомодовому кабелі траєкторії світлових променів мають помітний розкид, у результаті чого форма сигналу на приймальному кінці кабелю спотворюється (рис. 4.6). Центральне волокно має діаметр 62,5 мкм, а діаметр зовнішньої оболонки 125 мкм (це іноді позначається як 62,5/125). Для передачі використовується звичайний (не лазерний) світлодіод, що знижує вартість і збільшує термін служби приймачів-передавачів у порівнянні з одномодовим кабелем. Довжина хвилі світла в багатомодовому кабелі дорівнює 0,85 мкм, при цьому спостерігається

розкид довжин хвиль близько 30 – 50 нм. Припустима довжина кабелю становить 2 – 5 км. Багатомодовий кабель – це зараз основний тип оптоволоконного кабелю, тому що він дешевший і доступніший. Загасання в багатомодовому кабелі більше, ніж в одномодовому і становить 5 – 20 дБ/км.

Рисунок 4.6 – Поширення світла в багатомодовому кабелі

Типова величина затримки для найпоширеніших кабелів становить близько 4 – 5 нс/м, що близько до величини затримки в електричних кабелях.

Оптоволоконні кабелі, як і електричні, випускаються у виконанні *plenum* і *non-plenum*.

4.4 Бездротові канали зв'язку

Крім кабельних каналів, у комп'ютерних мережах використовуються також бездротові канали. Їхня головна перевага полягає в тому, що не потрібно ніякої прокладки проводів (не треба робити отворів у стінах, закріплювати кабель у трубах і ринвах, прокладати його під фальшпідлогами, над підвісними стелями або у вентиляційних шахтах, шукати і усувати ушкодження). До того ж комп'ютерні мережі можна легко переміщати в межах кімнати або будинку, тому що вони ні до чого не прив'язані.

Радіоканал використовує передачу інформації з радіохвиль, тому теоретично він може забезпечити зв'язок на багато десятків, сотнів і навіть тисяч кілометрів. Швидкість передачі досягає десятків Мбіт у секунду (тут багато чого залежить від обраної довжини хвилі і способу кодування).

Особливість радіоканалу полягає в тому, що сигнал вільно випромінюється в ефір, він не замкнутий у кабель, тому виникають проблеми сумісності з іншими джерелами радіохвиль (радіо- і телеоповіщальними станціями, радарами, радіоаматорськими і професійними передавачами і т.д.). У радіоканалі використовується передача у вузькому діапазоні частот і модуляція інформаційним сигналом сигналу несучої частоти.

Головним недоліком радіоканалу є його поганий захист від прослуховування, тому що радіохвилі поширюються неконтрольовано. Інший великий недолік радіоканалу – слабка перешкодозахищеність.

Для локальних бездротових мереж (*WLAN – Wireless LAN*) зараз застосовують підключення по радіоканалу на невеликих відстанях

(звичайно до 100 метрів) і в межах прямої видимості. Найчастіше використовуються два частотних діапазони – 2,4 ГГц і 5 ГГц. Швидкість передачі – до 54 Мбіт/с. Розповсюдженим є варіант зі швидкістю 11 Мбіт/с.

Мережі *WLAN* дозволяють встановлювати бездротові мережні з'єднання на обмеженій території (звичайно всередині офісного або університетського будинку або в таких громадських місцях, як аеропорти). Вони можуть використовуватися в тимчасових офісах або в інших місцях, де прокладка кабелів нездійсненна, а також як доповнення до наявної провідної локальної мережі, для забезпечення користувачів можливістю працювати переміщуючись будинком.

Популярна технологія *Wi-Fi* (*Wireless Fidelity*) дозволяє організувати зв'язок між комп'ютерами числом від 2 до 15 за допомогою концентратора (який називається точкою доступу, *Access Point, AP*) або декількох концентраторів, якщо комп'ютерів від 10 до 50. Крім того, ця технологія дає можливість зв'язати дві локальні мережі на відстані до 25 кілометрів за допомогою потужних бездротових мостів. Об'єднання комп'ютерів за допомогою однієї точки доступу показане на рис. 4.7. Важливо, що багато мобільних комп'ютерів (ноутбуків) уже мають вбудований контролер *Wi-Fi*, що істотно спрощує їхнє підключення до бездротової мережі.

Рисунок 4.7 – Об'єднання комп'ютерів за допомогою технології *Wi-Fi*

Радіоканал широко застосовується в глобальних мережах як для наземної, так і для супутникового зв'язку. У цьому застосуванні в радіоканалу немає конкурентів, тому що радіохвилі можуть дійти до будь-якої точки земної кулі.

Інфрачервоний канал також не вимагає сполучних проводів, тому що використовує для зв'язку інфрачервоне випромінювання (подібно пульту дистанційного керування домашнього телевізора). Головна його перевага в порівнянні з радіоканалом – нечутливість до електромагнітних перешкод, що дозволяє застосовувати його, наприклад, у виробничих умовах, де завжди багато перешкод від силового устаткування. Хоча у цьому випадку потрібна досить висока потужність передачі, щоб не впливали ніякі інші джерела теплового (інфрачервоного) випромінювання. Погано працює інфрачервоний зв'язок і в умовах сильної запиленості повітря.

Швидкості передачі інформації з інфрачервоного каналу звичайно не перевищують 5–10 Мбіт/с, але при використанні інфрачервоних лазерів може бути досягнута швидкість більше 100 Мбіт/с. Таємність переданої інформації, як і у випадку радіоканалу, не досягається, також потрібні порівняно дорогі приймачі і передавачі. Все це приводить до того, що застосовують інфрачервоні канали в локальних мережах досить рідко. В основному вони використовуються для зв'язку комп'ютерів з периферією (інтерфейс *IrDA*).

Інфрачервоні канали діляться на дві групи:

1. Канали прямої видимості, у яких зв'язок здійснюється на променях, що йдуть безпосередньо від передавача до приймача. При цьому зв'язок можливий тільки при відсутності перешкод між комп'ютерами мережі. Зате довжина каналу прямої видимості може досягати декількох кілометрів.
2. Канали на розсіяному випромінюванні, які працюють на сигналах, відбитих від стін, стелі, підлоги та інших перешкод. Перешкоди в цьому випадку не так суттєві, але зв'язок може здійснюватися тільки в межах одного приміщення.

Якщо говорити про можливі топології, то найбільш природно всі бездротові канали зв'язку підходять для топології типу «шина», у якій інформація передається одночасно всім абонентам. Але при використанні вузьконаправленої передачі і/або частотного поділу каналами можна реалізувати будь-які топології ("кільце", "зірка", комбіновані топології) як на радіоканалі, так і на інфрачервоному каналі.

Контрольні запитання

1. Що таке середовище передачі даних? Які основні середовища передачі даних використовуються в комп'ютерних мережах?
2. Які існують основні параметри кабелів?
3. Що являє собою кабель "кручена пара". Які типи цього кабелю використовуються в комп'ютерних мережах і які їх параметри?
4. Яка структура волоконно-оптичного кабелю? Які типи цього кабелю використовуються в комп'ютерних мережах і які їх параметри?
5. Які бездротові канали зв'язку використовуються в комп'ютерних мережах?

5 ПІДКЛЮЧЕННЯ ЛІНІЙ ЗВ'ЯЗКУ. КОДУВАННЯ СИГНАЛІВ. МЕТОДИ ЦИФРОВОГО КОДУВАННЯ

5.1 Узгодження, екранування і гальванічна розв'язка ліній зв'язку

Електричні лінії зв'язку (кручені пари, коаксіальні кабелі) вимагають проведення спеціальних заходів, без яких неможлива не тільки безпомилкова передача даних, але і взагалі будь-яке функціонування мережі. Оптоволоконні кабелі вирішують усі подібні проблеми автоматично.

Узгодження електричних ліній зв'язку застосовується для забезпечення нормального проходження сигналу довгою лінією без відбиттів і перекручувань. Слід зазначити, що в локальних мережах кабель працює в режимі довгої лінії навіть при мінімальних відстанях між комп'ютерами, тому що швидкості передачі інформації і частотний спектр сигналу дуже великі.

Принцип узгодження кабелю простий: на його кінцях необхідно встановити резистори (термінатори), які мають опір, що дорівнює хвильовому опору використовуваного кабелю.

Хвильовий опір – це параметр даного типу кабелю, що залежить тільки від його пристрою (перетину, кількості і форми провідників, товщини і матеріалу ізоляції і т.д.). Величина хвильового опору обов'язково вказується в супровідній документації на кабель і становить від 50–100 Ом для коаксіального кабелю, до 100–150 Ом для крученої пари або плоского багатодротового кабелю. Точне значення хвильового опору легко можна виміряти за допомогою генератора прямокутних імпульсів і осцилографа саме по відсутності перекручування форми переданого кабелем імпульсу. При цьому необхідно, щоб відхилення величини узгоджуючого резистора не перевищувало 10 % у той або інший бік.

Якщо узгоджуючий навантажувальний опір $R_н$ менше хвильового опору кабелю $R_в$, то фронт переданого прямокутного імпульсу на приймальному кінці буде затягнутий, якщо ж $R_н$ більше $R_в$, то на фронті буде коливальний процес (рис. 5.1).

Мережні адаптери, їхні приймачі і передавачі спеціально розраховуються на роботу з даним типом кабелю з відомим хвильовим опором. Тому навіть при ідеально узгодженому на кінцях кабелю, хвильовий опір якого істотно відрізняється від стандартного, мережа, швидше за все, працювати не буде або буде працювати зі збоями.

Сигнали з пологими фронтами передаються довгим електричним кабелем краще, ніж сигнали із крутими фронтами. Їхня форма значно менше спотворюється (рис. 5.2). Це пов'язано з різницею величин загасання для різних частот (високі частоти загасають сильніше). Найменше спотворюється форма синусоїдального сигналу, він просто зменшується за амплітудою. Для поліпшення якості передачі нерідко

використовуються трапецієподібні або колоколоподібні імпульси (рис. 5.3), близькі за формою до напівхвилі синуса, для чого штучно затягаються або згладжуються фронти прямокутних сигналів.

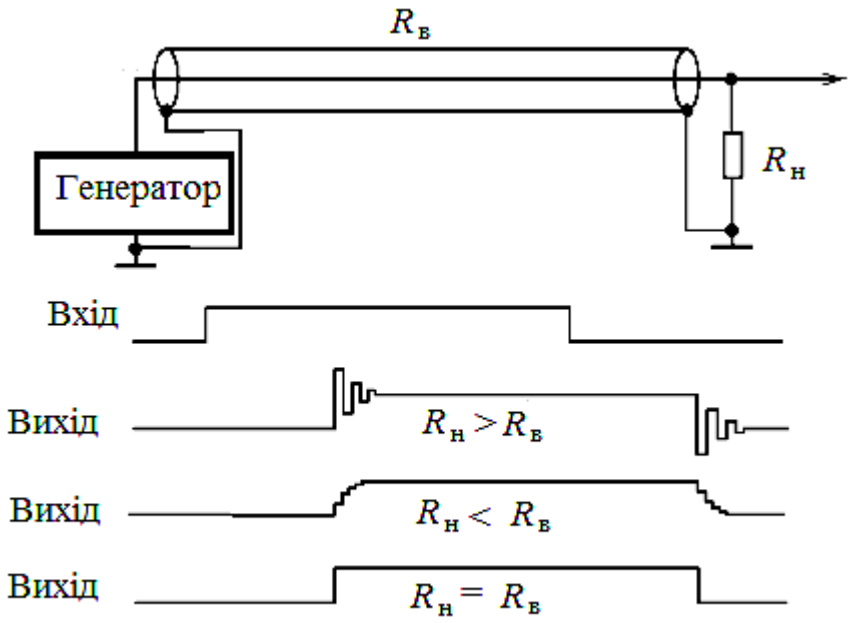

Рисунок 5.1 – Передача сигналів електричним кабелем

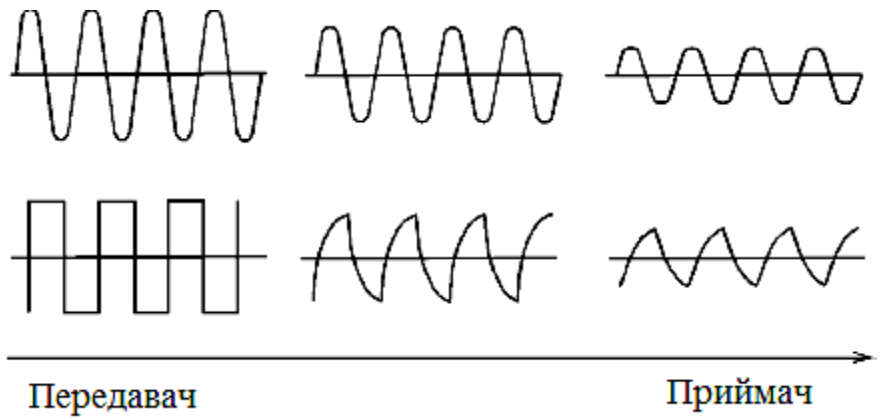

Рис. 5.2 – Загасання сигналів в електричному кабелі

Рис. 5.3 – Трапецієподібний і колоколоподібний імпульси

Екранування електричних ліній зв'язку застосовується для зниження впливу на кабель зовнішніх електромагнітних полів. Екран являє собою мідну або алюмінієву оболонку (плетену або з фольги), у яку вкладаються дроти кабелю. Екранування буде працювати, якщо екран заземлений, оскільки необхідно, щоб наведені на нього струми стікали на землю. Крім того, екранування помітно зменшує і зовнішні випромінювання кабелю, що важливо для забезпечення таємності переданої інформації. Побічними корисними ефектами екранування є збільшення міцності кабелю і труднощі з механічним підключенням до кабелю для підслуховування. Екран помітно підвищує механічну міцність кабелю, хоча збільшує його вартість.

Знизити вплив наведених перешкод можливо і без екрана, якщо використовувати диференціальну передачу сигналу (рис. 5.4). У цьому випадку передача йде двома проводами, причому обидва проводи є сигнальними. Передавач формує протифазні сигнали, а приймач реагує на різницю сигналів в обох проводах. Умовою узгодження є рівність опорів узгоджуючих резисторів R половині хвильового опору кабелю $R_в$. Якщо обидва дроти мають однакову довжину і прокладені поруч (в одному кабелі), то перешкоди діють на них приблизно однаково, і в результаті різницевий сигнал між проводами практично не спотворюється. Саме така диференціальна передача застосовується в кабелях із кручених пар. Але екранування і у цьому випадку істотно поліпшує перешкодостійкість.

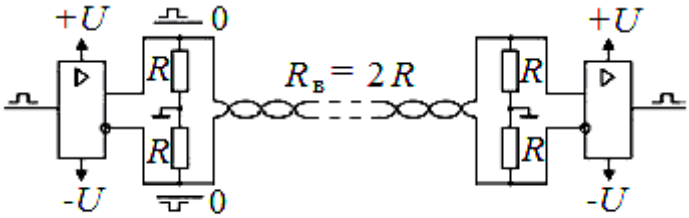

Рис. 5.4 – Диференціальна передача сигналів крученою парою

При використанні електричного кабелю потрібна **гальванічна розв'язка** комп'ютерів від мережі. Справа в тому, що електричним кабелем (як сигнальним проводом, так і екраном) можуть іти не тільки інформаційні сигнали, але і так званий вирівнюючий струм, який виникає внаслідок неідеальності заземлення комп'ютерів.

Коли комп'ютер незаземлений, то на його корпусі створюється наведений потенціал близько 110 вольтів змінного струму (половина живлячої напруги). Його можливо відчути на собі, якщо одною рукою взятися за корпус комп'ютера, а іншою за батарею центрального опалення або за який-небудь заземлений прилад.

При автономній роботі комп'ютера відсутність заземлення, як правило, серйозно не впливає на його роботу. Правда, іноді збільшується кількість збоїв у роботі машини. Але при з'єднанні декількох

територіально рознесених комп’ютерів електричним кабелем заземлення стає більш серйозною проблемою. Якщо один з комп’ютерів, що з’єднуються, заземлений, а інший ні, то можливий вихід з ладу одного з них або обох. Тому комп’ютери вкрай бажано заземлювати.

У випадку використання триконтактної вилки і розетки, у яких є нульовий провод, це виходить автоматично. При двоконтактній вилці і розетці необхідно вживати спеціальних заходів, організовувати заземлення окремим проведенням великого перетину. Варто також відзначити, що у випадку трифазної мережі бажано забезпечити живлення всіх комп’ютерів від однієї фази.

Проблема ускладнюється ще і тим, що "земля", до якої приєднуються комп’ютери, найчастіше далека від ідеалу. Теоретично заземлюючі проводи комп’ютерів повинні сходитися в одному місці, з’єднаному короткою масивною шиною із заритим у землю масивним провідником. Така ситуація можлива, тільки якщо комп’ютери не занадто рознесені, і заземлення дійсно зроблене грамотно. Найчастіше ж заземлювальна «шина» має значну довжину, у результаті чого струми, які по ній стікають, створюють деяку різницю потенціалів між її окремими точками. Особливо велика ця різниця потенціалів у випадку підключення до «шини» потужних і високочастотних споживачів енергії.

Приєднані до однієї і тієї ж «шини», але в різних точках, комп’ютери мають на своїх корпусах різні потенціали (рис. 5.5). У результаті електричним кабелем, що з’єднує комп’ютери, потече вирівнюючий струм (змінний з високочастотними складовими).

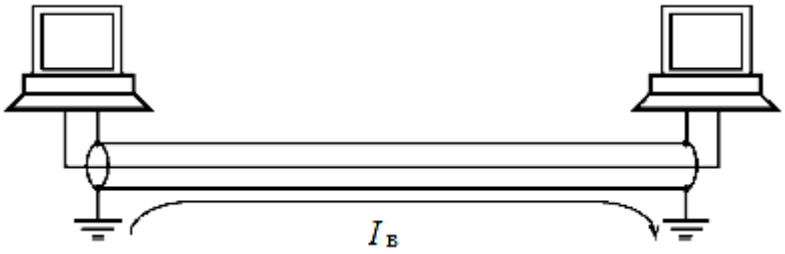

Рис. 5.5 – Вирівнюючий струм при відсутності гальванічної розв’язки

Гірше, коли комп’ютери підключаються до різних «шин» заземлення. Вірівнюючий струм може досягати в цьому випадку величини в декілька ампер. Подібні струми смертельно небезпечні для малосигнальних вузлів комп’ютера. Крім того, струм, що вирівнює, істотно впливає на переданий сигнал, часом повністю забиваючи його. Навіть тоді, коли сигнали передаються без участі екрана (наприклад, двома проводами, вкладеним в екран) індуктивна дія струму, що вирівнює, заважає передачі інформації. Саме тому екран завжди повинен бути заземленим тільки в одній точці.

Якщо кожний з комп’ютерів самостійно заземлений, то заземлення екрана в одній точці стає неможливим без гальванічної розв’язки комп’ютерів від мережі. При такому підключенні не повинно бути зв’язку постійним струмом між корпусом ("землею") комп’ютера і екраном ("землею") мережного кабелю. У той же час, інформаційний сигнал повинен передаватися з комп’ютера в мережу і з мережі в комп’ютер. Для гальванічної розв’язки найчастіше застосовують імпульсні трансформатори, які входять до складу мережного устаткування (наприклад, мережних адаптерів). Трансформатор пропускає високочастотні інформаційні сигнали і забезпечує повну ізоляцію постійному струму.

Грамотне з’єднання комп’ютерів локальної мережі електричним кабелем (рис. 5.6) обов’язково повинне містити в собі наступне:

- кінцеве узгодження кабелю за допомогою термінаторів;
- гальванічну розв’язку комп’ютерів від мережі;
- заземлення кожного комп’ютера;
- заземлення екрана (якщо він є) в одній точці.

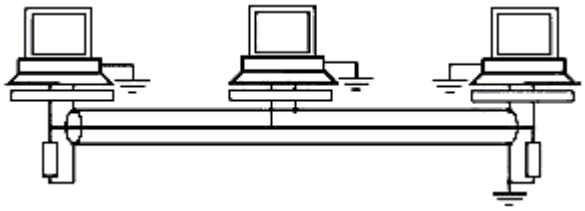

Рис. 5.6 – Правильне з’єднання комп’ютерів мережі (гальванічна розв’язка умовно показана у вигляді прямокутника)

Не варто зневажати якою-небудь із цих вимог. Наприклад, гальванічна розв’язка мережних адаптерів часто розраховується на допустиме напруження ізоляції всього лише 100 В, що при відсутності заземлення одного з комп’ютерів може легко призвести до виходу з ладу його адаптера.

Слід зазначити, що для приєднання коаксіального кабелю найчастіше застосовуються рознімання в металевому корпусі. Цей корпус не повинен з’єднуватися ні з корпусом комп’ютера, ні з "землею" (на платі адаптера він установлений із пластиковою ізоляцією під кріпильною планкою). Заземлення екрана кабелю мережі краще робити не через корпус комп’ютера, а окремим спеціальним проведенням, що забезпечує кращу надійність. Пластмасові корпуси рознімань *RJ*-45 для кабелів з неекранованими кручеными парами знімають цю проблему.

Важливо також враховувати, що екран кабелю, заземлений в одній точці, є радіоантеною із заземленою основою. Він може вловлювати і підсилювати високочастотні перешкоди з довжиною хвилі, кратної його довжині. Для зниження цього "антенного ефекту" застосовується

багатоточечне заземлення екрана високою частотою. У кожному мережному адаптері "земля" мережного кабелю з'єднується з "землею" комп'ютера через високовольтні керамічні конденсатори. Для приклада на рис. 5.7 показана спрощена схема гальванічної розв'язки, застосовувана в мережних адаптерах *Ethernet*.

Приймач-передавач прямо пов'язаний з кабелем мережі, але гальванічно розв'язаний за допомогою трансформаторів від комп'ютера і іншої частини мережного адаптера. Це продиктовано особливостями протоколу *CSMA/CD* і манчестерського коду, який застосовується в *Ethernet*. Забезпечення повної розв'язки живлення приймача-передавача здійснюється за допомогою перетворювача живлячої напруги, що має всередині також трансформаторну гальванічну розв'язку. Обплетення коаксіального кабелю з'єднано із загальним дротом комп'ютера через високовольтний конденсатор. Паралельно конденсатору включаються резистор з більшим опором (1 МОм), що запобігає електричному удару користувача при одночасному торканні ним обплетення кабелю (корпуса рознімання) і корпуса комп'ютера.

У випадку застосування кручених пар все набагато простіше. Кожна кручена пара має імпульсні трансформатори, що розв'язують, на обох своїх кінцях. Жоден із проводів кручених пар не заземлюється (вони обидва є сигнальними). До того ж рознімання для кручених пар мають пластмасовий корпус.

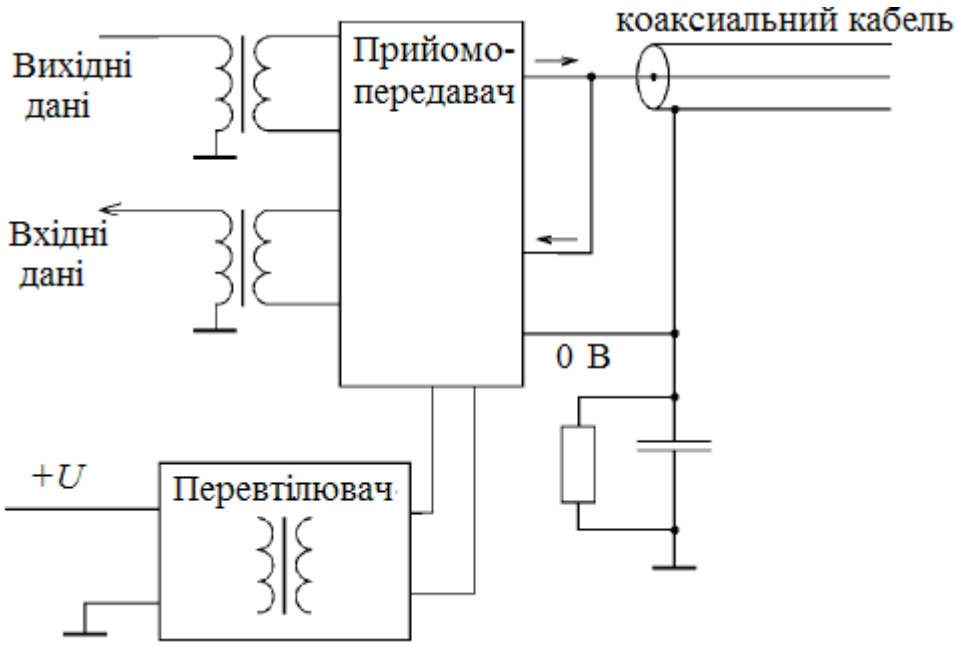

Рис. 5.7 – Схема гальванічної розв'язки в мережі Ethernet

5.2 Кодування сигналів

При передачі цифрової інформації за допомогою цифрових сигналів застосовується цифрове кодування, яке керує послідовністю прямокутних імпульсів відповідно до послідовності переданих даних.

При цифровому кодуванні застосовують або потенційні, або імпульсні коди.

При **потенційному кодуванні** інформативним є рівень сигналу. При **імпульсному кодуванні** використовуються або перепади рівня (транзитивне кодування), або полярність окремих імпульсів (уніполярне, полярне, біполярне кодування).

В окрему групу імпульсних кодів виділяють двофазні коди, при яких у кожному бітовому інтервалі обов'язково присутній перехід з одного стану в інший (такі коди дозволяють виділяти синхросигнал з послідовності станів лінії, тобто вони є самосинхронізуючими).

Вимоги до методів цифрового кодування

При використанні прямокутних імпульсів для передачі дискретної інформації необхідно вибрати такий спосіб кодування, який одночасно досягав би декількох цілей:

- мав при одній і тій же бітовій швидкості найменшу ширину спектра результуючого сигналу;
- забезпечував синхронізацію між передавачем і приймачем;
- мав здатність розпізнавати помилки;
- мав низьку вартість реалізації.

Синхронізація передавача і приймача потрібна для того, щоб приймач точно знав, у який момент часу необхідно зчитувати нову інформацію з лінії зв'язку. Ця проблема в мережах вирішується складніше, ніж при обміні даними між близько розташованими пристроями, наприклад між блоками всередині комп'ютера або ж між комп'ютером і принтером.

На невеликих відстанях добре працює схема, основана на окремій тактуючій лінії зв'язку, так що інформація знімається з лінії даних тільки в момент приходу тактового імпульсу. У мережах використання цієї схеми викликає труднощі через неоднорідність характеристик провідників у кабелях. На більших відстанях нерівномірність швидкості поширення сигналу може призвести до того, що тактовий імпульс прийде настільки пізніше або раніше відповідного сигналу даних, що біт даних буде пропущений або полічений повторно. Іншою причиною, за якою у мережах відмовляються від використання тактуючих імпульсів, є економія провідників у дорогих кабелях.

Тому в мережах застосовуються так звані коди, що самосинхронізуються, сигнали яких несуть для передавача вказівки про те, у який момент часу потрібно здійснювати розпізнавання чергового біта (або декількох бітів, якщо код орієнтований більш ніж на два стани сигналу). Будь-який різкий перепад сигналу, так званий фронт, може служити вказівкою для синхронізації приймача з передавачем.

Найпоширеніші такі коди:

1***) NRZ*** (*Non-Return to Zero,* без повернення до нуля) – потенційний код, стан якого прямо або інверсно відображає значення біта даних;

2**) диференціальний *NRZ*** – стан змінюється на початку бітового інтервалу для “1” і не змінюється при “0”;

3) ***NRZI*** (*Non-Return to Zero Inverted,* без повернення до нуля з інверсією) – стан змінюється на початку бітового інтервалу при передачі “0” і не змінюється при передачі “1”. Використовується в *FDDI*, 100*BaseFX*;

4) ***RZ*** (*Return to Zero* , з поверненням до нуля) – біполярний імпульсний код, який самосинхронізується і подає “1” і “0” імпульсами протилежної полярності, які тривають половину такту (в другу половину такту стан встановлюється в нуль); усього використовується три стани;

5) ***AMI*** (*Bipolar Alternate Mark Inversion,* біполярне кодування з альтернативною інверсією) – використовує три стани: “0” , “+” “ і” для кодування логічного нуля використовується стан 0, а логічна одиниця кодується по черзі станами “+” “ і”. Використовується в *ISDN*, *DSx*;

6) **манчестерське кодування** (*manchester encoding)* – двофазне полярне кодування, що самосинхронізується. Логічна одиниця кодується перепадом потенціалу в середині такту від низького рівня до високого, логічний нуль – зворотним перепадом (якщо необхідно подати два однакових значення підряд, на початку такту відбувається додатковий службовий перепад потенціалу). Використовується в *Ethernet*;

7) **диференціальне манчестерське кодування** (*differential manchester encoding*) – двофазне полярне кодування, що самосинхронізується. Логічний нуль кодується наявністю перепаду потенціалу на початку такту, а логічна одиниця – відсутністю перепаду; у середині такту перепад є завжди (для синхронізації). В *Token Ring* застосовується модифікація цього методу, крім “0” і “1”, що використовує службові біти “*J*” і “*K*”, які не мають перепаду в середині такту (“*J*” не має перепаду на початку такту, “*K*” – має);

8) ***MLT*-3** – трирівневе кодування зі скремблюванням без самосинхронізації, логічний нуль кодується збереженням стану, а логічна одиниця кодується по черзі наступними станами: “+V ” , “0”, “-V”, “0’, “+V” і т.д. Використовується в *FDDI* і 100*BaseTX*;

9) ***PAM*5** (*Pulse Amplitude Modulation*) – п’ятирівневе біполярне кодування, при якому кожна пара бітів даних подається одним з п’яти рівнів потенціалу. Застосовується в 1000*BaseT*;

10) **2*B*1*Q*** (2 *Binary* 1 *Quarternary*) – пари бітів даних подаються одним четвертинним символом, тобто одним із чотирьох рівнів потенціалу. Застосовується в *ISDN*;

Способи цифрового кодування даних наведені на рис. 5.8

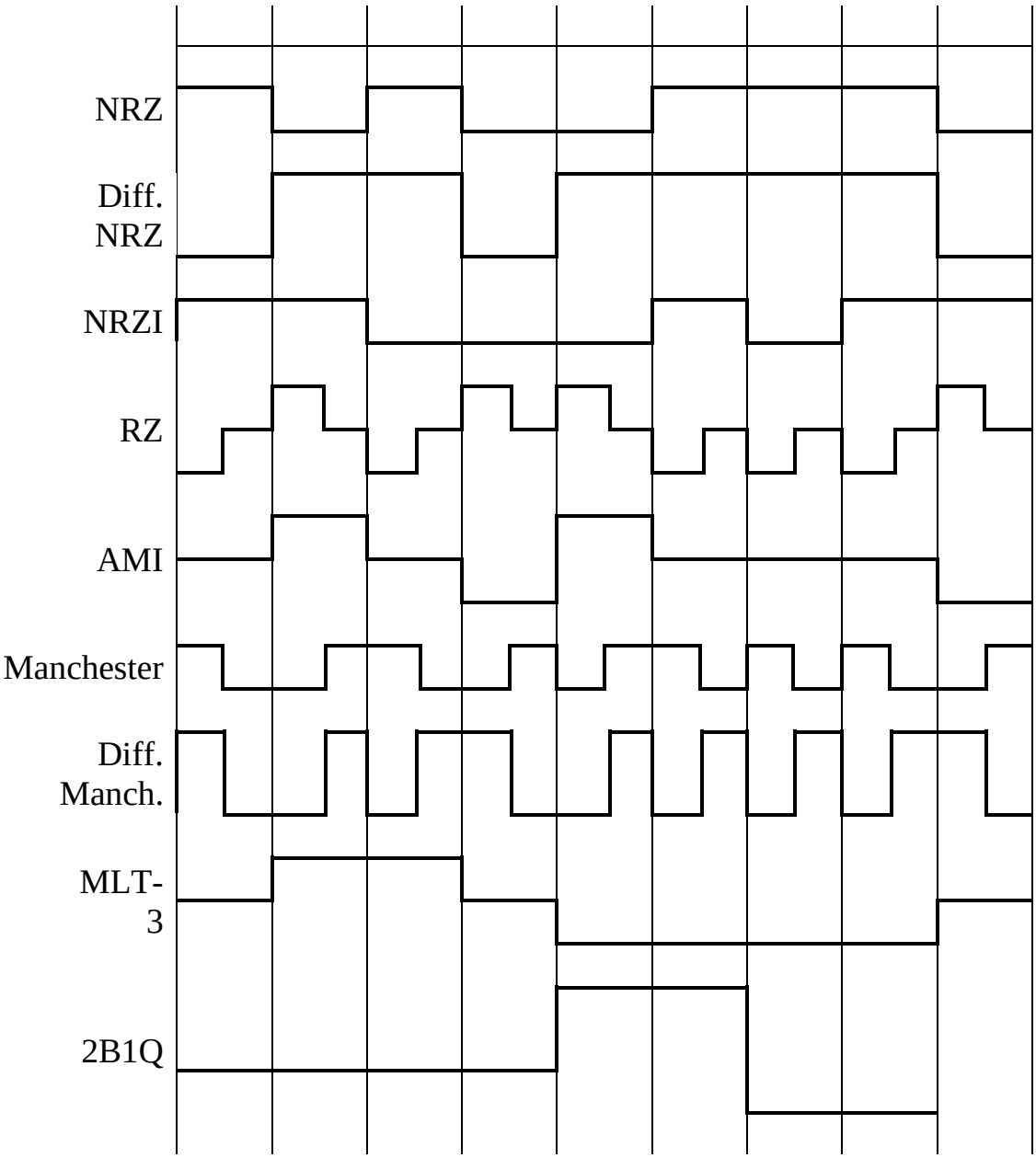

Рис. 5.8 – Способи цифрового кодування даних

5.3 Логічне кодування

Деякі різновиди цифрового кодування дуже чутливі до характеру переданих даних. Наприклад, при передачі довгих послідовностей логічних нулів за допомогою потенційного коду типу *NRZ* або *AMI* сигнал на лінії довгий час не змінюється, і приймач може помилитися з моментом зчитування чергового біта. Для коду *NRZ* подібні проблеми виникають і при передачі довгих послідовностей логічних одиниць. Логічне кодування (якому може піддаватися вихідна послідовність даних) повинно впроваджувати в довгі послідовності бітів біти із протилежним значенням або взагалі заміняти їх іншими послідовностями. Крім виключення

“проблемних” бітових послідовностей, логічне кодування дозволяє також збільшити кодову відстань між символами (для спрощення декодування), поліпшити спектральні характеристики сигналу, а крім того, передавати в загальному потоці службові сигнали.

В основному для логічного кодування застосовуються три групи методів:

1) вставка бітів;
2) надлишкове кодування;
3) скремблювання.

Вставка бітів (*bit stuffing*) – найбільш прямолінійний спосіб виключення довгих послідовностей, наприклад, логічних одиниць. Якщо в переданій послідовності зустрічається безперервний ланцюжок “1”, то передавач вставляє “0” після кожної, наприклад, п’ятої “1”. Приймач відкидає всі ці зайві “0”, які зустрічаються після п’яти “1”. Зрозуміло, можна проводити і зворотну операцію – вставку “1” у довгі послідовності “0”. Схема вставки бітів застосовується, наприклад, у протоколі *HDLC*.

Надлишкове кодування основане на розбивці вихідної послідовності біт на ділянки однакової довжини – символи. Потім кожний символ заміняється (як правило, табличним способом) на новий, що має або більшу кількість бітів, або іншу основу системи числення (наприклад, на символ, що складається із трійкових розрядів).

Серед логічних кодів можна виділити:

1. **Код 4B/5B** (табл. 5.1) замінює кожні 4 біти вхідного потоку (вихідний символ) на 5-бітний вихідний символ. Тому що кількість різних 5-бітних символів дорівнює 32, а вихідні символи можуть містити лише одну з 16-бітових комбінацій (тобто надмірність дорівнює 32/16 = 2), серед можливих вихідних кодів можна відібрати 16 “зручних” комбінацій, що не містять великої кількості нулів (більше трьох підряд), серед кодів, що залишилися, виділити службові символи (для підтримки синхронізації, виділення границь кадрів і їхніх полів і т.д.), а коди, що залишилися, вважати забороненими.

Таблиця 5.1– Код 4B/5B

Вхідний символ	Вихідний символ	Вхідний символ	Вихідний символ
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

Накладні витрати при кодуванні 4B/5B становлять 25 % (один зайвий біт на чотири біти даних), відповідно для досягнення тієї ж пропускної здатності, що і без логічного кодування, передавач повинен працювати на підвищеній на 25 % частоті. Код 4B/5B використовується в *FDDI* і *Fast Ethernet* (100*BaseFX* і 100*BaseTX*).

2. **Код 8B/10B** заміняє кожний 8-бітний вихідний символ 10-бітним вихідним символом. При тому ж рівні накладних витрат (25 %), що у випадку коду 4B/5B, має 4-кратну надмірність (1024 вихідних символів і 256 вихідних символів). При кодуванні 8B/10B кожному вихідному символу відповідає два вихідних символи, вибір з яких здійснюється залежно від останнього біта попереднього переданого символу. У результаті код забезпечує стабільне співвідношення “0” і “1” у вихідному потоці, незалежно від вихідних даних. Ця властивість важлива для лазерних передавачів, оскільки від даного співвідношення залежить їхнє нагрівання і кількість помилок прийому. Код 8B/10B використовується в *Gigabit Ethernet* (1000*BaseSX,* 1000*BaseLX,* 1000*BaseCX*).

3. **Код 8B/6T** кодує кожні 8 біт вихідної інформації шістьма трійковими:

(T – *ternary*, трійковий) розрядами, що приймають значення {“+”, “0”, “–”}. Наприклад, “00000000” = “+–00+–“, “11111110” = “–+0+00”. Надмірність коду 8B/6T вище, ніж у коду 4B/5B і становить 36/28 = 729/256 = 2,85. Застосовується в *Fast Ethernet* (100*BaseT*4).

Скремблювання полягає в побітному обчисленні вихідної послідовності на основі значень бітів вихідної послідовності та вже обчислених бітів результату.

Наприклад, скремблер може обчислювати для кожного біта наступний вираз: $B_i=A_i \oplus B_{i-3} \oplus B_{i-5}$, де A_i – і-й біт вихідної послідовності, B_i – і-й біт результату скремблювання, $\oplus$ – операція додавання за модулем два.

Так, для вихідної послідовності 110110000001 скремблер дасть наступний результуючий код: $B_1 = A_1 = 1$ (перші три цифри результуючого коду будуть збігатися з вихідним, тому що ще немає потрібних попередніх цифр).

$B_2 = A_2 = 1$; $B_3 = A_3 = 0$; $B_4 = A_4 \oplus B_1 = 0$; $B_5 = A_5 \oplus B_2 = 0$; $B_6 = A_6 \oplus B_3 \oplus B_1 = 1$ і т.д.

Таким чином, на виході скремблера з’явиться послідовність 110001101111, у якій немає послідовності із шести нулів, які були присутніми у вихідному коді.

Після одержання результуючої послідовності приймач передає її дескремблеру, що відновлює вихідну послідовність на основі зворотного співвідношення: $C_i=B_i \oplus B_{i-3} \oplus B_{i-5} = (A_i \oplus B_{i-3} \oplus B_{i-5}) \oplus B_{i-3} \oplus B_{i-5} = A_i$.

Різні алгоритми скремблювання відрізняються різною кількістю доданків і різним зсувом між ними (у наведеному вище прикладі використовується два доданки зі зсувом 3 і 5). Наприклад, в *ISDN*

використовуються два варіанти скремблювання: зі зсувом 5 і 23 і зі зсувом 18 і 23.

Існують спеціальні методи скремблювання, які застосовуються спільно з певними методами фізичного кодування. Наприклад, для поліпшення коду *AMI* застосовуються методи *B*8*ZS* і *HDB*3.

Метод *B*8*ZS* (*Bipolar with* 8-*Zeros Substitution*, біполярний із заміною 8 нулів) заміняє послідовності, що складаються з 8 нулів на "000V10V1", де V – сигнал одиниці забороненої в даному такті полярності, а 1 – сигнал одиниці коректної полярності. Якщо на 8 тактах приймач спостерігає три початкових нулі і два перекручування полярності, то він заміняє ці 8 бітів на 8 логічних нулів.

Метод *HDB*3 (*High-Density Bipolar* 3-*Zeros*, біполярний трьох-нульовий високої щільності) заміняє послідовності із чотирьох нулів, що ідуть підряд, на один із чотирьох чотирирозрядних біполярних кодів залежно від передісторії – полярності попереднього імпульсу і попередньої заміни.

Контрольні запитання

1. Для чого використовуються узгодження, екранування і гальванічна розв'язка ліній зв'язку?
2. Що таке кодування сигналів?
3. Які вимоги ставляться до методів цифрового кодування?
4. Що таке логічне кодування?
5. Які групи методів логічного кодування використовуються в комп'ютерних мережах?

6 ПЕРЕДАЧА ДАНИХ НА КАНАЛЬНОМУ РІВНІ. ПРОТОКОЛИ *SLIP, HDLC, PPP*. МЕТОДИ ПОВТОРНОЇ ПЕРЕДАЧІ. ПРОТОКОЛИ *ABP, GBN, SRP*. ОЦІНКА ЕФЕКТИВНОСТІ ПЕРЕДАЧ

На рівні каналу даних вирішується ряд проблем, які властиві тільки цьому рівню:

- реалізація сервісу для мережного рівня;
- об'єднання бітів, що надходять із фізичного рівня, у кадри;
- обробка помилок передачі;
- керування потоком кадрів.

Основне завдання канального рівня – забезпечити сервіс мережному рівню. Канальний рівень може забезпечувати різні класи сервісу. Три загальні класи сервісу:

- сервіс без повідомлення і без з'єднання;
- сервіс із повідомленням і без з'єднання;
- сервіс із повідомленням і з'єднанням.

Сервіс без повідомлення і без з'єднання не потребує підтвердження прийому переданого кадру і припускає, що до початку передачі повинно встановлюватися з'єднання, а після передачі – розриватися. Якщо в результаті перешкод на фізичному рівні кадр буде загублений, то ніяких спроб на канальному рівні його відновити не буде. Цей клас сервісу використовується там, де фізичний рівень забезпечує високу надійність при передачі. У цьому випадку відновлення при втраті кадрів можливо покласти на верхні рівні. Цей клас сервісу також застосовується при передачі даних у реальному часі там, де краще втратити частину даних, чим збільшити затримку їх доставки. Наприклад, передача мови. Більшість ЛОМ використовує цей клас сервісу на канальному рівні.

Наступний клас сервісу **– повідомлення без з'єднання.** У цьому класі одержання кожного відправленого кадру повинно бути підтверджене. Якщо підтвердження не прийшло протягом певного часу, то кадр повинен бути посланий знову. Цей клас сервісу використовується в ненадійному фізичному середовищі передачі, наприклад, при безпровідному зв'язку.

Найбільш складним є клас сервісу на канальному рівні – **сервіс із повідомленням і з'єднанням.** Цей клас сервісу припускає, що до початку передачі між машинами встановлюється з'єднання, і дані передаються за цим з'єднанням. Кожний переданий кадр нумерується, і канальний рівень гарантує, що кадр буде обов'язково отриманий і тільки один раз, і всі кадри будуть отримані в належній послідовності. При сервісі без з'єднання цього гарантувати неможливо тому, що втрата повідомлення про одержання кадру приведе до його пересилання так, що може з'явитися кілька ідентичних кадрів.

При сервісі з повідомленням і з'єднанням передача розбивається на три етапи. На першому етапі встановлюють з'єднання: на обох машинах ініціюють лічильники, що відслідковують, які кадри були прийняті, а які ні. На другому етапі передають один або кілька кадрів. На третьому –

з’єднання розривають: змінні, лічильники, буфери та інші ресурси, використані для підтримки з’єднання, звільняються.

6.1 Розбивання на кадри

Сервіс, створюваний канальним рівнем для мережного, опирається на сервіс, створюваний фізичним рівнем. На фізичному рівні протікають потоки бітів. Послана кількість бітів не обов’язково дорівнює прийнятій, значення посланого біта так само не обов’язково дорівнює прийнятому. Тому потрібні спеціальні зусилля на канальному рівні з виявлення і виправлення помилок.

Типовий підхід до рішення цієї проблеми – розбивання потоку бітів на кадри, підрахунок контрольної суми для кожного кадру при посиланні даних. При прийманні, контрольна сума обчислюється для кожного кадру заново і дорівнює тій, що зберігається в кадрі. Якщо вони різняться, то це є ознакою помилки передачі. Канальний рівень повинен вжити заходів щодо виправлення помилки, наприклад, скинути поганий кадр, надіслати повідомлення про помилку тому, хто надіслав цей кадр.

6.2 Виявлення помилок

Для рішення проблеми попадання кадрів на мережний рівень за призначенням і у належній послідовності встановлюється зворотний зв’язок між відправником і одержувачем у вигляді кадру підтвердження. Якщо кадр-підтвердження несе позитивну інформацію, то вважається, що передані кадри пройшли нормально, якщо там є повідомлення про помилку, то передані кадри треба передати заново.

Однак можливі випадки, коли через помилки в каналі кадр зникне цілком. У цьому випадку одержувач ніяк не буде реагувати, а відправник буде як завгодно довго чекати підтвердження. Для рішення цієї проблеми на канальному рівні вводять таймери. Таймер – це лічильник, що збільшує або зменшує своє значення на одиницю автоматично при одержанні такту імпульсу. Коли значення цього лічильника досягає заздалегідь певного значення або нуля, виникає переривання. Як тільки канальний рівень передає черговий кадр на фізичний, то одночасно він встановлює таймер на певний час. Цього часу повинно вистачати на те, щоб одержувач отримав кадр, а відправник отримав підтвердження.

Якщо відправник не одержить підтвердження раніше, ніж мине час, встановлений на таймері, то він буде вважати, що кадр загублено і повторить його передачу ще раз.

Однак, якщо кадр підтвердження був загублений, то цілком можливо, що той самий кадр одержувач оримає двічі. Як бути? Щоб розв’язати цю проблему кожному кадру присвоюють порядковий номер. За допомогою цього номера одержувач може виявити дублі.

Отже, таймери і нумерація кадрів – це основні засоби канального рівня, який забезпечує доставку кожного кадру мережному рівні в точку призначення в єдиному екземплярі.

Специфіка локальних мереж також знайшла своє відбиття в поділі канального рівня на два підрвіні, які часто називають також рівнями. Канальний рівень (*Data Link Layer*) ділиться в локальних мережах на два підрівні:

1) логічної передачі даних (*Logical Link Control, LLC*);

2) керування доступом до середовища (*Media Access Control, MAC*).

Рівень *MAC* з'явився через існування в локальних мережах середовища передачі даних, що розділяється поміж декількома вузлами. Саме цей рівень забезпечує коректне спільне використання загального середовища, надаючи йому відповідно до певного алгоритму в розпорядження ту або іншу станції мережі. Після того як доступ до середовища отриманий, ним може користуватися більш високий рівень – рівень *LLC*, який організує передачу логічних одиниць даних, кадрів інформації, з різним рівнем якості транспортних послуг. У сучасних локальних мережах набули поширення кілька протоколів рівня *MAC*, які реалізують різні алгоритми доступу до середовища, що розділяється. Ці протоколи повністю визначають специфіку таких технологій, як *Ethernet, Fast Ethernet, Gigabit Ethernet, Token Ring, FDDI, l00VG-AnyLAN*.

Рівень *LLC* відповідає за передачу кадрів даних між вузлами з різним ступенем надійності, а також реалізує функції інтерфейсу із прилягаючим до нього мережним рівнем. Саме через рівень *LLC* мережний протокол запитує в канального рівня потрібну йому транспортну операцію з потрібною якістю. На рівні *LLC* існує кілька режимів роботи, які відрізняються наявністю або відсутністю на цьому рівні процедур відновлення кадрів у випадку їхньої втрати або перекручування, тобто транспортних послуг, що відрізняються якістю цього рівня.

Протоколи рівнів *MAC* і *LLC* взаємно незалежні. Кожний протокол рівня *MAC* може застосовуватися з будь-яким протоколом рівня *LLC* і навпаки.

6.3 Протокол *HDLC*

HDLC – протокол високорівневого керування каналом передачі даних, є опублікованим *ISO* стандартом і базовим для побудови інших протоколів канального рівня (*SDLC, LAP, LAPB, LAPD, LAPX* і *LLC*). Він реалізує механізм керування потоком за допомогою безперервного *ARQ* і має необов'язкові можливості (опції), які підтримують напівдуплексну і повнодуплексну передачу, одноточкову і багатоточкову конфігурації, а також канали, що комутуються та не комутуються.

Керування потоком в *HDLC* здійснюється за допомогою передавальних і приймаючих вікон. Вікно встановлюється на кожному кінці каналу зв'язку, щоб забезпечити резервування ресурсів обох станцій.

Цими ресурсами можуть бути ресурси обчислювача або простір буфера. У більшості випадків вікно забезпечує і буферний простір, і правила нумерації (повідомлень).

Вікно встановлюється під час ініціювання сеансу зв'язку між станціями. Якщо станція А і станція В повинні обмінятися даними, А резервує вікно для В, а В резервує вікно для А. Використання вікон необхідно для повнодуплексних протоколів, які передають безперервний потік кадрів у приймаючий вузол без періодичних підтверджень із зупинкою і очікуванням.

6.4 Протоколи *SLIP, PPP*

Протокол *SLIP* (*Serial Line IP, RFC*-1055) – це найпростіший спосіб інкапсуляції IP-дейтаграм для послідовних каналів зв'язку.

Цей протокол став популярним завдяки можливостям підключення домашніх персональних машин до мережі Інтернет через порт *RS*-232, який з'єднаний з модемом. IP-дейтаграма у випадку *SLIP* повинна завершуватися спеціальним символом *0xC0*, який називається кінцевим. У багатьох реалізаціях дейтаграма і починається із цього символу. Якщо якийсь байт дейтаграми дорівнює кінцевому символу, то замість нього передається двобайтова послідовність *0xDB, 0xDC*. Октет *0xDB* виконує в *SLIP* функцію *ESC*-символу. Якщо ж байт дейтаграми дорівнює *0xDB*, то замість нього передається послідовність *0xDB, 0xDD*.

Використання протоколу *SLIP* припускає виконання ряду умов:

1. Кожний партнер обміну повинен знати *IP*-адресу свого адресата, тому що не існує методу обміну такого роду інформацією.
2. *SLIP* на відміну від *Ethernet* не використовує контрольних сум, тому виявлення і корекція помилок цілком лягає на програмне забезпечення верхніх рівнів.
3. Кадр *SLIP* не має поля тип, тому його не можливо використовувати, на відміну від кадрів *Ethernet*, для реалізації інших протоколів методом інкапсуляції.

Вперше протокол *SLIP* був застосований в 1984 році в ОС 4.2 *BSD*. Швидкість передачі інформації при використанні протоколу *SLIP* не перевищує 19.2 Кб/с, що звичайно досить для інтерактивного обміну в рамках протоколів *Telnet* або *RLOGIN*. Максимальний розмір переданого блоку (*MTU*) для *SLIP* лежить поблизу 256-512 байт, що забезпечує розумний компроміс між значенням затримки відгуку (~256 мс.) і ефективністю використання каналу (~98 % для *CSLIP*). При цьому для передачі одного символу (натиснута клавіша) використовується 20 байт заголовка в IP-дейтаграмі і 20 байт *TCP*-заголовка. Якщо врахувати витрати формування *SLIP*-кадру, то накладні витрати перевершують 40 байт.

Частково цей недолік усунутий у новій версії *CSLIP* (*Compressed SLIP, RFC*-1144, яка запропонована Джекобсоном в 1990 році). В *CSLIP*

заголовок скорочується до 3–5 байт (проти 40 в *SLIP*). Ця версія протоколу здатна підтримувати до 16 *TCP*-з'єднань на кожному з кінців послідовного каналу. Багато сучасних *SLIP*-драйверів підтримують і *CSLIP*.

6.5 Протокол *PPP (Point-to-Point Protocol)*

Однією із причин малого числа каналів зв'язку *IP* з безпосереднім з'єднанням була відсутність стандартного протоколу формування пакета даних *Internet*. Протокол *Point-to-Point Protocol (PPP)* (протокол каналу зв'язку з безпосереднім з'єднанням) вирішує цю проблему.

Крім розв'язування проблеми формування стандартних пакетів даних *Internet IP* у каналах з безпосереднім з'єднанням, *PPP* також повинен був вирішити інші проблеми, у тому числі присвоєння і керування адресами *IP*, асинхронне (старт/стоп) і синхронне біт-орієнтоване формування пакета даних, мультиплексування протоколу мережі, конфігурацію каналу зв'язку, перевірку якості каналу зв'язку, виявлення помилок і узгодження варіанта для таких здатностей, як узгодження адреси мережного рівня і узгодження компресії інформації.

PPP вирішує ці питання шляхом забезпечення розширюваного протоколу керування каналом (*Link Control Protocol) (LCP)* і сімейства протоколів керування мережею *(Network Control Protocols) (NCP)*, які дозволяють погоджувати факультативні параметри конфігурації і різні можливості.

Сьогодні *PPP*, крім *IP*, забезпечує також і інші протоколи, у тому числі *IPX* і *DECnet* .

На відміну від *SLIP*-протоколу, *PPP* може працювати через будь-який інтерфейс *DTE/DCE* (наприклад, *EIA RS*-232-*C, EIA RS*-422, *EIA RS*-423 і *CCITT* V.35).

Протокол *PPP* досить невибагливий і може працювати без керуючих сигналів модемів (таких, як *Request to Send, Clear to Send, Data Carrier Detect* і *Data Terminal Ready*). Єдиною абсолютною вимогою, яку ставить *PPP*, є вимога забезпечення дубльованих схем (або спеціально призначених, або тих, що перемикаються), які можуть працювати як у синхронному, так і в асинхронному послідовному бітовому режимі, прозорому для блоків даних канального рівня *PPP*.

PPP не висуває яких-небудь обмежень, які стосуються швидкості передачі інформації, крім тих, які визначаються конкретним застосованим інтерфейсом *DTE/DCE*.

PPP забезпечує метод передачі дейтаграм через послідовні канали зв'язку з безпосереднім з'єднанням. Він містить три основних компоненти:

1. Метод формування дейтаграм для передачі послідовними каналами. *PPP* використовує протокол *High-level Data Link Control*

(HDLC) (протокол керування каналом передачі даних високого рівня) як базис для формування дейтаграм при проходженні через канали з безпосереднім з'єднанням.

2. Розширюваний протокол *LCP* (*Link Control Protocol*) для організації, вибору конфігурації і перевірки з'єднання каналу передачі даних.

3. Сімейство протоколів *NCP* **(***Network Control Protocols*) для організації і вибору конфігурації різних протоколів мережного рівня. *PPP* призначений для забезпечення одночасного користування множиною протоколів мережного рівня.

У порівнянні із протоколом *SLIP* протокол *PPP* є значно більш розвиненим інструментом для роботи на послідовних лініях і має такі переваги:

- можливість одночасної роботи з різними мережними протоколами, а не тільки по *IP*;

- перевірка цілісності даних шляхом підрахунку контрольної суми;

- підтримка динамічного обміну адресами *IP*;

- можливість стиснення заголовків *IP*- і *TCP*-п*акетів*, розроблених *Van Jacobson* (механізм схожий на реалізований у протоколі *CSLIP*).

6.6 Протокол керування каналу зв'язку *PPP* (*LCP*)

LCP забезпечує метод організації, вибору конфігурації, підтримки і закінчення роботи каналу з безпосереднім з'єднанням. Процес *LCP* проходить через 4 фази, що чітко розрізняються:

1. Організація каналу і узгодження його конфігурації. Перш ніж буде зроблений обмін яких-небудь дейтаграм мережного рівня (наприклад *IP*), *LCP* спочатку повинен відкрити зв'язок і погодити параметри конфігурації. Ця фаза завершується після того, як пакет підтвердження конфігурації буде відправлений і прийнятий;

2. Визначення якості каналу зв'язку. У цій фазі перевіряється канал, щоб визначити, чи є якість каналу достатньою для виклику протоколів мережного рівня. Ця фаза є повністю факультативною. *LCP* може затримати передачу інформації протоколів мережного рівня до завершення цієї фази.

3. Узгодження конфігурації протоколів мережного рівня. Після того як *LCP* завершить фазу визначення якості каналу зв'язку конфігурація мережних протоколів може бути окремо обрана відповідними *NCP*, і вони можуть бути в будь-який момент викликані і звільнені для наступного використання. Якщо *LCP* закриває даний канал, він інформує про це протоколи мережного рівня, для того щоб вони могли вжити відповідних заходів.

4. Припинення дії каналу. *LCP* може в будь-який момент закрити канал. Це звичайно робиться за запитом користувача (людини), але може відбутися і через яку-небудь фізичну подію, таку, як втрата носія або закінчення періоду бездіяльності таймера.

Існує три класи пакетів *LCP*:

1) пакети для організації каналу зв'язку. Використовуються для організації і вибору конфігурації каналу;

2) пакети для завершення дії каналу. Використовуються для завершення дії каналу зв'язку;

3) пакети для підтримки працездатності каналу. Використовуються для підтримки і налагодження каналу.

6.7 Методи повторної передачі (*ARQ)*

Можливий ряд варіантів механізму *ARQ*. Кожний правильно прийнятий кадр може бути підтверджений окремим спеціальним кадром, або підтвердження може бути вставлене в поле керування інформаційних кадрів, які переносять дані у зворотному напрямку. В останньому випадку також повинні застосовуватися спеціальні кадри підтвердження, оскільки інформаційного кадру в потрібний момент може не виявитися.

Існує два види підтвердження приймання: позитивне (*ACK*) і негативне (*NACK* або *NAK*). Але в кожному разі, щоб уникнути перевантажень, повинні застосовуватися перерви. Передавальна сторона, що не одержала відповіді (*ACK* або *NACK*) протягом заданого проміжку часу після передачі, повторює відповідний кадр. Щоб організувати процедуру перерви, кадри повинні зберігатися в накопичувачі передавальної сторони до одержання підтвердження правильності передачі.

Існує три основні способи обробки відповідей на позитивні і негативні підтвердження:

1) стартостопний, або передача із зупинкою і очікуванням (*SAW - Stop And Wait),* який часто називають блоковим методом передачі (*ABP*).

2) повернення на *N* кадрів (*GBN – Go Back N),* який також називають потоковим методом передачі.

3) метод вибіркового (селективного) повтору (*SR – Selective Repeat*).

6.8 Процедура *SAW* (*ABP*)

Відповідно до цієї процедури без підтвердження може бути переданий тільки один кадр. Після передачі чергового кадру передавальна сторона чекає підтвердження. Якщо надходить негативне підтвердження або відбудеться перевищення часу тайм-ауту, кадр передається повторно. Кадр скидається (стирається) з накопичувача передавача лише після

одержання позитивного підтвердження. Тимчасова діаграма роботи процедури *ARQ* типу *SAW* зображена на рис. 6.1.

Дану процедуру зручно використовувати при напівдуплексному зв'язку, коли передача сторін чергується. Однак вона неефективна у випадку організації повнодуплексного зв'язку, особливо, якщо час поширення сигналу за каналом значно більше часу передачі кадру, що типово для супутникових і ряду інших каналів.

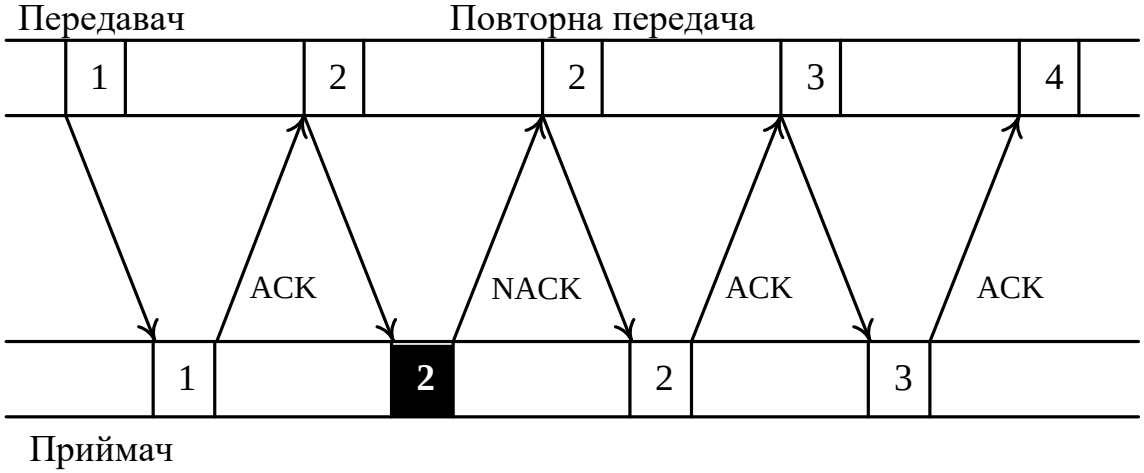

Рисунок 6.1 – Передача кадрів відповідно до процедури *SAW* (*ABP*)

Якщо час поширення дуже малий (при невеликій довжині каналу або через низьку швидкість передачі), процедура *SAW* не призведе до серйозного зниження продуктивності всієї системи.

6.9 Процедура *GBN*

У цьому випадку кадри передаються безупинно без очікування підтвердження прийому певної кількості кадрів. При одержанні негативного підтвердження або після закінчення встановленого часу очікування непідтверджений і всі наступні кадри передаються повторно. Приклад такої передачі за процедурою *GBN* поданий на рис. 6.2,

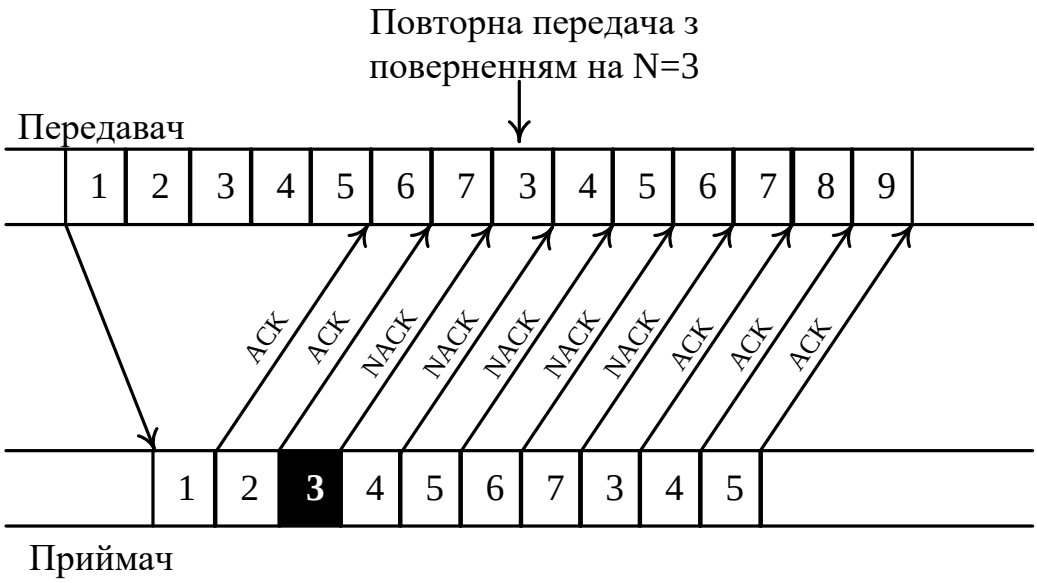

Рисунок 6.2 – Передача кадрів відповідно до процедури *GBN*

де *N* – затримка кругового поширення, тобто проміжок часу від моменту початку передачі кадру до моменту одержання підтвердження на нього.

У практичних версіях процедур *GBN*, наприклад у складі протоколу V.42, не всі кадри вимагають підтвердження. Позитивне підтвердження може служити підтвердженням правильної передачі не тільки даного кадру, але і всіх попередніх.

Процедуру *GBN* часто називають *ARQ* типу *REJ* (*REJect*).

6.10 Процедура *SR*

Відповідно до процедури SR повторна передача даних здійснюється тільки для кадру, на який надійшло негативне підтвердження або минув час тайм-ауту підтвердження. Дана процедура, у порівнянні із процедурами SAW і GBN, істотно збільшує пропускну здатність СПД. Але для передачі і прийому кадрів, які мають різні номери, на прийомній стороні повинен бути буферний накопичувач із довільним доступом. Зі збільшенням затримки поширення сигналу в каналі зв’язку необхідно збільшувати буферну пам’ять. Очевидно, реалізація процедури *SR* є більш складною і дорогою. З цієї причини вона довго не могла знайти широкого комерційного застосування. Навіть у найбільш досконалому на сьогоднішній день протоколі V.42 процедура селективного повтору не є обов’язковою. Тимчасова діаграма передачі кадрів згідно з процедурою *SR* показана на рис. 6.3.

Спосіб *SR* часто називають *ARQ* типу *SREJ* (*Selective REJect*).

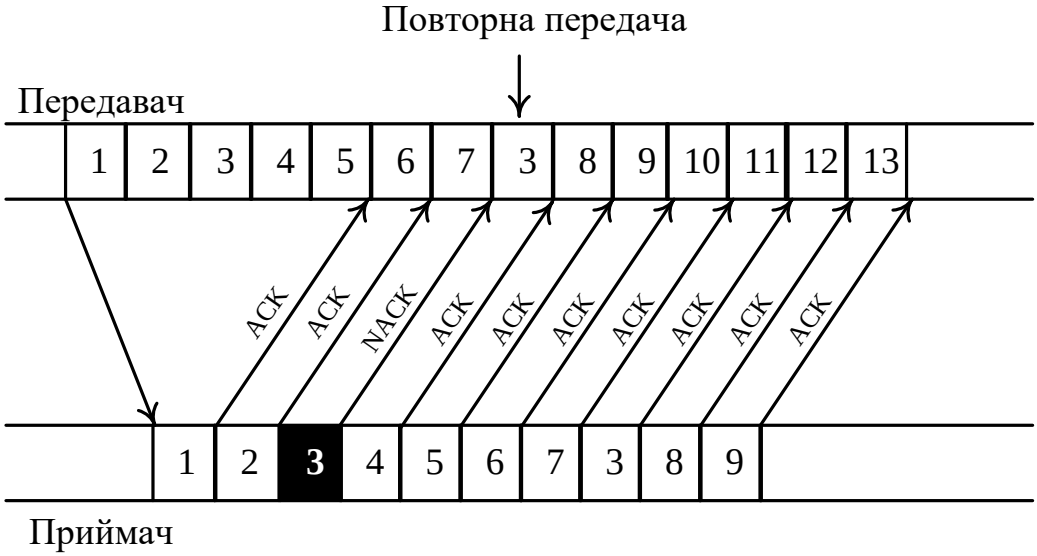

Рисунок 6.3. – Передача кадрів відповідно до процедури *SR*

Ефективність СПД зі схемою *ARQ* типу *SR* в ідеальному випадку залежить тільки від імовірності безпомилкового прийому кадрів, тобто від якості каналу зв'язку.

6.11 Розрахунок первинних параметрів

До первинних параметрів можливо віднести:

$t_{П}$ – час передачі кадру (с) від передавача до приймача;

$t_{Р}$ – час затримки (с) передачі біта від передавача до приймача;

T_{MIN} – мінімальний час передачі поточного кадру після одержання підтвердження (с).

Для цього застосовуються такі рівняння:

$$t_{П} = L_{П} / V_{П} \quad (6.1)$$

$$t_{Р} = \ell / C \quad (6.2)$$

$$T_{MIN} = t_{П} + 2t_{Р} + 2t_{О} + t_{АСК}, \quad (6.3)$$

де $t_{О}$ – час обробки кадру приймачем і передавачем; $t_{АСК}$ – час передачі відповідного кадру.

6.12 Ефективність при застосуванні протоколу *ABP*

Ефективність протоколу при відсутності помилок передачі (ймовірність $\rho = 0$):

$$\eta(ABP,0) = t_{П} / T_{MIN}. \quad (6.4)$$

Ефективність протоколу при наявності помилок:

$$\eta(ABP, \rho) = (1-\rho)\cdot t_{\text{ПMAX}}/((1-\rho)\cdot T_{MIN} + \rho \cdot T_{OUT}), \quad (6.5)$$

де ρ – імовірність помилки; T_{OUT} – час до повторної передачі кадру при відсутності підтвердження з боку приймача про прийом попереднього кадру.

Кінцеві рівняння розрахунку ефективності:
для повного дуплекса (*full-duplex — FD*) при $T_{OUT} = t_{\text{ПMAX}}$

$$\eta_{FD}(ABP, \rho) = (1-\rho)\cdot t_{\text{П}} /((1-\rho)\cdot T_{\text{MIN}} + \rho \cdot t_{\text{П. MAX}}); \quad (6.6)$$

для напівдуплекса (*half-duplex – HD*) при $T_{OUT} = T_{\text{MIN}}$

$$\eta_{HD}(ABP, \rho) = (1-\rho)\cdot t_{\text{П.MAX}}/ T_{\text{MIN}} \quad (6.7)$$

6.13 Ефективність при застосуванні протоколу *SRP*

$$\eta(SRP,0) = min\{W^* t_{\text{ПMAX}} / T_{\text{MIN}},1\}, \quad (6.8)$$

де W – ширина вікна, кількість кадрів, що передаються без підтвердження з боку приймача.

При W = ∞ час передачі кожного пакету буде дорівнювати $t_{П.MAX}/(1-\rho)$, тоді верхня границя ефективності при *SRP* буде дорівнювати

$$\eta_{\infty}(SRP, \rho) = 1- \rho. \quad (6.10)$$

Для випадку прямої і зворотної передачі (*round-trip – RT*) при $T_{MIN} = T_{OUT} = W^* t_{П}$ при виконанні умови, що $\rho^* W \le 10$ % маємо

$$\eta_{RT}(SRP, \rho) \approx (2 + \rho\cdot (W - 1))/ (2 + \rho\cdot (3W - 1)). \quad (6.11)$$

6.14 Ефективність при застосуванні протоколу *GBN*

Для протоколу *GBN* аналогічна ефективність при $W = T_{\text{MIN}} / t_{П}$ дорівнює

$$\eta_{RT}(GBN, \rho) = 1/(1 + \rho W/(1- \rho)). \quad (6.12)$$

При $\rho^* W \le 10$ % має місце:

$$\eta_{\text{RT}}(SRP, \rho) \approx \eta_{RT}(GBN, \rho) \approx 1 - \rho W. \quad (6.13)$$

Висновки:

– зростання ефективності протоколів обміну таке: *ABP*, *GBN*, *SRP*;

– при збільшенні розміру вікна W і відсутності помилок ефективність протоколів *SRP* і *GBN* збільшується в W разів;

– розмір вікна вибирається $W \le T_{\text{MIN}} / t_{\text{ПMAX}}$.

Контрольні запитання

1. Які класи сервіса забезпечує канальний рівень?
2. Для чого використовується розбивання на кадри на канальному рівні?
3. Що виконують протоколи *SLIP*, *HDLC*, *PPP*?
4. Які методи повторної передачі використовуються в комп'ютерних мережах?
5. Як розрахувати ефективність передач при використанні протоколів *ABP*, *GBN*, *SRP*?

7 ТЕХНОЛОГІЯ *ETHERNET*: СТАНДАРТИ, ДОСТУП, КАДРИ, АДРЕСАЦІЯ

Технологія *Ethernet* була розроблена в дослідницькому центрі компанії *Xerox* у середині 1970-х років. У 1980 році фірми *DEC, Intel* і *Xerox* випустили другу фірмову версію стандарту *Ethernet* (*Ethernet DIX* або *Ethernet* II). На основі *Ethernet DIX* був розроблений стандарт 802.3. Ці два стандарти дуже близькі, але є і деякі відмінності, наприклад, у форматі кадру. У теперешний час термін *Ethernet* зазвичай використовується для опису всіх локальних мереж, що використовують метод доступу з контролем несучої і виявленням колізій (*CSMA/CD, Carrier Sense Multiple Access/Collision Detection*).

7.1 Метод доступу CSMA/CD

Метод доступу *CSMA/CD* визначає, по-перше, у який спосіб станція визначає момент, коли вона може передати кадр, по-друге – як повинні поводитися станції у випадку одночасного початку передачі кадрів двома або більше вузлами.

Кожна станція постійно прослуховує мережу. Якщо в мережі присутній сигнал несучої частоти, це означає, що інша станція передає свій кадр. Для того щоб мати право передати кадр, станція повинна дочекатися “тиші” (відсутності несучої), виждати технологічну паузу (9.6 мкс) і, якщо за час паузи сигнал несучої не з’явився, почати передачу.

Всі станції, що прослуховують мережу, розпізнають переданий кадр, і та з них, чия адреса записана в поле одержувача, приймає кадр повністю і передає його протоколам верхніх рівнів. Інші станції “чужі” кадри повинні ігнорувати.

Можлива ситуація, коли дві станції одночасно починають передачу кадрів. Така ситуація називається **колізією** (*collision*). Настання колізії передавальна станція може визначити за відмінністю переданих і прийнятих нею даних (під час передачі кадру станція продовжує прослуховувати мережу). Станція, що виявила колізію, повинна припинити передачу кадру, передати в мережу спеціальний сигнал *затору* (*jam*), що складається з 32 бітів, і витримати паузу випадкової тривалості (обчисленої за спеціальним алгоритмом). Після цього вона може знову спробувати передати свій кадр (звичайно, дочекавшись “тиші” і зробивши технологічну паузу).

Інтервал часу до повторної спроби доступу після колізії визначається як випадкове число інтервалів відстрочки (один інтервал відстрочки дорівнює 512 бітовим інтервалам, тобто 51,2 мкс). Кількість інтервалів відстрочки визначається як випадкове ціле число, рівномірно розподілене в інтервалі $0..2^n$ ($1<=n<=10$) або $0..2^{10}$ ($10<n<=16$). Тут n – номер спроби передачі кадру. Якщо 16 спроб закінчуються невдало (породжуючи

колізії), підрівень *MAC* відкидає кадр і передає верхнім рівням повідомлення про помилку.

7.2 Формати кадрів Ethernet

У мережах *Ethernet* можуть застосовуватися кадри чотирьох форматів:

1. *Ethernet* II (*Ethernet DIX*).
2. *Ethernet* 802.2.
3. *Ethernet* 802.3.
4. *Ethernet SNAP*.

На рис.7.1 наведені формати кадрів (перший рядок – позначення полів, другий рядок – розміри полів у байтах).

Кадр *Ethernet* II

P	*DA*	*SA*	*Type*	*Data*	*FCS*
8	6	6	2	46 – 1500	4

Кадр *Ethernet* 802.2/*LLC*

P	*SFD*	*DA*	*SA*	*Length*	*DSAP*	*SSAP*	*Control*	*Data*	*FCS*
7	1	6	6	2	1	1	1/2	43/42 – 1497/1496	4

Кадр Ethernet 802.3 (“Raw”)

P	*SFD*	*DA*	*SA*	*Length*	*Data*	*FCS*
7	1	6	6	2	46 – 1500	4

Кадр Ethernet SNAP

P	*SFD*	*DA*	*SA*	*Length*	*DSAP (0xAA)*	*SSAP (0xAA)*	*Control (0x03)*	*ProtID*	*Data*	*FCS*
7	1	6	6	2	1	1	1	5	38 – 1492	4

Рисунок 7.1 – Формати кадрів *Ethernet*

Формати кадрів містять такі поля:

1) поле ***P*** (*Preamble*, преамбула) складається із семи байт 10101010 і використовується для синхронізації. Преамбула кадру *Ethernet* II містить також поле *SFD*;

2) поле ***SFD*** (*Start of Frame Delimiter*, роздільник початку кадру) має значення 10101011 і вказує на те, що наступний байт належить заголовку кадру;

3) поле ***DA*** (*Destination Address*, адреса призначення) містить адресу одного із трьох типів:

а) індивідуальна (*unicast*) адреса – перший біт старшого байта дорівнює 0, вказує на єдиного одержувача (являє собою його *MAC*-адресу); унікальність адреси забезпечують виробники мережного устаткування: у другому і третьому байті зберігається номер фірми-виготовлювача, а інші заповнюються виготовлювачем; деякі мережні адаптери дозволяють встановлювати для них довільну *MAC*-адресу;

б) широкомовна (*broadcast*) адреса – складається із усіх одиниць (0x*FFFFFFFFFFFF*), вказує на те, що даний кадр повинен бути отриманий всіма вузлами мережі;

в) групова (*multicast*) адреса – перший біт старшого байта дорівнює 1, в інших бітах зберігається номер групи вузлів, для яких призначений даний кадр.

4) поле ***SA*** (*Source Address*, адреса джерела) містить *MAC*-адресу відправника кадру (завжди є індивідуальною адресою);

5) поле ***Type*** (тип) вказує на протокол верхнього рівня, чиї дані передаються в кадрі (фактично, виконує функції полів *DSAP* і *SSAP* із заголовка кадру *LLC*);

6) поле ***Length*** (довжина) містить розмір поля *Data* (у байтах);

7) поле ***Data*** (дані) містить дані, передані протоколом верхнього рівня;

8) поле ***FCS*** (*Frame Check Sequence*, контрольна послідовність кадру) містить контрольну суму кадру, обчислену за алгоритмом *CRC*-32;

9) поля ***DSAP, SSAP*** і ***Control*** становлять заголовок LLC-*Кадру*;

10) поле ***ProtID*** (ідентифікатор протоколу) дозволяє використовувати кадри *Ethernet* для передачі даних більш широкої множини протоколів верхнього рівня. Це поле складається із двох підполів: трибайтового *OUI* (*Organizationally Unique Identifier*, організаційно-унікальний ідентифікатор), який зберігає номер організації, що контролює коди протоколів у другому (двобайтовому) підполі *Type* (тип). *IEEE* присвоєний *OUI* = 0x00000.

7.3 Специфікації фізичного середовища *Ethernet*

Історично перші мережні технології *Ethernet* використовували коаксіальний кабель діаметром 0,5 дюйма. Надалі були визначені і інші специфікації фізичного рівня для стандарту *Ethernet*, що дозволяють використовувати різні середовища передачі даних. Метод доступу *CSMA/CD* і всі тимчасові параметри залишаються тими ж для будь-якої специфікації фізичного середовища технології *Ethernet* 10 Мбіт/с.

Фізичні специфікації технології *Ethernet* на сьогоднішній день включають такі середовища передачі даних.

1. l0*Base*-5 – коаксіальний кабель діаметром 0,5 дюйма, який називається «товстим» коаксіалом. Має хвильовий опір 50 Ом. Максимальна довжина сегмента –500 метрів (без повторювачів).

2. l0*Base*-2 – коаксіальний кабель діаметром 0,25 дюйма, який називається «тонким» коаксіалом. Має хвильовий опір 50 Ом. Максимальна довжина сегмента – 185 метрів (без повторювачів).

3. l0*Base-T* – кабель на основі неекранованої крученої пари (*Unshielded Twisted Pair, UTP*). Утворює зіркоподібну топологію на основі концентратора. Відстань між концентратором і кінцевим вузлом повинна бути не більше 100 м.

4. l0*Base-F* – волоконно-оптичний кабель. Топологія аналогічна топології стандарту l0*Base-T*. Є кілька варіантів цієї специфікації: *FOIRL* (відстань до 1000 м), l0*Base-FL* (відстань до 2000 м), l0*Base-FB* (відстань до 2000 м).

Число 10 у зазначених вище назвах позначає бітову швидкість передачі даних цих стандартів – 10 Мбіт/с, а слово *Base* означає метод передачі на одній базовій частоті 10 МГц (на відміну від методів, що використовують кілька несучих частот, які називаються *Broadband* – широкосмуговими). Останній символ у назві стандарту фізичного рівня позначає тип кабелю.

7.4 Стандарт 10*Base*-5

Стандарт 10*Base*-5 в основному відповідає експериментальній мережі *Ethernet* фірми *Xerox* і може вважатися класичним *Ethernet*. Він використовує як середовище передачі даних коаксіальний кабель із хвильовим опором 50 Ом, діаметром центрального мідного проводу 2,17 мм і зовнішнім діаметром близько 10 мм («товстий» *Ethernet*). Такі характеристики мають кабелі марок *RG*-8, *RG*-11.

Кабель використовується як моноканал для всіх станцій. Сегмент кабелю має максимальну довжину 500 м (без повторювачів) і повинен мати на кінцях узгоджуючі термінатори опором 50 Ом, які поглинають сигнали, що поширюються кабелем і перешкоджають виникненню відбитих сигналів.

Станція повинна підключатися до кабелю за допомогою приймача-передавача – **трансивера** (*transmitter+receiver = transceiver*). Трансивер встановлюється безпосередньо на кабелі і живиться від мережного адаптера комп'ютера.

Трансивер з'єднується з мережним адаптером інтерфейсним кабелем *AUI* (*Attachment Unit Interface*) довжиною до 50 м, який складається з 4 кручених пар (адаптер повинен мати рознімання *AUI*). Наявність стандартного інтерфейсу між трансивером і іншою частиною мережного адаптера є дуже корисною при переході з одного типу кабелю на інший. Для цього досить тільки замінити трансивер, а інша частина мережного адаптера залишається незмінною, тому що вона відпрацьовує протокол рівня *MAC*. При цьому необхідно тільки, щоб новий трансивер (наприклад, трансивер для крученої пари) підтримував стандартний інтерфейс *AUI*. Для приєднання до інтерфейсу *AUI* використовується рознімання *DB*-15.

Допускається підключення до одного сегмента не більше 100 трансиверів, причому відстань між підключеннями трансиверів не повинна бути менше 2,5 м. На кабелі є розмітка через кожні 2,5 м, що позначає точки підключення трансиверів.

Трансивер – це частина мережного адаптера, яка виконує такі функції:

- приймання і передачу даних з кабелю на кабель;
- визначення колізій на кабелі;
- електричну розв'язку між кабелем і іншою частиною адаптера;
- захист кабелю від некоректної роботи адаптера.

Стандарт 10*Base*-5 визначає можливість використання в мережі спеціального пристрою – повторювача (*repeater*). Повторювач служить для об'єднання в одну мережу декількох сегментів кабелю і збільшення тим самим загальної довжини мережі.

Стандарт дозволяє використання в мережі не більше 4 повторювачів і відповідно не більше 5 сегментів кабелю. При максимальній довжині сегмента кабелю в 500 м це дає максимальну довжину мережі 10*Base*-5 в 2500 м. Тільки 3 сегменти з 5 можуть бути навантаженими, тобто такими, до яких підключаються кінцеві вузли. Між навантаженими сегментами повинні бути ненавантажені сегменти. Максимальна конфігурація мережі являє собою два навантажених крайніх сегменти, які з'єднуються ненавантаженими сегментами ще з одним центральним навантаженим сегментом.

Правило застосування повторювачів у мережі *Ethernet* 10*Base*-5 називаться **«правило 5-4-3»**: 5 сегментів, 4 повторювачі, 3 навантажених сегменти. Обмежене число повторювачів пояснюється додатковими затримками поширення сигналу, які вони вносять. Застосування повторювачів збільшує час подвійного поширення сигналу, яке для надійного розпізнавання колізій не повинне перевищувати час передачі кадру мінімальної довжини, тобто кадру в 72 байт або 576 бітів.

Кожний повторювач підключається до сегмента одним своїм трансивером, тому до навантажених сегментів можна підключити не більше 99 вузлів. Максимальне число кінцевих вузлів у мережі 10*Base*-5 становить 99x3 = 297 вузлів.

До **переваг стандарту** 10*Base*-5 належать:

- гарна захищеність кабелю від зовнішніх впливів;
- порівняно велика відстань між вузлами;
- можливість простого переміщення робочої станції в межах довжини кабелю AUI.

Недоліками 10*Base*-5 є:

- висока вартість кабелю;
- складність прокладки через велику його твердість;
- потреба в спеціальному інструменті для закладення кабелю;
- зупинка роботи всієї мережі при ушкодженні кабелю або поганому з'єднанні;

- необхідність заздалегідь передбачити підводку кабелю до всіх можливих місць установки комп'ютерів.

Структура мережі стандарту 10*Base*-5 наведена на рис. 7.2.

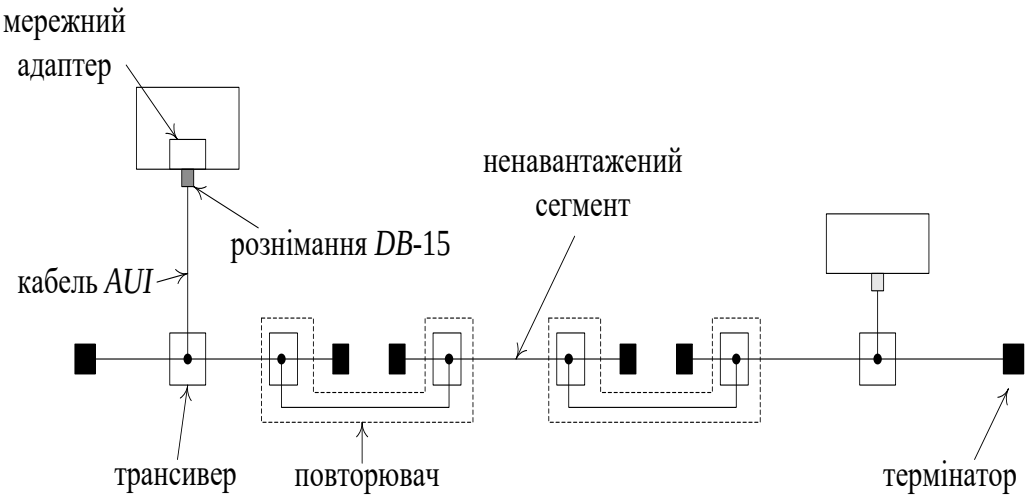

Рисунок 7.2 – Структура мережі стандарту 10*Base*-5

7.5 Стандарт 10Base-2

Стандарт 10*Base*-2 використовує як передавальне середовище коаксіальний кабель із діаметром центрального мідного проводу 0,89 мм і зовнішнім діаметром близько 5 мм («тонкий» *Ethernet*). Кабель має хвильовий опір 50 Ом. Такі характеристики мають кабелі марок *RG*-58/*U, RG*-58 *A/U, RG*-58 *C/U*.

Максимальна довжина сегмента без повторювачів становить 185 м, сегмент повинен мати на кінцях узгоджуючі термінатори 50 Ом. Тонкий коаксіальний кабель дешевше товстого, через що мережі 10*Base*-2 іноді називають мережами *Cheapernet* (від *cheaper* – більш дешевий). Але за дешевизну кабелю доводиться розплачуватися якістю – «тонкий» коаксіал має гіршу перешкодозахищеність, гіршу механічну міцність і більш вузьку смугу пропущення.

Станції підключаються до кабелю за допомогою високочастотного *BNC T-конектора*, що являє собою трійник, один відвід якого з'єднується з мережним адаптером, а два інших – із двома кінцями розриву кабелю. Максимальна кількість станцій, що підключаються до одного сегмента, – 30. Мінімальна відстань між станціями – 1м. Кабель «тонкого» коаксіала має розмітку для підключення вузлів із кроком в 1 м.

Стандарт 10*Base*-2 також передбачає використання повторювачів, застосування яких також повинне відповідати «правилу 5-4-3». У цьому випадку мережа буде мати максимальну довжину в 5х185 = 925 м.

Очевидно, що це обмеження є більш сильним, чим загальне обмеження в 2500 метрів.

Зауваження. Для побудови коректної мережі *Ethernet* потрібно витримати багато обмежень, причому деякі з них належать до тих самих параметрів мережі – наприклад, максимальна довжина або максимальна кількість комп'ютерів у мережі повинні задовольняти одночасно декілька різних умов. Коректна мережа *Ethernet* повинна відповідати всім вимогам, але на практиці потрібно задовольнити тільки найбільш тверді. Так, якщо в мережі *Ethernet* не повинно бути більше 1024 вузлів, а стандарт 10*Base*-2 обмежує число навантажених сегментів трьома, то загальна кількість вузлів у мережі 10*Base*-2 не повинне перевищувати 29x3 = 87.

Загальним недоліком стандартів 10*Base*-5 і 10*Base*-2 є відсутність оперативної інформації про стан моноканалу. Ушкодження кабелю виявляється відразу ж (мережа перестає працювати), але для пошуку відрізка кабелю, що відмовив, необхідний спеціальний прилад – кабельний тестер. Структура мережі стандарту 10*Base*-2 наведена на рис 7.3.

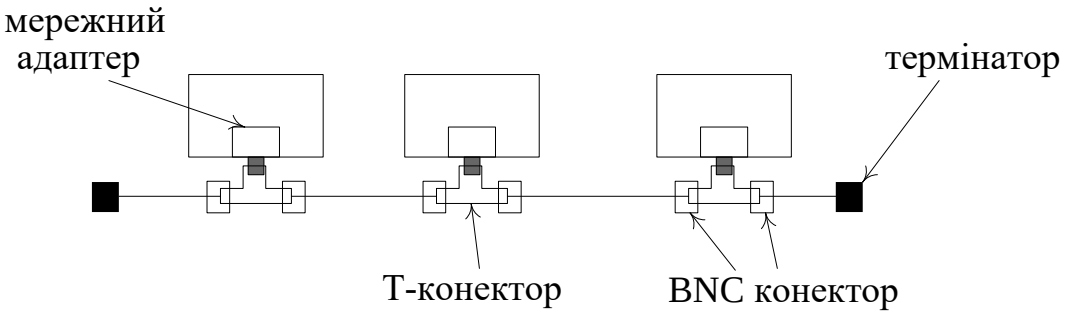

Рисунок – 7.3 Структура мережі стандарту 10*Base*-2

7.6 Стандарт 10*Base-T*

Мережі 10*Base-T* використовують як середовище дві неекрановані кручені пари (*Unshielded Twisted Pair, UTP*).

Кінцеві вузли з'єднуються за топологією «крапка-крапка» зі спеціальним пристроєм – багатопортовим повторювачем за допомогою двох кручених пар. Одна кручена пара потрібна для передачі даних від станції до повторювача (вихід *Tx* мережного адаптера), а інша – для передачі даних від повторювача до станції (вхід *Rx* мережного адаптера). Багатопортові повторювачі в цьому випадку називаються концентраторами (англомовні терміни – *hub* або *concentrator*). Концентратор здійснює функції повторювача сигналів на всіх відрізках кручених пар, підключених до його портів, так що утворюється єдине середовище передачі даних – логічний моноканал (логічна загальна «шина»).

Повторювач виявляє колізію в сегменті у випадку одночасної передачі сигналів за декількома своїми *Rx*-входами і посилає *jam*-послідовність на всі свої *Tx*-виходи. Стандарт визначає бітову швидкість передачі даних 10 Мбіт/с і максимальну відстань відрізка крученої пари між двома безпосередньо зв'язаними вузлами (станціями і концентраторами) не більше 100 м при наявності крученої пари якості не нижче категорії 3. Ця відстань визначається смугою пропущення крученою пари. На довжині 100 м вона дозволяє передавати дані зі швидкістю 10 Мбіт/с при використанні манчестерського коду.

Концентратори 10*Base-T* можна з'єднувати один з одним за допомогою тих же портів, які призначені для підключення кінцевих вузлів. При цьому потрібно подбати про те, щоб передавач і приймач одного порту були з'єднані відповідно із приймачем і передавачем іншого порту.

Для забезпечення синхронізації станцій при реалізації процедур доступу *CSMA/CD* і надійного розпізнавання станціями колізій у стандарті визначено максимальне число концентраторів між будь-якими двома станціями мережі – 4. Це правило зветься **«правило 4-х хабів»,** і воно заміняє «правило 5-4-3», яке застосовується до коаксіальних мереж.

Загальна кількість станцій у мережі 10*Base-T* не повинна перевищувати загальної межі в 1024, і для даного типу фізичного рівня цю кількість дійсно можна досягти. Для цього досить створити дворівневу ієрархію концентраторів, розташувавши на нижньому рівні достатню кількість концентраторів із загальною кількістю портів 1024. Кінцеві вузли потрібно підключити до портів концентраторів нижнього рівня. При цьому виконується правило 4-х хабів – між будь-якими кінцевими вузлами буде рівно 3 концентратори.

Мережі, побудовані на основі стандарту 10*Base-T*, мають у порівнянні з коаксіальними варіантами *Ethernet* багато переваг. Ці переваги пов'язані з розподілом загального фізичного кабелю на окремі кабельні відрізки, підключені до центрального комунікаційного пристрою. І хоча логічно ці відрізки як і раніше утворять загальне розподілене середовище, їхній фізичний поділ дозволяє контролювати їх стан і відключати у випадку обриву, короткого замикання або несправності мережного адаптера на індивідуальній основі. Ця обставина істотно полегшує експлуатацію великих мереж *Ethernet,* тому що концентратор звичайно автоматично виконує такі функції, повідомляючи при цьому адміністратора мережі про виниклу проблему.

У стандарті 10*Base-T* визначена процедура тестування фізичної працездатності двох відрізків крученої пари, що з'єднують трансивер кінцевого вузла і порт повторювача. Ця процедура називається тестом зв'язності (*link test*), і вона основана на передачі кожні 16 мс спеціальних імпульсів *J* і *K* манчестерського коду між передавачем і приймачем кожної кручений пари. Якщо тест не проходить, то порт блокується і відключає проблемний вузол від мережі. Оскільки коди *J* і *K* є забороненими при

передачі кадрів, то тестові послідовності не впливають на роботу алгоритму доступу до середовища.

Поява між кінцевими вузлами активного пристрою, що може контролювати роботу вузлів і ізолювати від мережі некоректно працюючі, є головною перевагою технології 10*Base*-T у порівнянні зі складними в експлуатації коаксіальними мережами. Завдяки концентраторам мережа Ethernet придбала деякі риси відмовостійкої системи.

Структура мережі стандарту 10*Base-T* наведена на рис 7.4.

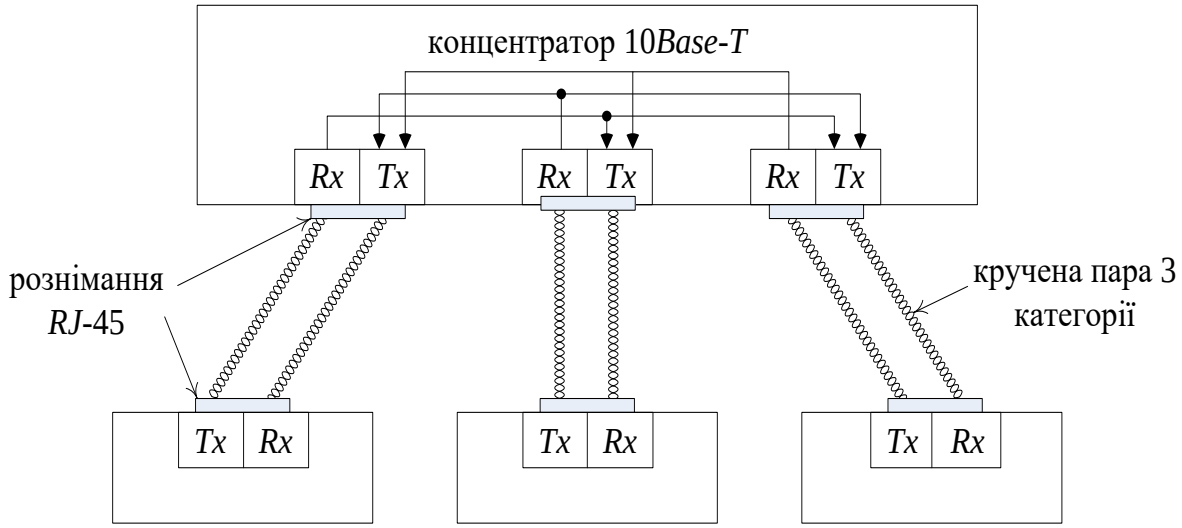

Рисунок 7.4 – Структура мережі стандарту 10*Base-T*

7.7 Оптоволоконний *Ethernet*

Функціонально мережа *Ethernet* на оптичному кабелі складається з тих же елементів, що і мережа стандарту 10*Base-T* – мережних адаптерів, багатопортового повторювача і відрізків кабелю, що з'єднують адаптер з портом повторювача. Як і у випадку крученої пари, для з'єднання адаптера з повторювачем використовуються два оптоволокна. Одне з'єднує вихід T_x адаптера із входом R_x повторювача, а інше - вхід R_x адаптера з виходом T_x повторювача.

Стандарт *FOIRL (Fiber Optic Inter-Repeater Link)* є першим стандартом комітету 802.3 для використання оптоволокна в мережах *Ethernet*. Він гарантує довжину оптоволоконного зв'язку між повторювачами до 1 км при загальній довжині мережі не більше 2500 м. Максимальне число повторювачів між будь-якими вузлами мережі – 4. Максимальний діаметр мережі *Ethernet* (2500 м) тут можна досягти, хоча максимальні відрізки кабелю між усіма 4 повторювачами, а також між повторювачами і кінцевими вузлами неприпустимі. Інакше вийде мережа довжиною 5000 м.

Стандарт 10*Base-FL* є незначним поліпшенням стандарту *FOIRL*. Збільшено потужність передавачів, тому максимальна відстань між вузлом і концентратором збільшилася до 2000 м. Максимальне число повторювачів між вузлами залишилося рівним 4, а максимальна довжина мережі - 2500 м.

Стандарт 10*Base-FB* призначений тільки для з'єднання повторювачів. Кінцеві вузли не можуть використовувати цей стандарт для приєднання до портів концентратора. Між вузлами мережі можна встановити до 5 повторювачів 10*Base-FB* при максимальній довжині одного сегмента 2000 м і максимальній довжині мережі 2740 м.

Як і в стандарті 10*Base-T*, оптоволоконні стандарти *Ethernet* дозволяють з'єднувати концентратори тільки в деревоподібні ієрархічні структури. Будь-які петлі між портами концентраторів не допускаються.

7.8 Методика розрахунку конфігурації мережі *Ethernet*

Дотримання численних обмежень, встановлених для різних стандартів фізичного рівня мереж *Ethernet*, гарантує коректну роботу мережі (звичайно, при справному стані всіх елементів фізичного рівня).

Найбільш часто доводиться перевіряти обмеження, пов'язані з довжиною окремого сегмента кабелю, а також кількістю повторювачів і загальною довжиною мережі. Правила «5-4-3» для коаксіальних мереж і «4-х хабів» для мереж на основі крученої пари і оптоволокна не тільки дають гарантії працездатності мережі, але і залишають великий «запас міцності» мережі. Наприклад, якщо порахувати час подвійного оберту в мережі, що складається з 4-х повторювачів 10*Base*-5 і 5-ти сегментів максимальної довжини 500 м, то виявиться, що воно становить 537 бітових інтервалів. Час передачі кадру мінімальної довжини, що становить разом із преамбулою 72 байт, дорівнює 575 бітовим інтервалам. При цьому 38 бітових інтервалів є запасом надійності стандарту Ethernet. Проте комітет 802.3 говорить, що і 4 додаткових бітових інтервали створюють достатній запас надійності.

Комітет *IEEE* 802.3 наводить вихідні дані про затримки, що вносять повторювачі і різні середовища передачі даних, для тих фахівців, які хочуть самостійно розрахувати максимальну кількість повторювачів і максимальну загальну довжину мережі, не задовольняючись тими значеннями, які наведені в правилах «5-4-3» і « 4-х хабів». Особливо такі розрахунки корисні для мереж, які складаються зі змішаних кабельних систем, наприклад коаксіального і оптоволоконного кабелів, для яких правила про кількість повторювачів не розраховані. При цьому максимальна довжина кожного окремого фізичного сегмента повинна строго відповідати стандарту, тобто 500 м для «товстого» коаксіала, 100 м для крученої пари і т.д.

Щоб мережа *Ethernet*, яка складається із сегментів різної фізичної природи, працювала коректно, необхідне виконання чотирьох основних умов:

1) кількість станцій у мережі повинна бути не більше 1024;

2) максимальна довжина кожного фізичного сегмента повинна бути не більше величини, зазначеної у відповідному стандарті фізичного рівня;

3) час подвійного оберту сигналу (*Path Delay Value, PDV*) між двома самими віддаленими одна від одної станціями мережі повинен бути не більше 575 бітових інтервалів;

4) скорочення міжкадрового інтервалу *IPG* (*Path Variability Value, PW*) при проходженні послідовності кадрів через усі повторювачі повинне бути не більше ніж 49 бітових інтервалів. Оскільки відправлені кадри кінцевих вузлів забезпечують початкову міжкадрову відстань у 96 бітових інтервалів, то після проходження повторювача вона повинна бути не менше ніж 96 – 49 = 47 бітових інтервалів.

Дотримання цих вимог забезпечує коректність роботи мережі навіть у випадках, коли порушуються прості правила конфігурування, які визначають максимальну кількість повторювачів і загальну довжину мережі в 2500 м.

Домен колізій

У технології *Ethernet*, незалежно від застосовуваного стандарту фізичного рівня, існує поняття домену колізій.

Домен колізій *(collision domain)* – це частина мережі *Ethernet*, на яку розповсюджується сигнал колізії.

Мережа *Ethernet*, побудована на повторювачах, завжди утворює один домен колізій. Домен колізій відповідає одному поділеному середовищу. Мости, комутатори і маршрутизатори поділяють мережу *Ethernet* на декілька доменів колізій.

Контрольні запитання

1. Який метод доступу використовується в мережах *Ethernet*?
2. Що таке колізія, домен колізій?
3. Які формати кадрів використовуються в мережах *Ethernet*?
4. Які існують специфікації фізичного середовища *Ethernet*, яка його структура?
5. Яка методика використовується для розрахунку конфігурації мережі *Ethernet*?

8 ТЕХНОЛОГІЇ TOKEN RING, TOKEN BUS, FDDI

8.1 Технологія Token Ring

Технологія *Token Ring* (маркерне кільце) була розроблена фірмою *IBM* наприкінці 1970-х років. Специфікації *IEEE* 802.5 практично повторюють фірмові специфікації, відрізняючись лише в деяких деталях (наприклад, *IEEE* 802.5 не обмовляє середовище передачі і топологію мережі, а фірмовий стандарт визначає кручену пару як середовище і зірку як фізичну топологію).

Мережі *Token Ring* можуть працювати на одній із двох бітових швидкостей: 4 Мбіт/с (*IEEE* 802.5) або 16 Мбіт/с (*IEEE* 802.5r). В одному кільці можуть бути присутні тільки станції, які працюють на одній швидкості.

Token Ring визначає логічну топологію "кільце": кожна станція пов'язана із двома сусідніми. Фізично ж станції з'єднуються в зіркоподібну мережу, у центрі якої знаходиться пристрій багатостанційного доступу (*MSAU, Multi-Station Access Unit*), який по суті є повторювачем. Як правило, *MSAU* вміє виключати непрацюючу станцію з кільця (для цього використовується шунтувальне реле). *MSAU* мають також окремі рознімання для об'єднання декількох *MSAU* в одне велике кільце.

Максимальна кількість станцій у кільці – 250 (*IEEE* 802.5), 260 (*IBM Token Ring*, кабель *STP*) і 72 (*IBM Token Ring*, кабель *UTP*).

Максимальна довжина кільця *Token Ring* становить 4000 м.

Наприкінці 1990-х років компанією *IBM* розроблений новий варіант технології *Token Ring – High Speed Token Ring* (*HSTR*), який підтримує швидкості в 100 і 155 Мбіт/с. Ведуться розробки версії *Token Ring* зі швидкістю в 1 Гбіт/с.

8.2 Маркерний метод доступу

Token Ring – це найпоширеніша технологія локальної мережі з передачею маркера. У таких мережах циркулює (передається станціями один одному в певному порядку) спеціальний блок даних – маркер (*token*). Станція, яка прийняла маркер, має право передавати свої дані. Для цього вона змінює в маркері один біт ("маркер зайнято"), додає до нього свої дані і передає в мережу (наступної станції). Станції передають такий кадр далі по кільцю, поки він не досягне одержувача, який скопіює з нього дані і передасть далі. Коли відправник одержує свій кадр із даними, що зробив повне коло, він його відкидає і або передає новий кадр даних (якщо не минув максимальний час володіння маркером), або змінює біт зайнятості маркера на "вільний" і передає маркер далі кільцем.

Протягом усього часу володіння маркером, до і після передачі свого кадру, станція повинна видавати, заповнюючи послідовність *(fill sequence*),

довільну послідовність 0 і 1. Це робиться для підтримки синхронізації і контролю за обривом кільця.

Основний режим роботи адаптера – повторення: передавач побітно видає дані, які надійшли до приймача. Коли в станції є кадр для передачі і прийнято вільний маркер, то станція переходить у режим передачі. При цьому бітовий потік, що надходить до приймача, аналізується на наявність службових кадрів і або (якщо виявлено службовий кадр) ініціюється переривання (припинення передачі свого кадру і видача кадру переривання), або прийняті дані відкидаються.

У мережах *Token Ring* 4 Мбіт/с станція звільняє маркер тільки після повернення її кадру даних. Мережі *Token Ring* 16 Мбіт/с використовують алгоритм раннього звільнення маркера (*Early Token Release*): маркер передається в кільце відразу по закінченні передачі кадру даних. При цьому кільцем одночасно передається кілька кадрів даних, але генерувати їх у кожний момент часу може тільки одна станція, яка володіє в цей момент маркером.

За правильною роботою мережі стежить активний монітор (*Active Monitor, AM*), який обирається під час ініціалізації кільця як станція з максимальною MAC-адресою. У випадку відмови активного монітора проводяться вибори нового (всі станції в мережі, крім активного монітора, вважаються резервними моніторами (*Standby monitor*)). Основна функція активного монітора – це контроль наявності єдиного маркера в кільці. Монітор випускає в кільце маркер і видаляє кадри, які пройшли більше одного оберту кільцем. Щоб повідомити інші станції про себе, активний монітор періодично передає службовий кадр *AMP*. Якщо за якийсь час (достатній для оберту маркера кільцем) маркер не повернеться до активного монітора, то маркер вважається загубленим, і активний монітор генерує новий маркер.

На режим передачі кадрів впливають зазначені в стандарті максимальні інтервали часу, за дотриманням яких стежать спеціальні таймери в мережних адаптерах (наведені значення за замовчуванням, адміністратор мережі може їх змінювати):

- час утримання маркера (*Token Holding, THT*) – 8,9 мс; після закінчення цього інтервалу станція повинна припинити передачу своїх даних (поточний кадр можна передати) і звільнити маркер; за час утримання маркера станція може передати декілька (невеликих) кадрів;
- припустимий час передачі кадру (*Valid Transmission, TVX*) – 10 мс; максимальний час, у який повинна укластися передача одного кадру; контролюється активним монітором;
- час очікування вільного маркера (*No Token, TNT*) – 2,6 с; час очікування вільного маркера активним монітором; якщо за цей час маркер не з’явиться, активний монітор виконує очищення кільця і генерує новий маркер;
- період посилання *AMP* (*Active Monitor, TAM*) – 7 с;

- час очікування *AMP* (*Standby Monitor Detect AMP*, *TSM*) – 16 с; якщо за цей інтервал не було ні одного кадру *AMP*, ініціюються вибори нового активного монітора.

8.3 Формати кадрів *Token Ring*

Token Ring визначає три типи кадрів (рис 8.1): маркер, кадр даних (службових або користувальницьких) і переривання.

Маркер

Поле	*SD*	*AC*	*ED*
Довжина (байт)	1	1	1

Кадр даних

Поле	SD	AC	FC	DA	SA	RI	Info	FCS	ED	FS
Довжина (байт)	1	1	1	6	6	≥0	≥0	4	1	1

Переривання

Поле	SD	ED
Довжина (байт)	1	1

Рисунок 8.1 – Формати кадрів *Token Ring*

Поле *SD* (*Starting Delimiter*, початковий обмежник) вказує на початок кадру і має значення *JK*0*JK*000 у манчестерському коді. Оскільки в полі присутні спеціальні коди *J* і *K*, послідовність даних неможливо переплутати з обмежником кадру.

Поле *ED* (*Ending Delimiter*, кінцевий обмежник) має значення *JK*1*JK*1*IE*, де біт *I* (*Intermediate*, проміжний) вказує, чи є кадр проміжним у послідовності кадрів (*I*=1) або останнім/єдиним (*I*=0), а біт *E* (*Error*, помилка) вказує на виявлену помилку (*E*=1).

Поле *AC* (*Access Control*, керування доступом) має формат *PPPTMRRR*, де біти *PPP* (*Priority*, пріоритет) містять пріоритет маркера, біт *T* (*Token*, маркер) відрізняє вільний маркер (*T* = 1) від кадру даних (*T*=0), біт *M* (*Monitor*, монітор) використовується для розпізнавання кадрів, які зробили більше одного оберту кільцем: монітор встановлює *M* = 1 у всіх кадрах, які проходять через нього (інші станції встановлюють *M* = 0), а кадри з *M* = 1 повинні видалятися монітором. Біти *RRR* (*Priority reservation,* пріоритет резервування) вказують пріоритет станції, яка бажає захопити маркер.

Поле *FC* (*Frame Control*, керування кадром) має формат *FFZZZZZZ*. Біти *FF* визначають тип кадру:

- 00 – кадр даних зі службовою інформацією (*MAC*-кадр);
- 01 – кадр даних користувача (*LLC*-кадр);
- 10, 11 – резерв.

Біти *ZZZZZZ* використовуються *LLC*-кадрами для зберігання інформації про пріоритет кадру рівня *LLC*. MAC-кадри в цих бітах зберігають свій тип. *IEEE* 802.5 визначає 25 типів MAC-кадрів, серед яких основні:

- CT (Claim Token, заявка на створення маркера) – відправляється резервним монітором при підозрі про відмову активного монітора;
- DAT (Duplicate Address Test, тест на дублювання адреси). Відправляється станцією при підключенні до кільця для перевірки унікальності своєї адреси;
- *AMP* (*Active Monitor Present*, присутній монітор є активним). Регулярно (раз в 7 с) відправляється активним монітором для підтвердження своєї присутності;
- *SMP* (*Standby Monitor Present*, присутній монітор є резервним) – відповідь на кадр AMP;
- *BCN* (*Beacon*, бакен) – відправляється станцією, яка виявила мережну проблему (тишу або нескінченний потік, що може вказувати на обрив кабелю, наявність несправного адаптера в однієї зі станцій і т.п.);
- *PRG* (*Purge*, очищення) – сигнал від активного монітора про очищення кільця від усіх кадрів.

Поле *DA* (*Destination Address*, адреса призначення) має структуру, подібну до структури адреси в стандарті *IEEE* 802.3. Старший біт адреси визначає одержувача: 0 – індивідуальний (одна станція), 1 – груповий. Другий біт адреси визначає спосіб призначення адреси: 0 – глобально (універсально, зашито в ПЗП адаптера), 1 – локально. Інші біти використовуються для вказівки адреси станції, кільця або групи одержувачів. Кілька адрес зарезервовано для службових цілей:

FF FF FF FF FF FF – широкомовний кадр (всім станціям);
C0 00 FF FF FF FF – широкомовний *MAC*-кадр;
C0 00 00 00 00 01 – активний монітор ;
C0 00 00 00 00 02 – сервер параметрів кільця;
C0 00 00 00 00 08 – монітор помилок кільця;
C0 00 00 00 00 10 – сервер звітів про конфігурацію;
C0 00 00 00 01 00 – міст;
C0 00 00 00 20 00 – керування мережею.

Поле *SA* (*Source Address*, адреса джерела) має той же формат, що і адреса призначення, за винятком старшого біта. В адресі джерела старший біт називається *RII* (*Routing Information Indicator*) і вказує (якщо *RII*=1) на наявність даних у полі *RI*.

Поле *RI* (*Routing Information*, маршрутна інформація), якщо використовується (*RII*=1), містить послідовність (двобайтних) адрес сегментів на шляху до одержувача. Дані цього поля гкеруютьуправляють роботою мостів у режимі маршрутизації від джерела.

Поле *Info* містить або дані користувача (кадр *LLC*), або службові дані, обумовлені типом кадру (кадр *MAC*). Стандарт не обмежує розмір цього поля, хоча практично його максимальний розмір визначається співвідношенням часів передачі кадру і утримання маркера. Для 4 Мбіт/с максимальний розмір кадру найчастіше встановлюється в 4 Кбайт, а для 16 Мбіт/с – в 16 Кбайт. Мінімальний розмір поля даних не визначений.

Поле *FCS* (*Frame Check Sequence*, контрольна сума) зберігає 4-байтный CRC-*Код* для всіх полів з *FC* по *Info* включно.

Поле *FS* (*Frame Status*, статус кадру) має формат *ACrrACrr*. Біти *rr* зарезервовані і не використовуються, інші біти дублюються для надійності. Біт *A* (*Address Recognized*, адреса розпізнана) вказує на те, що одержувач кадру присутній у кільці, а біт *C* (*Frame Copied*, кадр скопійований) вказує на те, що приймач скопіював кадр собі в буфер. За цими полями станція-відправник може довідатися, що переданий нею кадр був отриманий.

8.4 Система пріоритетного доступу

Мережі *Token Ring* гарантують, що кожна станція буде одержувати право на передачу даних не рідше, ніж раз у встановлений інтервал часу. Крім того, використовується система пріоритетів, яка дозволяє деяким станціям користуватися мережею частіше інших. Для цього в кадрі *Token Ring* виділено два поля: поле пріоритету і поле резервування. Усього рівнів пріоритету вісім: від нижчого (0) до вищого (7). Маркер теж завжди має деякий рівень пріоритету. Станція може захопити маркер тільки в тому випадку, якщо пріоритет кадру, що вона збирається передати, не нижче пріоритету маркера (бітів *PPP* поля *AC*).

Станція, яка захопила маркер, зберігає старе значення його пріоритету, записує в нього пріоритет свого кадру і обнуляє поле резервування. Якщо в кільці є станція, яка бажає передати кадр із більш високим пріоритетом, то вона записує пріоритет свого кадру в поле резервування кадру, який минає кільцем. У результаті чого після оберту кільцем в полі резервування буде записано максимальний пріоритет з кадрів, які очікують передачі. Тоді станція переписує пріоритет з поля резервування в поле пріоритету маркера і видає вільний маркер у кільце (захопити такий маркер зможе тільки станція з кадром зазначеного пріоритету).

Станція, яка підвищує пріоритет маркера, стає запам’ятовувальною станцією (*stacking station*) і організує стек для зберігання ще необслужених низьких пріоритетів. Коли через таку станцію проходить вільний маркер із пріоритетом, рівним пріоритету на верхівці стека, то вона витягає наступне значення зі стека і знижує пріоритет маркера до нього.

Механізм пріоритетів у мережах *Token Ring* не є обов’язковим до використання. Як правило, більшість додатків ним не користується, і кільце працює в непріоритетному режимі (пріоритет маркера завжди дорівнює 0). Існує тенденція до переносу механізмів пріоритетного

обслуговування на рівні, вищому за канальний (пріоритетне обслуговування можуть забезпечувати, наприклад, маршрутизатори).

При побудові більших мереж *Token Ring* доводиться використовувати велику кількість кілець. Окремі кільця зв'язуються одне з одним, як і в інших мережах, за допомогою мостів. Мости бувають "прозорими" (*IEEE* 802.1*d*) і з маршрутизацією від джерела. Останні дозволяють зв'язати в єдину мережу кілька кілець, які використовують загальну мережну *IPX*- або *IP*-адресу.

Використання мостів дозволяє перебороти і обмеження щодо числа станцій у мережі (260 для специфікації *IBM* і 250 для *IEEE*). Мости можуть зв'язувати між собою фрагменти мереж, які використовують різні протоколи, наприклад, 802.5, 802.4 і 802.3. Пакети з кільця 1, адресовані об'єкту цього ж кільця, ніколи не потраплять у кільце 2 і навпаки. Через міст пройдуть лише пакети, які адресовані об'єктам сусіднього кільця. Фільтрація пакетів здійснюється за фізичною адресою і номером порту. На основі цих даних формується власна база даних, яка містить інформацію про об'єкти кілець, підключених до мосту. Схема розподілу мережі за допомогою мостів може сприяти зниженню ефективного завантаження мережі.

Мости з маршрутизацією від джерела можуть поєднувати тільки мережі *Token Ring*, а маршрутизація пакетів покладається на всі пристрої, які посилають інформацію в мережу (звідси і назва цього виду мостів). Це означає, що в кожному з мережних пристроїв повинно бути завантажене програмне забезпечення, яке дозволяє маршрутизувати пакети від відправника до одержувача. Ці мости не створюють власних баз даних про розташування мережних об'єктів і посилають пакет у сусіднє кільце на основі маршрутної вказівки, яка надійшла від відправника самого пакета. Таким чином, база даних про розташування мережних об'єктів виявляється розподіленою між станціями, що зберігають власні маршрутні таблиці. Програми маршрутизації використовують мережний драйвер адаптера.

Мости з маршрутизацією від джерела переглядають усі кадри, які надходять, і відбирають ті, що мають індикатор інформації про маршрут *RII*=1. Такі кадри копіюються, і за інформацією про маршрут визначається, чи треба їх посилати далі. Мости з маршрутизацією від джерела можуть бути настроєні на широкомовну передачу за всіма маршрутами, або на широкомовну передачу за одним маршрутом.

У мережах зі складною топологією маршрути формуються відповідно до ієрархічного протоколу *STP* (*Spanning Tree Protocol*). Цей протокол організує маршрути динамічно з вибором оптимального маршруту, якщо адресат можливо досягти декількома шляхами. При цьому мінімізується транзитний трафік.

8.5 Технологія *Token Bus IEEE* 802.4 (Маркерна «шина»)

Стандарт *IEEE* 802.4 описує властивості мереж, які відомі як маркерна «шина». З погляду правил надання доступу цей стандарт схожий зі стандартом *Token Ring*. Як фізичне середовище використовується 75-омний кабель. При необхідності побудови мережі типу дерева, а також для збільшення довжини мережі використовуються повторювачі. Мережа може забезпечити пропускну здатність до 10 Мбіт/с при смузі пропуску кабелю 12 МГц.

Для доступу до мережного середовища станція повинна одержати пакет-маркер. Отримавши маркер, мережний пристрій може почати передачу даних, а завершивши цю процедуру, пристрій повиннен переслати маркер наступної мережної станції. Передача маркера відбувається доти, поки він не досягне молодшої станції, після чого він вертається до першої станції.

Станції одержують доступ до шини в результаті змагальної процедури, яка називається “вікно відгуків”. Вікно відгуків являє собою часовий інтервал, який дорівнює за тривалістю одному системному такту, що у свою чергу дорівнює часу поширення сигналу шиною. Цей час відлічується від моменту закінчення передачі кадру керування. Протягом цього часу станція-ініціатор очікує відгуку від інших станцій. Будь-яка станція мережі, будучи власником маркера, може запустити цей процес за допомогою посилання кадру “пошук наступної станції".

Запити на підключення здійснюються шляхом відправлення пакета установки наступної станції**,** у полі даних якої записується адреса станції доступу шиною. Адреса наступної сусідньої станції менше адреси станції-відправника (маркер рухається в напрямку убування адрес). Найчастіше посилається кадр із одним вікном відгуків. При цьому запити можуть посилати станції з адресами, не меншими, ніж адреса найближчого сусіда. Якщо процес ініційовано станцією з найменшим номером, то посилається пакет із двома вікнами відгуків, одне для станції з меншим ніж у попередника номером, а інша – з адресою більш, чим у попередника. Після цього станція чекає відповіді протягом одного такту. Якщо відповіді немає, то маркер передається наступній станції. Якщо ж отримано одну відповідь, ініціюється підключення станції за допомогою пакета "установка наступної станції". При одержанні більше одного відгуку виникає конфлікт, для вирішення якого посилається пакет "дозвіл конфлікту із чотирма вікнами". Станції заносять свої запити у вікна відповідно до перших двох бітів своєї адреси. Якщо спроба розв’язати конфлікт при цьому не вдалася, пакет відсилається повторно. У новій спробі беруть участь тільки станції, що брали участь у першому раунді, а для порівняння використовуються вже наступні два біти адреси. Процедура може завершитися підключенням однієї зі станцій або вичерпанням числа спроб.

Станція може відключитися від мережі в будь-який час, але це викличе ініціалізацію системи і тимчасове порушення роботи мережі. Тому для відключення від мережі станція повинна дочекатися одержання

маркера, після чого вона посилає пакет типу "установка наступної станції", у полі даних якого знаходиться адреса її спадкоємця. Якщо тримач маркера одержить пакет, що показує наявність у мережі ще одного власника маркера, то він видаляє свій маркер і переходить у режим очікування. Одержавши маркер, станція повинна почати передачу даних або передати його наступній станції. Після передачі маркера станція протягом одного циклу прослуховує мережу, щоб переконатися в активності свого спадкоємця. Якщо спадкоємець не посилає нічого впродовж секунди, то станція повторює передачу маркера. Якщо і це не допомагає, то посилає пакет "хто наступний?" з адресою спадкоємця в полі даних і трьома вікнами відгуків. Якщо станція виявляє в полі даних адресу свого попередника, то вона посилає кадр типу "установка наступної станції" за адресою відправника. За відсутності кадру "установка наступної станції" станція посилає такий пакет сама собі із двома вікнами для виявлення активних мережних пристроїв.

При виявленні втрати маркера запускається процедура ініціалізації мережі, при цьому посилається пакет "вимога маркера". Станція, яка послала запит, прослуховує шину і при виявленні мережної активності вибуває зі змагання (є станція з більшою, ніж у неї адресою). У мережі визначено 4 класи обслуговування (6, 4, 2, 0). Станція може передавати дані класу 6 протягом припустимого часу втримання маркера *THT* (для класу 6). При *N* станцій у мережі максимальний час очікування буде дорівнює *THT***N*. По завершенні передачі даних класу 6 (або якщо вони не передавалися зовсім) можливо передавати дані класу 4. Аналогічно визначено час обігу маркера для класів 4, 2 і 0.

8.6 Технологія *FDDI*

Специфікація *FDDI* (*Fiber Distributed Data Interface*, оптоволоконний інтерфейс розподілу даних) розроблена і стандартизована інститутом *ANSI* (в 1986-1988 рр. – група *X*3*T*9.5, після 1995 року – група *X*3*T*12).

FDDI – це історично перша технологія локальних мереж, яка використовує як середовище передачі оптоволоконний кабель. Початкові версії *FDDI* забезпечують швидкість передачі 100 Мбіт/с подвійним оптоволоконним кільцем довжиною до 100 км. У нормальному режимі дані передаються тільки одним кільцем з пари – первинним (*primary*). Вторинне (*secondary*) кільце використовується у випадку відмови частини первинного кільця. Первинним і вторинним кільцем дані передаються в протилежних напрямках. Це дозволяє підтримувати порядок вузлів мережі при підключенні вторинного кільця до первинного. У випадку декількох відмов мережа *FDDI* розпадається на декілька окремих (але функціонуючих) мереж.

Мережі *FDDI* не мають собі рівних при побудові опорних магістралей (*backbone*) локальних мереж, дозволяючи реалізувати принципово нові можливості – вилучену обробку зображень і інтерактивну

графіку. Звичайно пристрої (*DAS - Dual Attached Station*) підключаються до обох кілець одночасно. Пакети цими кільцями рухаються в протилежних напрямках. У нормі тільки одне кільце активне (первинне), але при виникненні перебою (відмова в одному з вузлів) активізується і друге кільце. Це помітно підвищує надійність системи, дозволяючи обійти несправну ділянку (схема з'єднань всередині станцій-концентраторів на рис. 8.2 є сильно спрощеною). Передбачено можливість підключення станцій тільки до одного кільця (*SAS - Single Attached Station*), що помітно дешевше. До одного кільця можна підключити до 500 *DAS* і 1000 *SAS*. Сервер і клієнт мають різні типи інтерфейсів.

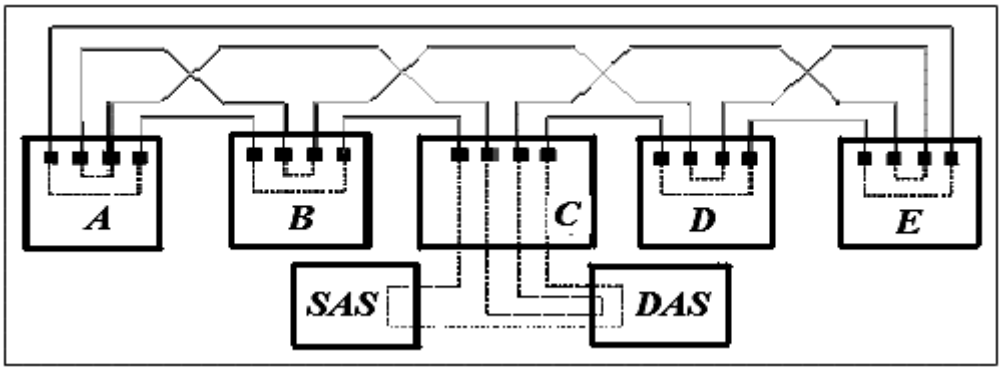

Рисунок 8.2 – Схема зв'язків у подвійному кільці *FDDI*

Технологія забезпечує передачу синхронного і асинхронного трафіку. Синхронний трафік передається завжди, незалежно від завантаженості кільця. Асинхронний трафік може довільно затримуватися. Кожній станції виділяється частина смуги пропущення, у межах якої станція може передавати синхронний трафік. Частина смуги пропущення кільця, яка залишилась, відводиться під асинхронний трафік. Мережі *FDDI* не визначають пріоритетів для кадрів. Будь-який пріоритетний трафік повинен передаватися як синхронний, а інші дані – асинхронно.

FDDI використовує маркерний метод доступу, близький до методу доступу мереж *Token Ring*. Основна відмінність полягає у плаваючому значенні часу утримання маркера для асинхронного трафіку. При невеликому завантаженні мережі час утримання зростає, а при перевантаженнях – зменшується. Під час ініціалізації кільця вузли домовляються про максимально припустимий час оберту маркера по кільцю *T_Opr*. Для синхронного трафіку час утримання маркера не змінюється. Для передачі синхронного кадру вузол завжди має право захопити минаючий маркер і втримувати його впродовж заздалегідь заданого фіксованого часу. Якщо вузол хоче передати асинхронний кадр, він повинен виміряти час оберту маркера (*Token Rotation Time, TRT*) – інтервал між двома проходженнями маркера через нього. Якщо кільце не перевантажене (*TRT<T_Opr*), то вузол може захопити маркер і передати свій кадр (або кадри) у кільце, при цьому припустимий час утримання

маркера *THT=T_Opr-TRT*. Якщо кільце перевантажене (*TRT>T_Opr*), то вузол не має права захоплювати маркер.

Як і мережі Token Ring 16 Мбіт/с, FDDI використовує алгоритм раннього звільнення маркера, у результаті чого в кільці одночасно може просуватися кілька кадрів (маркер завжди один). Формат кадру *FDDI* дуже близький до формату кадру *Token Ring*, за винятком полів пріоритету.

Стандарт *FDDI* визначає чотири компоненти:

1. *MAC* (*Media Access Control*), що визначає формати кадрів, маніпуляції з маркером, адресацію, обробку помилок при логічних відмовах (відповідає канальному рівню моделі *OSI*).

2. *PHY* (*Physical*) виконує фізичне і логічне кодування і декодування, синхронізацію і кадрування.

3. *PMD* (*Physical Medium Dependent*) визначає властивості оптичних або електричних компонентів, параметри ліній зв'язку (*PMD* і *PHY* відповідають фізичному рівню *OSI*).

4. *SMT* (*Station Management*) виконує всі функції з керування і контролю роботи інших компонентів, визначає конфігурацію вузлів і кілець, процедури підключення/відключення, ізоляцію елементів, що відмовили, забезпечує цілісність кільця (підключаючи вторинне кільце при відмові первинного).

Як середовище передачі *FDDI* може використовувати багатомодове оптоволокно (*MMF-PMD*, довжина кабельного сегмента до 2 км), одномодове оптоволокно (*SMF-PMD*), кручену пару категорії 5 або екрановану кручену пару *STP* Type 1 (*TP-PMD*). Всі оптоволоконні варіанти *FDD*I використовують довжину хвилі 1300 нм. Різновид *FDDI* крученої парі іноді називають *CDDI* (*Copper Distributed Data Interface*) або *TPDDI* (*Twisted Pair Distributed Data Interface*).

FDDI використовує окремі лінії для передачі і прийому сигналів. Логічним є кодування – 4B/5B. Фізичне кодування при використанні оптоволокна – це *NRZI*, при використанні крученої пари – це *MLT*-3.

FDDI-кадри використовують заголовки, обумовлені стандартом *IEEE* 802.2, які не мають поля типу, що є присутнім в *Ethernet*-заголовку. *FDDI* і *Ethernet* мають різний порядок передачі бітів, тому мости і маршрутизатори між *FDDI* і *Ethernet* повинні вміти виконувати відповідні перетворення. Через особливості маршрутизаторів не всі протоколи можуть бути реалізовані на стеку *FDDI* і *Ethernet*.

Для розв'язування проблеми створені гібридні прилади (*brouter*), які для одних протоколів працюють як маршрутизатори, будучи мостами для інших. Ці прилади для одних пакетів прозорі, інші ж пересилаються з використанням інкапсуляції. Зважаючи на те, що *FDDI* може пересилати до 400000 пакетів у секунду, схеми розпізнавання адрес мосту повинні мати відповідну швидкодію.

Нетрадиційним для інших мереж є концентратор, який використовується в *FDDI*. Він дозволяє підключити кілька приладів SAS-типу до стандартного FDDI-кільця, створюючи структури типу дерева. Але

такі структури несуть у собі певні обмеження довжини мережних елементів. Так при використанні повторювача віддалення не повинно перевищувати 1,5 км, а у випадку мосту – 2,5 км (одномодовий варіант).

Незважаючи на ці обмеження і те, що базовою топологією мереж *FDDI* є кільце, зіркоподібні варіанти також мають право на життя, допустимі і комбінації цих топологій. У межах одного будинку підключення доцільно робити через концентратор, окремі ж будинки поєднуються за схемою кільця. До кільця *FDDI* можуть також легко підключатися і субмережі *Token Ring* (через міст або маршрутизатор).

Концентратори бувають двох типів: *DAS* і *SAS*. Такі прилади підвищують надійність мережі, тому що не змушують мережу при відключенні окремого приладу переходити в аварійний режим обходу. Застосування концентраторів знижує і вартість підключення до *FDDI*. Концентратори можуть допомогти при створенні невеликих групових підмереж, призначених для розв'язання специфічних задач.

Новим пристроєм, використовуваним в *FDDI*-вузлах, є міжвузлові процесори (*Internetwork Nodal Processor - INP*), які є розвитком ідей *Front End Processor* (*FEP*). *INP*, завдяки модульності, можуть допомогти користувачеві адаптуватися до змін, що постійно відбуваються у мережах, де він працює. *INP* може виконувати функції багатопротокольного мосту або маршрутизатора. Керування *FDDI*-устаткуванням здійснюється за допомогою протоколу *SNMP* і бази даних *MIB*. Передбачено деякі додаткові діагностичні засоби, які виявляють не тільки апаратні перебої, але і деякі програмні помилки.

Застосування мостів для об'єднання *FDDI*-мереж дозволяє забезпечити високий ступінь мережної безпеки і вирішити багато топологічних проблем, зняти обмеження із граничного числа *DAS*-підключень (<500). Вибір між мостом і маршрутизатором визначається тим, що важливіше: вартість, гнучкість системи або ж висока пропускна здатність.

При обривах оптоволокна можливе часткове (при двох обривах) або повне (при одному обриві) відновлення зв'язності мережі (рис. 8.3).

Технологія *FDDI* досить легко інтегрується з *Ethernet* і *Token Ring*, у результаті чого часто використовується як високошвидкісна магістраль, що поєднує ці мережі, а також для високошвидкісного підключення серверів.

Останнім часом, цю нішу в *FDDI* відвойовують більше дешеві високошвидкісні модифікації *Ethernet – Fast Ethernet* і *Gigabit Ethernet*, але вони не можуть гарантувати у порівнянні з *FDDI* відмовостійкість і відстані між вузлами.

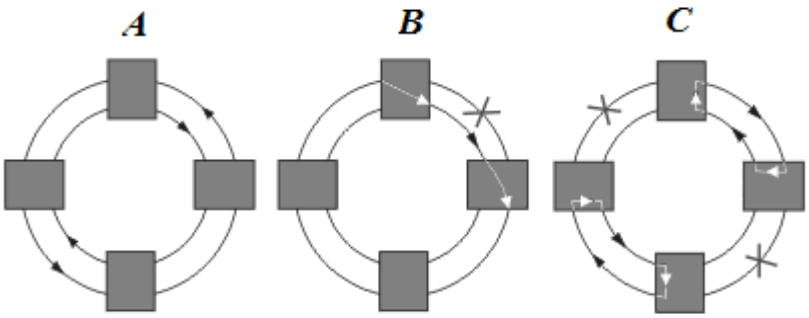

Рис. 8.3 – Варіанти зв’язків у випадку обривів волокон

Контрольні запитання

1. Який метод доступу використовується в технології *Token Ring*?
2. Які формати кадрів є в технології *Token Ring*?
3. Який метод доступу використовується в технології *Token Bus*?
4. Який метод доступу використовується в технології *FDDI*?
5. За рахунок чого в мережах *FDDI* досягається висока відмовоздатність?

9 АКТИВНЕ УСТАТКУВАННЯ ЛКМ. ТЕХНОЛОГІЇ FAST ETHERNET, GIGABIT ETHERNET, 10GIGABIT ETHERNET

9.1 Мережні адаптери

Мережний адаптер *(Network Interface Card, NIC)* (із драйвером) реалізують фізичний рівень і підрівень *MAC* канального рівня.

Основна функція мережного адаптера – передача і прийом кадрів між комп'ютером і середовищем передачі (кабелем).

Передача кадру в кабель складається з таких етапів:

1) прийом через інтерфейс *LLC*-кадру і адресної інформації (через загальні області пам'яті);
2) оформлення *MAC*-кадру (відкидання прапорців, заповнення адрес, обчислення *CRC*);
3) логічне кодування (при використанні надлишкових кодів – 4B/5B, 8B/6T та ін.) – якщо є необхідність;
4) подача сигналу в кабель із використанням фізичного коду (манчестерський, *NRZI*,*MLT*-3 та ін.).

Прийом кадру з кабелю:

1) прийом сигналів;
2) виділення сигналу із шуму (виділення бітової послідовності);
3) логічне декодування (якщо воно використовується);
4) перевірка CRC; якщо помилка, то кадр відкидається, інакше витягається *LLC*-кадр і передається *LLC*-підрівню (через загальну область пам'яті).

Розподіл обов'язків між мережним адаптером і його драйвером стандартами не визначається, тому кожний виробник вирішує це питання самостійно. Найчастіше мережні адаптери діляться на адаптери для клієнтських комп'ютерів і адаптери для серверів.

Робочі станції найчастіше оснащуються більше простими адаптерами, основну роботу при цьому виконує драйвер. Недоліком такого підходу є високий ступінь завантаження центрального процесора комп'ютера рутинними роботами по передачі кадрів з оперативної пам'яті комп'ютера в мережу.

Для серверів розробляються більш інтелектуальні адаптери з вбудованими процесорами і т.п.

9.2 Концентратори (повторювачі, хаби)

Практично у всіх сучасних технологіях локальних мереж визначено пристрій, що має кілька рівноправних назв – концентратор (concentrator), хаб (*hub*), повторювач (*repeator*). Залежно від області застосування цього пристрою значною мірою змінюється склад його функцій і конструктивне виконання.

Незмінною залишається тільки основна функція – це повторення

кадру.

У найпростішому випадку повторювач має два порти. Завданням повторювача є: передача сигналу з одного порту в інші з відновленням форми і обробкою колізій, а також ізоляція порту, на якому він виявляє безперервні помилки. Кожний порт має власний трансивер і детектор колізій. Повторювач прослуховує сигнали на всіх портах. При виявленні несучої на одному з портів він синхронізується за преамбулою з прийнятою послідовністю сигналів і транслює в усі інші порти з номінальною амплітудою імпульсів. При зникненні несучої всі порти знову переходять у стан очікування сигналу на якому-небудь із портів.

Якщо під час трансляції сигналу в якому-небудь із портів виявляється колізія, повторювач в усі порти посилає ***jam*-послідовність**. Це робиться для того, щоб вузли, підключені до всіх портів, могли б розпізнати колізію (транслювати амплітудні перекручування повторювач не може).

Якщо трансивер одного з портів виявляє колізії підряд 32 рази, то порт ***ізолюється (partitioned)*** – сигнали з цього порту перестають транслюватися в інші. Сегментовані пакети порту транслюються. Якщо трансиверу вдається передати пакет у порт, який сегментовано, без колізії, сегментація знімається і порт переходить у нормальний режим роботи.

Ця ***автоізоляція (auto partition)*** призначена для підвищення живучості мережі. Для повторювачів *Fast Ethernet* правила ізоляції і «реабілітації» портів трохи складніші. Приводом для ізоляції є і довга «балакуча» ***(jabber)*** посилка (40–75 Кбіт).

Повторювач працює на рівні фізичних сигналів – закодованих бітових ланцюжків. Ніякого аналізу кадрів він не виконує. Всі вузли, підключені до портів одного повторювача, перебувають в одному домені колізій. Для збільшення числа вузлів, які підключаються, і відстані між ними в мережі можуть бути присутніми безліч з'єднаних між собою повторювачів.

Мережа на повторювачах повинна задовольняти наступні обмеження:

- неприпустимі з'єднання повторювачів у вигляді петлі – мережа не повинна мати замкнутих контурів;
- між будь-якою парою станцій мережі на 10 Мбіт/с може бути не більше повторювачів;
- затримка поширення сигналів між будь-якою парою вузлів не повинна перевищувати 25 мкс для 10 Мбіт/с і 2,5 мкс для 100 Мбіт/с;
- повторювач *Fast Ethernet* 100 Мбіт/с класу I у сегменті може бути тільки один;
- повторювачів класу II не може бути більше двох.

9.3 Мости

Міст **(*bridge*)** є засобом передачі кадрів між двома (або більше) сегментами – доменами колізій. Міст аналізує заголовок кадру. Його цікавлять *MAC*-адреси джерела і одержувача. Міст прослуховує кадри, які надходять, і складає таблиці *MAC*-адрес вузлів, підключених до цих портів (за адресними джерелами). Якщо кадр, який надійшов, має адресу призначення, що належить тому ж сегменту, то цей кадр мостом фільтрується і нікуди не транслюється. Якщо адреса призначення відома мосту і належить до іншого сегмента, міст транслює цей кадр у відповідний порт. Якщо положення адресата призначення ще не відомо мосту, кадр транслюється на усі порти (крім того порту, звідки він надійшов).

Широкомовні і багатоадресні кадри також транслюються в усі порти. Трансляція припускає доступ до сегмента за звичайною схемою: очікування відсутності несучої, передача кадру і у випадку колізій повторні спроби передачі. Для виконання цих процедур міст повинен мати буферну пам'ять для проміжного зберігання кадрів, а також пам'ять для зберігання таблиць *MAC*-адрес вузлів сегментів усіх портів.

Описаний алгоритм поведінки належить до «прозорого» мосту ***(transparent bridge)***, який визначається стандартом *IEEE* 802.*Id*.

Кадри із широкомовними *MAC*-адресами передаються мостом на всі його порти, як і кадри з невідомою адресою призначення. Такий режим поширення кадрів називається **затопленням мережі *(flood)***. Наявність мостів у мережі не перешкоджає поширенню широкомовних кадрів за всіма сегментами мережі, зберігаючи її прозорість. Однак це є перевагою тільки в тому випадку, коли широкомовна адреса вироблена коректно працюючим вузлом. Однак часто трапляється так, що в результаті яких-небудь програмних або апаратних перебоїв протокол верхнього рівня або сам мережний адаптер починають працювати некоректно і постійно з високою інтенсивністю генерувати кадри із широкомовною адресою протягом тривалого проміжку часу. Міст у цьому випадку передає ці кадри в усі сегменти, заповнюючи мережу помилковим трафіком. Така ситуація називається **широкомовним штормом *(broadcast storm).***

На жаль, мости не захищають мережі від широкомовного шторму, у всякому разі, за замовчуванням, як це роблять маршрутизатори. Максимум, що може зробити адміністратор за допомогою мосту для боротьби із широкомовним штормом – це встановити для кожного вузла гранично допустиму інтенсивність генерації кадрів із широкомовною адресою. Але при цьому потрібно точно знати, яка інтенсивність є нормальною, а яка є помилковою. При зміні протоколів ситуація в мережі може змінитися, і те, що вчора вважалося помилковим, сьогодні може виявитися нормою. Таким чином, мости мають у своєму розпорядженні досить грубі засоби боротьби із широкомовним штормом.

9.4 Обмеження топології мережі, побудованої на мостах

Слабкий захист від широкомовного шторму – це одне з головних обмежень мосту, але не єдине. Іншим серйозним обмеженням його функціональних можливостей є нездатність підтримки петлеподібних конфігурацій мережі. Якщо в мережі, побудованої з використанням мостів, з'являться замкнуті маршрути, то це призводить до таких наслідків:

- «розмноженні» кадру, тобто появи декількох його копій;
- нескінченної циркуляції копій кадру петлею в протилежних напрямках, тобто, засмічування мережі непотрібним трафіком;
- постійної перебудови мостами своїх адресних таблиць, тому що кадр із адресою джерела буде з'являтися то на одному порту, то на іншому;
- більшої затримки передачі кадрів за рахунок їх буферизації і послідовного обслуговування портів.

Щоб виключити всі ці небажані ефекти, мости потрібно застосовувати так, щоб між логічними сегментами не було петель, тобто будувати за допомогою мостів тільки деревоподібні структури, які гарантують наявність тільки одного шляху між будь-якими двома сегментами.

Міст доцільно встановлювати в місці мережі, яка забезпечує не більше 20 % передач через міст.

Збільшувати кількість портів у сегменті, не порушуючи обмежень на число повторювачів, дозволяють ***стекові повторювачі***. Так, через звичайні порти можуть з'єднуватися в ланцюжок до 4-х стеків повторювачів *Ethernet* і до 2-х стеків повторювачів *Fast Ethernet* класу 2. Стек з повторювачів *Fast Ethernet* класу 2 може виглядати і як пристрій класу, тоді він не може з'єднуватися з іншими повторювачами.

Комутатор *(switch)* у принципі виконує ті ж функції, що і міст, але для обслуговування потоку даних, що надходять на кожний порт. У пристрій встановлюється окремий спеціалізований процесор, який реалізує алгоритм мосту. Комутатор використовується як засіб сегментації – зменшення кількості вузлів у доменах колізій.

У граничному випадку (***мікросегментації)*** до кожного порту комутатора підключається тільки один вузол. При цьому комутатор повинен направити в потрібний порт кожний кадр, що ставить високі вимоги до продуктивності процесора комутатора. Якщо до порту комутатора підключається один вузол (станція або інший комутатор), то з'являється можливість роботи в повнодуплексному режимі. При цьому колізії як такі є відсутними.

Існують два основних підходи до комутації: із проміжним збереженням кадрів і комутація «на ходу».

Технологія із проміжним зберіганням *(store and forward)* припускає, що кожний кадр, який надійшов до порту, цілком приймається в буферну пам'ять. Далі процесор аналізує його заголовок, адресу джерела

використовує для побудови своїх таблиць, а за адресою призначення визначає порт, куди кадр повинен бути переданий. У випадку багатоадресної або широкомовної передачі це буде група із портів. Передача в порт виконується в міру його звільнення, відповідно до процедури *CSMA/CD*. Після успішної передачі (в усі необхідні порти) кадр із пам'яті видаляється, звільняючи місце. Ця технологія дозволяє аналізувати кадр (перевіряти *CRC*-код) і ігнорувати помилкові кадри. Недоліком такого підходу є значна затримка передачі кадрів – принаймні, на час прийому кадру (для максимально довгого кадру при 10 Мбіт/с – 1,22 мс).

Комутація на ходу *(on-the-fly*) виконується, по можливості, без проміжного зберігання кадру. Порт приймає кадр, одночасно аналізуючи поле його заголовка. Як тільки будуть прийняті біти адреси призначення (перші 6 байт після преамбули), комутатор уже може пересилати кадр у порт або порти призначення, якщо вони не зайняті. У випадку, якщо порт призначення зайнятий, проміжне зберігання неминуче. Комутація на ходу вносить мінімальну затримку. При вільному порту призначення вона становить (8+6)8 = 112 *bt* (бітових інтервалів), для швидкості 10 Мбіт/с – 11,2 мкс. Однак перевірка CRC не відбувається, і комутатор поширює всі кадри, у тому числі і короткі, які відсічені колізіями (що є недоліком комутації на ходу).

На відміну від мостів, число портів у яких було невелике (часто всього два), комутатори мають багато портів, між якими для кожного пакета повинний встановлюватися віртуальний ланцюг передачі. У загальному випадку *N*-портовий комутатор з напівдуплексними портами повинен забезпечувати до *N/2* одночасно діючих віртуальних ланцюгів. У випадку повного дуплекса кількість ланцюгів теоретично може досягати і *N*.

Залежно від продуктивності комутатор може бути блокуючим і не блокуючим**.**

Не блокуючий комутатор здатний обробляти всі кадри, які приходять на всі його порти з максимальною швидкістю, що забезпечує середовище передачі. Очевидно, що для цього продуктивність «комутаційної матриці» повинна бути не меншою, чим сума пропускної здатності половини портів. У випадку повного дуплекса в цьому співвідношенні пропускну здатність порту варто вважати рівною подвоєній бітовій швидкості (тобто 20, 200 або 2000 Мбіт/с). Для швидкості 10 Мбіт/с і при не дуже великій кількості портів це досягається відносно просто. Високі швидкості створюють певні труднощі, особливо при великій кількості портів.

Комутаційна матриця — це апаратна схема (електронний комутатор), який дозволяє організувати ланцюг передачі логічного сигналу між будь-якою парою портів. Процесор кожного порту приймає кадр спочатку у свій буфер. Як тільки процесор порту визначає адресу призначення чергового кадру, він запитує в матриці необхідне з'єднання.

Якщо вихідний порт вільний, то встановлюється логічний зв'язок і кадр через матрицю надходить на вхід передавача вихідного порту. Якщо вихідний порт зайнятий, то кадр зберігається в буферній пам'яті вхідного порту на час, необхідний для звільнення потрібного вихідного порту призначення. Неважко помітити, що обсяг (кількість елементів) схеми комутаційної матриці зростає пропорційно квадрату числа портів, тому матриця застосовується при обмеженому (фіксованому) числу портів.

Загальна шина високої продуктивності зв'язує процесори всіх портів. Кадри нею пересилаються дрібними фрагментами (осередками) на швидкості, істотно більшій за бітову швидкість портів. У результаті кожна передача займає невелику частину часу шини і декілька пар процесорів можуть обмінюватися кадрами псевдопаралельно. Продуктивність шини в ідеалі повинна бути не менше суми пропускної здатності половини портів. Доти, поки ця умова виконується, збільшення кількості портів не викликає особливих технічних проблем. Об'єднуюча шина широко використовується в **модульних комутаторах** на основі шасі. Тут шина реалізується у вигляді пасивної об'єднуючої панелі **(*passive back plane*),** а модулі із групами портів можуть встановлюватися у відносно довільній кількості з можливістю «гарячої» заміни **(*hot swap*).**

Пам'ять, що розділяється, – це єдина буферна пам'ять, доступна процесорам усіх портів комутатора. Всі вхідні кадри розміщаються в цій пам'яті, а процесорам вихідних портів передаються лише покажчики на блоки пам'яті, які містять призначені їм кадри. Процесори вихідних портів після успішної передачі даних відзначають ці блоки як вільні для подальшого використання. Загальна пам'ять дозволяє не робити великих запасів пам'яті для кожного порту (на випадок перевантажень). Поділювана пам'ять простіше реалізується в комутаторах на одній платі (шина пам'яті сугубо локальна).

На практиці використовуються і комбінації цих основних способів – наприклад, модулі з комутаційними матрицями можуть бути зв'язані між собою об'єднуючою шиною.

Конструктивно комутатори можуть мати кілька варіантів виконання залежно від їхнього призначення і продуктивності.

Комутатори з фіксованим числом портів є найдешевшими пристроями, які застосовуються для числа портів від 24 до 30. Часто 1-2 порти мають швидкість, на порядок більшу швидкості основної маси портів. Ці порти призначаються для підключення пріоритетних вузлів (серверів) і зв'язку з іншими комутаторами. Більш дорогі моделі можуть мати кілька гнізд для підключення різних інтерфейсних модулів, у тому числі оптичних, з резервуванням ліній і т.п. У великих мережах такі комутатори застосовуються на рівні поверхових розподільників, у малих мережах вони можуть бути і центральними пристроями.

Модульні комутатори можуть мати до сотні портів (залежно від розміру шасі, щільності портів модулів і продуктивності). Ці комутатори застосовуються як магістральні на рівні розподільників будинку. Вартість

порту знижується в міру збільшення числа встановлених модулів (накладні витрати на шасі великі), але однаково залишається вищою, ніж при використанні пристроїв з фіксованою конфігурацією. Продуктивність, як правило, теж є вищою.

Стекові комутатори в ідеалі повинні мати пропускну здатність стекового інтерфейсу не нижче суми пропускної здатності половини портів усіх комутаторів, які поєднані у стек. На практиці цей інтерфейс стає вузьким місцем, і кількість поєднуваних пристроїв часто обмежується чотирма. Топологія з'єднань пристроїв стека може бути різною: шина, кільце або зірка. При зв'язку в шину відмова одного пристрою може призвести до розпаду стека на дві незв'язані частини. Цей недолік усувається при об'єднанні пристроїв у кільце. І в шині, і в кільці пропускна здатність стекового інтерфейсу розділяється всіма пристроями. Цього недоліку дозволяє уникнути побудова стека за допомогою спеціального матричного комутатора, до якого підключаються поєднувані комутатори.

Теоретично пропускна здатність такого стека обмежується сумою пропускних здатностей комутаторів або сумою пропускних здатностей їх стекових інтерфейсів (меншою із цих сум). Комутатори, об'єднані в стек, являють собою один вузол у правилі 5–4–3.

9.5 Додаткові функції комутаторів

Крім забезпечення зв'язку між портами, комутатори можуть мати ряд додаткових можливостей:

1) SB – індикація рівня завантаження і колізій (лінійка світлодіодів або індикаторів перевищення критичних рівнів);

2) індикація стану портів. Для кожного порту відображається активність, швидкість (10/100, якщо можливий вибір), режим (напівдуплекс/дуплекс), стан (дозволений, заборонений вручну або ізольований автоматично через несправність або при порушенні захисту);

3) керованість (*management*) – можливість вилученого спостереження за станом портів і сегментація (відключення) портів за командою оператора, керування захистом;

4) моніторинг – збір статистики за портами і пристроями в цілому. У найпростішому випадку комутатор забезпечується індикатором рівня завантаження сегмента (*network utilization*) – лінійкою світлодіодів і індикатором колізій. Комутатор повинен відслідковувати стан і завантаження кожного порту. Для моніторингу пристрою можуть підтримуватися всі або частина функцій *RMON* та повинна бути можливість відбиття портів (*port mirroring*);

5) сегментованість комутатора (*segmented hub*) – можливість організації в одному фізичному пристрої декількох ізольованих сегментів. Кожний порт при конфігуруванні підключається до одного із сегментів. При необхідності зв'язок між сегментами забезпечується або зовнішніми

пристроями (мостами, комутаторами, маршрутизаторами), або внутрішніми мостами (якщо такі є);

6) підтримка трьох швидкостей – незалежний (можливо, автоматичний) вибір швидкості роботи (10, 100, 1000 Мбіт/с) кожного порту. Для комутаторів ця властивість цілком природна;

7) автоматичний вибір швидкості (10/100/1000) і режиму (напівдуплекс/дуплекс) роботи кожного порту (*autosence*). При цьому стандартний протокол узгодження може доповнюватися інтелектом (*smart auto-sensing*) – якщо за протоколом встановлена швидкість 100 Мбіт/с, а погана лінія (категорії 3) призводить до великої кількості помилок, то вибирається швидкість 10 Мбіт/с. Автоматичне узгодження працює не завжди, і багато адміністраторів віддають перевагу ручному конфігуруванню;

8) автоматична корекція полярності пар (для портів *RJ*-45) дозволяє використовувати лінії з переплутаними проводами в парі. Корисність цієї властивості сумнівна. Якщо такий пристрій, який нормально працює з «неправильною» лінією, раптом буде замінений звичайним,то проблема зі зв'язком виникне зненацька. Краще всі лінії привести у відповідність до стандарту до підключення активного устаткування;

9) можливість об'єднання в стек. Для комутаторів (*stackable Switch*), залежно від можливостей керування, об'єднання портів у єдиний сегмент може бути як безумовним, так і керованим. Якщо порти поєднуються, то весь стек з погляду правил побудови мережі 5-4-3 виступає в ролі єдиного комутатора, що є особливим для 100 Мбіт/с;

10) захист від несанкціонованого доступу – дозвіл роботи портів тільки з певними *MAC*-адресами вузлів. Для комутаторів при включеному захисті трансляція кадрів здійснюється тільки в порт адресата призначення, в інші порти посилає "порожній" кадр, який позначає зайнятість середовища передачі. Ці можливості більш характерні для комутаторів, але зустрічаються і у дорогих моделях інтелектуальних хабів.

9.6 Технологія *Fast Ethernet*

Комітет *IEEE* 802.3 прийняв специфікацію *Fast Ethernet* як стандарт 802.3u, що не є самостійним стандартом, а являє собою доповнення до існуючого стандарту 802.3 у вигляді розділів з 21 по 30.

Всі відмінності технології *Fast Ethernet* від *Ethernet* зосереджені на фізичному рівні (рис. 9.1). Рівні *MAC* і *LLC* в *Fast Ethernet* залишилися абсолютно тими ж.

Офіційний стандарт 802.3u установив три різні специфікації для фізичного рівня *Fast Ethernet* і дав їм наступні назви:

1. 100*Base-TX* для двопарного кабелю на неекранованій крученій парі *UTP* категорії 5 або екранованій крученій парі *STP Type* 1;

2. 100*Base-T*4 для чотирипарного кабелю на неекранованій крученій парі *UTP* категорії 3, 4 або 5;

3. 100*Base-FX* для багатомодового оптоволоконного кабелю, який використовує два волокна.

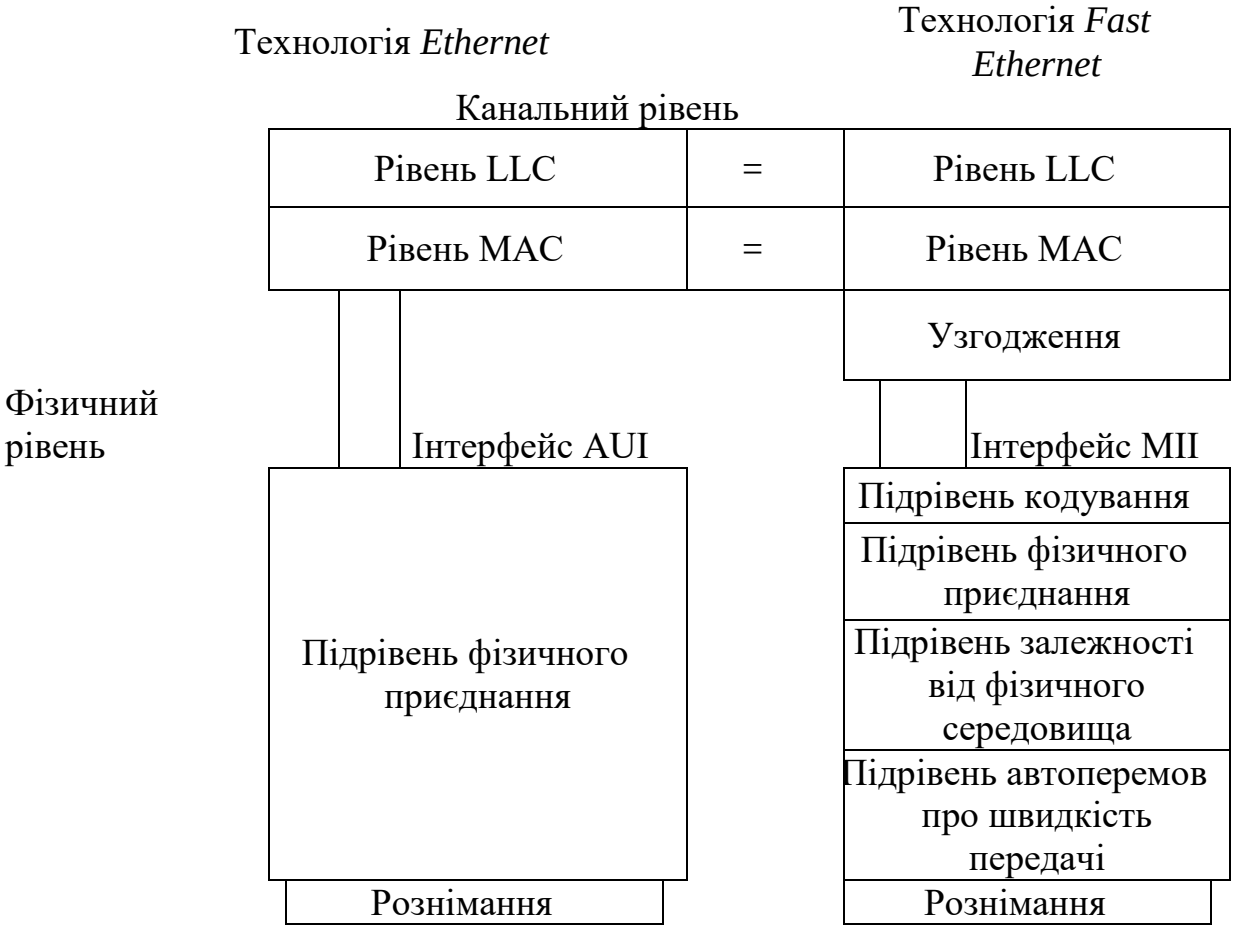

Рисунок 9.1 – Відмінності технологій *Ethernet* і *Fast Ethernet*

Для всіх трьох стандартів справедливі наступні твердження і характеристики:

1) формати кадрів технології *Fast Ethernet* не відрізняються від форматів кадрів технологій 10Мбітного *Ethernet*;

2) міжкадровий інтервал (*IPG*) дорівнює 0,96 мкс, а бітовий інтервал дорівнює 10 нс. Всі часові параметри алгоритму доступу (інтервал відстрочки, час передачі кадру мінімальної довжини і т.п.), виміряні в бітових інтервалах, залишилися такими ж, тому зміни в розділи стандарту, який стосується рівня *MAC*, не вносилися;

3) ознакою вільного стану середовища є передача нею символу *Idle* відповідного надлишкового коду (а не відсутність сигналів, як у стандартах *Ethernet* 10 Мбіт/с). Фізичний рівень включає три елементи:

3.1) рівень узгодження (*reconciliation sublayer*);

3.2) незалежний від середовища інтерфейс (*Media Independent Interface, MII*);

3.3) пристрій фізичного рівня (*Physical layer device, PHY*).

Рівень узгодження потрібний для того, щоб рівень *MAC*, розрахований на інтерфейс *AUI*, зміг працювати з фізичним рівнем через інтерфейс *MII.*

Пристрій фізичного рівня (*PHY*) складається, у свою чергу, з декількох підрівнів:

1) підрівня логічного кодування даних, що перетворює байти, які надходять від рівня *MAC*, в символи коду 4B/5B або 8B/6T (обидва коди використовуються в технології *Fast Ethernet*);

2) підрівня фізичного приєднання і підрівня залежності від фізичного середовища (*PMD*), які забезпечують формування сигналів відповідно до методу фізичного кодування, наприклад *NRZI* або *MLT*-3;

3) підрівня автоперемов, який дозволяє двом взаємодіючим портам автоматично вибрати найбільш ефективний режим роботи, наприклад, напівдуплексний або повнодуплексний (цей підрівень є факультативним).

Версія фізичного рівня **100*BaseFX*** визначає роботу з багатомодовом оптоволокном при довжині хвилі 1300 нм. Одним волокном передаються дані від вузла, другим – до вузла. З *FDDI* запозичений метод логічного кодування 4B/5B і метод фізичного кодування *NRZI*. У напівдуплексному режимі максимальна відстань між вузлами – 412 м, при повному дуплексі – 2 км (при одномодовому оптоволокні – до 32 км). 100*BaseFX* несумісний з 10*BaseFL*, оскільки використовує іншу довжину хвилі.

Версія фізичного рівня **100*BaseTX*** визначає роботу із крученою парою (*UTP Cat* 5 або *STP Type* 1). Використовується дві пари, призначення провідників (розведення по контактах рознімань) повністю збігається з 10Мбітним *Ethernet*. Максимальна довжина сегмента – 100 м. Логічне кодування – 4B/5B, фізичне кодування – *MLT*-3.

Версія фізичного рівня **100*BaseT*4** визначає роботу з більш розповсюдженою і дешевою крученою парою (*UTP Cat* 3). Використовується чотири пари: трьома парами передаються дані, четверта використовується для прослуховування несучої частоти і виявлення колізій. Логічне кодування – 8B/6T. При швидкості 33,3 Мбіт/с вкладається в смугу 16МГц (кручена пара, 3-тя категорія). Оскільки передача йде одночасно трьома парами, то сумарна швидкість становить 100 Мбіт/с. Максимальна довжина сегмента – 100 м.

Пристрої, які підтримують 100*BaseTX* або 100*BaseT*4, повинні мати функцію узгодження режимів (*auto-negotiation*), що дозволяє вибрати найефективніший режим, обом доступним учасникам обміну. Усього визначено 5 режимів (перераховані в порядку зростання пріоритету): 10*BaseT*, дуплексний 10*BaseT*, 100*BaseTX*, 100*BaseT*4, дуплексний 100*BaseTX*.

Для узгодження вузли передають пачки з 17-ти імпульсів *FLP* (*Fast Link Pulse*), у яких міститься слово, яке кодує найбільш пріоритетний з доступних режимів роботи. Якщо один з вузлів посилає імпульси *NLP* (використовувані в 10*BaseT* для контролю цілісності лінії), то другий вузол розуміє, що єдиний можливий режим – це 10*BaseT*.

Існує ще кілька версій *Fast Ethernet*, з яких становить інтерес **100*BaseSX***, яка працює з багатомодовим оптоволокном на довжині хвилі 830 нм (максимальна довжина кабльного сегмента – 300 м). Цей стандарт сумісний з 10*BaseFL* і підтримує автоматичне узгодження швидкості передачі 10/100 Мбіт/с.

Залежно від того, який набір методів логічного кодування підтримує повторювач, він належить до класу I (підтримує і 4B/5B, і 8B/6T) або до класу II (підтримує тільки один з методів). Повторювачі класу I порти всіх трьох типів - 100*BaseFX*, 100*BaseTX*, 100*BaseT*4, але за рахунок необхідності перетворення схем кодування вносять більшу затримку – до 140 *bt*. Тому в одному домені колізій може бути не більше одного повторювача класу I.

Повторювачі класу II мають або порти 100*BaseFX* і 100*BaseTX*, або тільки порти 100*BaseT*4. Внесена затримка може досягати 92 *bt* (*TX/FX*) або 67 *bt* (*T*4). Максимальна кількість повторювачів класу II в одному домені колізій – 2. Оскільки максимальний діаметр домену колізій *Fast Ethernet* для крученої пари – 205 м, а кожна станція може бути розташована від повторювача на відстані до 100 м, у найгіршому разі повторювачі (тільки класу II) повинні з'єднуватися кабелем довжиною не більше 5 м.

9.7 Технологія *Gigabit Ethernet*

Технологія *Gigabit Ethernet* описується двома стандартами: *IEEE* 802.3*z* (1998 рік) і *IEEE* 802.3*ab* (1999 рік). Розроблювачі стандартів намагалися максимально зберегти ідеї класичного *Ethernet*. Як і при переході від *Ethernet* до *Fast Ethernet*, основне нововведення складалося в десятикратному (у порівнянні з *Fast Ethernet*) зменшенні тривалості бітового інтервалу – до 1нс. Для того щоб зберегти максимальний діаметр домену колізій на рівні 200м, довелося збільшити мінімальний розмір кадру з 64 до 512 байт (4096 *bt*). Якщо передається короткий кадр, то його поле даних повинне бути доповнене до необхідної довжини забороненими символами. З іншого боку, ці обмеження диктуються необхідністю розпізнавання колізій, що істотно тільки для напівдуплексного режиму роботи. Для *Gigabit Ethernet* більш характерний дуплексний режим, при якому довжина кабельного сегмента обмежується не часом подвійного оберту, а загасанням сигналу і частотними властивостями лінії.

Для зниження накладних витрат при передачі коротких кадрів (наприклад, підтверджень прийому пакетів), передбачений пакетний режим передачі (*Burst Mode*). Вузол може передати підряд кілька невеликих кадрів (не доповнюючи кожного з них до 512 байт), сумарною довжиною не більше 8192 байт. Окремі кадри в такій групі можуть бути адресовані різним одержувачам.

Стандарт *IEEE* 802.3*z* визначає наступні версії: 1000*BaseSX*, 1000*BaseLX*, 1000*BaseCX*.

Версія **1000*BaseSX*** визначає роботу з багатомодовим оптоволокном на довжині хвилі 850 нм. Максимальна довжина сегмента при роботі в напівдуплексному режимі становить 100 м. У дуплексному режимі максимальна довжина кабелю залежить від його смуги пропущення і може досягати 800 м.

Версія **1000*BaseLX*** визначає роботу з багатомодовим або одномодовим оптоволокном при довжині хвилі 1310 нм. Максимальна довжина сегмента для одномодового волокна досягає 5 км, а для багатомодового – 550 м.

Версія **1000*BaseCX*** використовує як середовище передачі твінксіальний (*twinaxial*) кабель, який являє собою два коаксіальних кабелі (хвильовий опір 75 Ом) у загальному обплетенні. Такими кабелем можна організувати тільки напівдуплексний режим. Максимальна довжина сегмента становить 25 м, тому такий кабель найбільше застосовується для зв'язку устаткування в межах однієї кімнати.

Стандарт ***IEEE* 802.3*ab*** визначає версію *Gigabit Ethernet* на крученій парі 5 категорії – 1000Base*T*. Сигнал фізично кодується з використанням 5 рівнів потенціалу (код *PAM*-5) і передається одночасно чотирма парами. Код *PAM*-5 на тактовій частоті 125 МГц укладається в смугу пропущення 100 МГц кабелю 5 категорії. Для дуплексного режиму передача ведеться одночасно в обидва напрямки, а для виділення прийнятого сигналу приймач віднімає із прийнятої суміші сигналів свій власний сигнал. Для виконання цієї операції використовуються цифрові сигнальні процесори.

9.8 Технологія 10*Gigabit Ethernet*

Хоча *Ethernet* на 1 Гбіт/с все ще не використовував всі свої можливості, реалізовано вже 10Гбітний *Ethernet* (***IEEE* 802.3*ae***, 10*GBase-LW* або 10*GBase-ER*).

Цей стандарт затверджений у червні 2002 року і у випадку використання для побудови регіональних каналів відповідає специфікаціям *OC*-192c/*SDH VC*-4-46c (*WAN*). Випробувано канал довжиною 200 км із 10-ма сегментами. Існує серійне мережне устаткування, яке забезпечує надійну передачу при швидкості 10 Гбіт/с та довжині одномодового кабелю 10 км (λ=1310 *nm*). При роботі з оптичними волокнами можуть застосовуватися лазери з вертикальними резонаторами і поверхневим випромінюванням ***VCSEL*** (*Vertical Cavity Surface Emitting Laser*).

У випадку мультимодових варіантів використовуються волокна із градієнтним коефіцієнтом переломлення. У протоколі 10Гбіт/с *Ethernet* передбачений інтерфейс *chip-to-chip* (802.*3ae-XAUI* – букви ***ae*** означають тут *Ethernet Alliance*). Такі канали можуть використовуватися і в *LAN* для з'єднання перемикачів мережних кластерів. З'єднання організується за схемою «точка-точка».

Стандартизовано порти: 10*GBase-LR* (до 10 км за одномодовим волокном, для високопродуктивних магістральних і корпоративних

каналів), 10*GBase-ER* (до 40 км за одномодовим волокном), 10*GBase-SR* (до 28 м за мультимодовим волокном для з'єднань перемикачів один з одним), а також 10*GBase-LX*4 (до 300 м за мультимодовим волокном стандарту *FDDI* – для мереж у межах одного будинку).

В 10*GBase* для локальних мереж застосовується кодування 64B/66B (замість 8B/10B, яке використовується у звичайному гігабітному *Ethernet*), тому що стара схема дає 25 % збільшення паразитного трафіку.

У версії 10*GBase-X*4 використовується кодування 8B/10B. Там формуються 4 потоки по 3,125 Гбіт/с, які передаються одним волокном (1310 нм) із залученням техніки мультиплексування довжин хвиль (*WWDM*). У випадку 10*GBase-W* на рівні *MAC* вводиться більша мінімальна довжина *IPG*.

Контрольні запитання

1. Які функції мережного адаптера?
2. Які функції концентратора, мосту, комутатора?
3. Які специфікації фізичного середовища в технології *Fast Ethernet*?
4. У чому полягає відмінність технології *Fast Ethernet* від технології *Ethernet*?
5. Які специфікації фізичного середовища в технології *Gigabit Ethernet*?

10 АДРЕСАЦІЯ В *IP*-МЕРЕЖАХ. ПРОТОКОЛИ *IP, ARP, RARP, IPV*6

У стеку *TCP/IP* використовуються три типи адрес: локальні (що називаються також апаратними), *IP*-адреси і символьні доменні імена.

1. Під **локальною адресою** розуміють такий тип адреси, який використовується засобами базової технології для доставки даних у межах підмережі, що є елементом складеної інтермережі. Якщо підмережею інтермережі є локальна мережа, то локальна адреса – це *MAC*-адреса. *MAC*-адреса призначається мережним адаптерам і мережним інтерфейсам маршрутизаторів. *MAC*-адреси призначаються виробниками устаткування і є унікальними, тому що керуються централізовано. Для всіх існуючих технологій локальних мереж *MAC*-адреса має формат 6 байтів, наприклад: 00-E0-24-12-45-21. Однак протокол *IP* може працювати і над протоколами більше високого рівня, наприклад над протоколом *IPX* або *X*.25. У цьому випадку локальними адресами для протоколу *IP* відповідно будуть адреси *IPX* і *X*.25. Варто врахувати, що комп'ютер у локальній мережі може мати кілька локальних адрес навіть при одному мережному адаптері. Деякі мережні пристрої не мають локальних адрес. Наприклад, до таких пристроїв належать глобальні порти маршрутизаторів, призначені для з'єднань типу «точка-точка».

2. ***IP*-адреси** являють собою основний тип адрес, на підставі яких мережний рівень передає пакети між мережами. Ці адреси складаються з 4 байт (*IPv*4), наприклад 91.12.56.22. *IP*-адреса призначається адміністратором під час конфігурування комп'ютерів і маршрутизаторів. *IP*-адреса складається із двох частин: номера мережі і номера вузла. Номер мережі може бути обраний адміністратором довільно або призначений за рекомендацією спеціального підрозділу *Internet* (*Internet Network Information Center, InterNIC*), якщо мережа повинна працювати як складова частина *Internet*. Номер вузла в протоколі *IP* призначається незалежно від локальної адреси вузла. Маршрутизатор за визначенням входить відразу в кілька мереж. Тому кожний порт маршрутизатора має власну *IP*-адресу. Кінцевий вузол також може входити в декілька *IP*-мереж. У цьому випадку комп'ютер повинен мати декілька *IP*-адрес, за числом мережних зв'язків. Таким чином, *IP*-адреса характеризує не окремий комп'ютер або маршрутизатор, а одне мережне з'єднання.

3. **Символьні доменні імена.** Символьні імена в *IP*-мережах називаються доменними і будуються за ієрархічною ознакою. Складові повного символьного імені в *IP*-мережах розділяються крапкою і перераховуються в такому порядку: спочатку просте ім'я кінцевого вузла, потім ім'я групи вузлів (наприклад, ім'я організації), потім ім'я більшої групи (піддомену) і так до імені домену найвищого рівня (наприклад, домену об'єднувальної організації за географічним принципом: *UA* – Україна, *RU* – Росія, *US* – США). Прикладом доменного імені може бути ім'я cvk.gov.ua. Між доменним ім'ям і *IP*-адресою вузла немає ніякої алгоритмічної відповідності, тому необхідно використовувати якісь

додаткові таблиці або служби, щоб вузол мережі однозначно визначався як за доменним ім'ям, так і за *IP*-адресою. У мережах *TCP/IP* використовується спеціальна розподілена служба *Domain Name System* (*DNS*), яка встановлює цю відповідність на підставі створюваних адміністраторами мережі таблиць відповідності. Тому доменні імена називають також *DNS*-іменами

10.1 Класи *IP*-адрес

IP-адреса має довжину 4 байти (*IPv*4) і зазвичай записується у вигляді чотирьох чисел, які подають значення кожного байта в десятковій формі, які розділені крапками (рис. 10.1), наприклад, 93.123.12.52 – традиційна десяткова форма подання адреси. Адреса складається із двох логічних частин – номера мережі і номера вузла в мережі. Яка частина адреси належить до номера мережі, а яка – до номера вузла визначається значеннями перших біт адреси. Значення цих біт є також ознаками того, до якого класу належить та або інша *IP*-адреса. Якщо адреса починається з 0, то мережу відносять до класу A і номер мережі займає один байт, інші 3 байти інтерпретуються як номер вузла в мережі. Мережі класу A мають номера в діапазоні від 1 до 126. (Номер 0 не використовується, а номер 127 зарезервований для спеціальних цілей, про що буде сказано нижче.) Мереж класу A небагато, зате кількість вузлів у них може досягати 2^{24}, тобто 16 777 216 вузлів.

Якщо перші два біти адреси дорівнюють 10, то мережа належить до класу B. У мережах класу B під номер мережі і під номер вузла виділяється по 16 біт, тобто по 2 байти. Таким чином, мережа класу B є мережею середніх розмірів з максимальним числом вузлів 2^{16}, що становить 65 536 вузлів.

Клас	Байти									
	1						2		3	4
A	0	№ мережі					№ вузла			
B	1	0	№ мережі				№ вузла			
C	1	1	0	№ мережі			№ вузла			
D	1	1	1	0	Групова адреса					
E	1	1	1	1	0	Резерв				

Рисунок 10.1 – Структура *IP*-адреси

Якщо адреса починається з послідовності 110, то це мережа класу C. У цьому випадку під номер мережі виділяється 24 біти, а під номер вузла – 8 біт. Мережі цього класу найпоширеніші, число вузлів у них обмежено 2^8, тобто 256 вузлами.

Якщо адреса починається з послідовності 1110, то вона є адресою класу D і позначає особливу, групову адресу – *multicast*. Якщо в пакеті як

адреса призначення зазначена адреса класу D, то такий пакет повинен одержати всі вузли, яким присвоєна дана адреса.

Якщо адреса починається з послідовності 11110, то це значить, що дана адреса належить до класу E. Адреси цього класу зарезервовані для майбутніх застосувань.

Розподіл адресного простору наведено в таблиці 10.1.

Таблиця 10.1 – Розподіл адресного простору

Клас	Діапазон адрес мереж		Кількість вузлів
A	1.0.0.0	126.0.0.0	2^{24}
B	128.0.0.0	191.255.0.0	2^{16}
C	192.0.0.0	223.255.255.0	2^{8}
D	224.0.0.0	239.255.255.255	групова адреса
E	240.0.0.0	247.255.255.255	резерв

10.2 Особливі *IP*-адреси

У протоколі *IP* існує кілька угод про особливу інтерпретацію *IP*-адрес:

1. Якщо вся *IP*-адреса складається тільки із двійкових нулів, то вона позначає адресу того вузла, який згенерував цей пакет; цей режим використовується тільки в деяких повідомленнях *ICMP*.
2. Якщо в полі номера мережі стоять тільки нулі, то за замовченням вважається, що вузол призначення належить тій же самій мережі, що і вузол, який відправив пакет.
3. Якщо всі двійкові розряди *IP*-адреси дорівнюють 1, то пакет з такою адресою призначення повинен розсилатися всім вузлам, які знаходяться у тій же мережі, що і джерело цього пакета. Таке розсилання називається обмеженим широкомовним повідомленням (*limited broadcast*).
4. Якщо в полі номера вузла призначення стоять тільки одиниці, то пакет, що має таку адресу, розсилається всім вузлам мережі із заданим номером мережі. Наприклад, пакет з адресою 171.64.255.255 доставляється всім вузлам мережі 171.64.0.0. Таке розсилання називається широкомовним повідомленням (*broadcast*).

Таким чином, ні номер мережі, ні номер вузла не може складатися з одних нулів або одних одиниць. Це обмежує кількість вузлів у мережі співвідношенням:

$$N_{\text{вузлів}} = 2^n - 2, \qquad (10.1)$$

де n – кількість біт у полі номера вузла.

Кількість мереж обмежується співвідношенням:

$$N_{\text{мереж}} = 2^m - 2, \qquad (10.2)$$

де m – кількість біт у полі номера мережі.

Наприклад, кожна з 16382 (2^{14}-2) мереж класу B (14 біт під номер мережі, 16 біт під номер вузла) максимально може включати 65534 (2^{16}-2) вузла з номерами від x.x.0.1 до x.x.255.254.

Крім того, виділяється група адрес, перший байт яких дорівнює 127. Ці адреси використовуються для передачі даних між процесами на одному комп'ютері або для тестування. Дані, відправлені за такою адресою, розглядаються, як тільки що прийняті з мережі, у результаті чого утвориться наче б то "петля" (*loopback*). Найчастіше використовується адреса 127.0.0.1, але для цих цілей можна використовувати будь-яку адресу виду 127.x.x.x.

10.3 Протокол *IP*

Протокол *IP* (*Internet Protocol*, Протокол міжмережевої взаємодії), описаний в *RFC* 791. Основна функція протоколу *IP* – передача пакетів між вузлами, які належать до різних мереж, через проміжні мережі. Кожний пакет (дейтаграма – *datagram*, у термінології *TCP/IP*) обробляється незалежно від інших. Доставка дейтаграм не гарантується. Можливі втрати дейтаграм, доставка з помилками, дублювання і порушення порядку проходження. Друга функція протоколу *IP* – виконання фрагментації пакетів при передачі їх між мережами з різним максимально допустимим розміром поля даних кадру (*MTU*).

Істотна властивість протоколу *IP* складається в нетрадиційному порядку передачі бітів: байт передається, починаючи зі старшого біта. Крім того, нумерація бітів у байті також починається зі старшого: найстарший біт має номер 0, наймолодший – номер 7. Дейтаграма (*IP*-пакет) складається із заголовка і поля даних. Формат заголовка наведений на рис. 10.2.

<table>
<tr><td>Номер версії</td><td>Довжина заголовка</td><td>Тип сервісу</td><td colspan="2">Загальна довжина</td></tr>
<tr><td colspan="3">Ідентифікатор "великого пакета"</td><td>Прапори</td><td>Зсув фрагмента</td></tr>
<tr><td colspan="2">Час життя</td><td>Протокол</td><td colspan="2">Контрольна сума заголовка</td></tr>
<tr><td colspan="5">Адреса відправника</td></tr>
<tr><td colspan="5">Адреса одержувача</td></tr>
<tr><td colspan="2">Опції
(змінна довжина)</td><td colspan="3">Вирівнювання до 32-бітної границі
(заповнення нулями)</td></tr>
</table>

Рисунок 10.2 – Формат заголовка *IP*-пакета

1. Номер версії (*Version*) [4 біти] – вказує використовуваний формат заголовка. У цей час основна використовувана версія має номер 4.

2. Довжина заголовка (*Internet Header Length*) [4 біти] – довжина заголовка в 32-бітних словах, мінімальне допустиме значення – 5

(відповідає довжині заголовка в 20 байт). Максимальна довжина заголовка – 60 байт.

3. Тип сервісу (*Type of Service*) [8 біт] – вказує на бажані параметри якості обслуговування. Формат байта типу сервісу наведений на рис. 10.3.

4. Поле Пріоритету (*Precedence*) для звичайних пакетів дорівнює 0, інші значення (від 1 до 7) використовуються для службових цілей, чим більше значення, тим вище пріоритет (Рис. 10.3).

0	1	2	3	4	5	6	7
Пріоритет			D	T	R	0	0

Рисунок 10.3 – Структура поля “Пріоритет” заголовка *IP*-пакета

5. Поля *D* (*Delay* – затримка), *T* (*Throughput* – пропускна здатність) і *R* (*Reliability* – надійність) використовуються для вказівки найбільш важливого для передавального вузла параметра якості. Вибір відбувається між малою затримкою, великою пропускною здатністю і високою надійністю. Відповідний біт (біти) встановлюється в 1, інші – в 0. Як правило, поліпшення одного з параметрів пов’язане з погіршенням інших.

6. Загальна довжина (*Total Length*) [16 біт] – довжина дейтаграми (заголовка і даних) у байтах. Хоча розмір поля дозволяє створювати дейтаграми довжиною до 65535 байт, стандарт вимагає, щоб будь-який вузол міг приймати як мінімум 576-байтні дейтаграми, а відправляти дейтаграми більшої довжини тільки, будучи впевненим, що одержувач може їх прийняти.

7. Ідентифікатор “великого пакета” (*Identification*) [16 біт] – значення, однакове для всіх дейтаграм, що містять фрагменти одного пакета (“великого пакета”).

8. Прапори (*Flags*) [3 біти] – прапори, що керують фрагментуванням: 0 біт – зарезервований, повинен бути 0; 1 біт (*DF, Don’t Fragment*) – “0” = можна фрагментувати, “1” = не можна фрагментувати; 2 біт (*MF, More Fragments*) – “0” = останній фрагмент, “1” = ще будуть фрагменти.

9. Зсув фрагмента (*Fragment Offset*) [13 біт] – указує на місце у “великому пакеті”, з якого починаються дані поточної дейтаграми. Вимірюється в 64-бітних словах. Наприклад, зсув фрагмента, рівний 2, означає, що дані поточної дейтаграми повинні перебувати у “великому пакеті”, починаючи з 16-го байта. Перший фрагмент має нульовий зсув.

10. Час життя (*Time to Live, TTL*) [8 біт] – максимальний час перебування дейтаграми в мережі. Кожний маршрутизатор повинен зменшувати це значення на одиницю і відкидати дейтаграми зі значенням *TTL* = 0 (сповістивши про це відправникові). Наявність цього поля забезпечує знищення дейтаграми, “яка зациклилися” або “заблудилася”. Поле *TTL* також дозволяє обмежити дальність поширення дейтаграми (це

зручно, наприклад, при одночасній передачі безлічі абонентів) і є основою для роботи утиліти *traceroute*.

11. Протокол (*Protocol*) [8 біт] – вказує, дані якого протоколу верхнього рівня передаються в дейтаграмі. Можливі значення цього поля стандартизовані (*RFC* "*Assigned Numbers*"), деякі з них: 1 – *ICMP*, 4 – *IP*, 6 – *TCP*, 17 – *UDP*, 89 – *OSPF*.

12. Контрольна сума заголовка (*Header Checksum*) [16 біт] – контрольна сума всіх полів заголовка, яка обчислюється як доповнення суми всіх 16-бітових слів заголовка (з нульовими бітами в полі контрольної суми). Оскільки деякі поля заголовка (наприклад, час життя) змінюються при передачі дейтаграми через мережу, контрольна сума перераховується кожним маршрутизатором. Якщо отримано дейтаграму з неправильною контрольною сумою, така дейтаграма відкидається.

13. Адреса відправника (*Source Address*) [32 біта] – *IP*-адреса відправника дейтаграми.

14. Адреса одержувача (*Destination Address*) [32 біта] – *IP*-адреса одержувача дейтаграми.

15. Опції (*Options*) [змінна довжина] – необов'язкове поле, може містити дані про безпеку, маршрут дейтаграми (при маршрутизації від джерела) і т.д. В одній дейтаграмі може бути кілька опцій, кожна з яких складається з коду опції (1 байт), довжини опції (1 байт) і байтів даних опції. Якщо для опції не потрібні додаткові дані, вона складається з одного байта – коду опції.

Код опції складається з трьох полів:

0 біт ("Копіювати") – "0" = копіювати опції в усі фрагменти, "1" = копіювати опції тільки в перший фрагмент

1–2 біти ("Клас опції ") – 0 = керування дейтаграмами/мережею, 2 = налагодження мережі, 1 і 3 = зарезервовані.

3–7 біти ("Номер опції") – номер опції усередині класу, так для класу 0 визначено 7 номерів опцій, які несуть маршрути і дані про безпеку, а для класу 2 – тільки один номер опції 4, що несе тимчасові мітки, які використовуються при протоколюванні проходження дейтаграми за маршрутом.

10.4 Порядок розподілу *IP*-адрес

Номера мереж призначаються або централізовано, якщо мережа є частиною *Internet*, або довільно, якщо мережа працює автономно. Номера вузлів і в тому і в іншому випадку адміністратор вільний призначати за своїм розсудом, не виходячи, зрозуміло, з дозволеного для цього класу мережі діапазону.

Якщо ж деяка *IP*-мережа створена для роботи в «автономному режимі», без зв'язку з *Internet*, тоді адміністратор цієї мережі вільний призначити їй довільно обраний номер. Але і у цій ситуації для того щоб уникнути яких-небудь колізій, у стандартах *Internet* визначено кілька

діапазонів адрес, які рекомендуються для локального використання. Ці адреси не обробляються маршрутизаторами *Internet* ні за яких умов. Адреси, зарезервовані для локальних цілей, обрані з різних класів; у класі А – це мережа 10.0.0.0, у класі В – це діапазон з 16 номерів мереж 172.16.0.0-172.31.0.0, у класі С – це діапазон з 255 мереж – 192.168.0.0-192.168.255.0.

10.5 Фрагментація *IP*-пакетів

На шляху пакета від відправника до одержувача можуть зустрічатися локальні і глобальні мережі різних типів з різними допустимими розмірами полів даних кадрів канального рівня (*Maximum Transfer Unit - MTU*). Так, мережі *Ethernet* можуть передавати кадри, що несуть до 1500 байт даних. Для мереж *X*.25 характерний розмір поля даних кадру в 128 байт. Мережі *FDDI* можуть передавати кадри розміром в 4500 байт. В інших мережах діють свої обмеження. Протокол *IP* вміє передавати дейтаграми, довжина яких більше *MTU* проміжної мережі, за рахунок фрагментування – розбивання “великого пакета” на деяку кількість частин (фрагментів), розмір кожної з яких задовольняє проміжну мережу. Після того як всі фрагменти будуть передані через проміжну мережу, вони будуть зібрані на вузлі-одержувачі модулем протоколу *IP* назад у “великий пакет”. Відзначимо, що складання пакета із фрагментів здійснює тільки одержувач, а не який-небудь із проміжних маршрутизаторів. Маршрутизатори можуть тільки фрагментувати пакети, але не збирати їх. Це пов’язано з тим, що різні фрагменти одного пакета не обов’язково будуть проходити через ті самі маршрутизатори.

Для того щоб не переплутати фрагменти різних пакетів, використовується поле ідентифікації, значення якого повинне бути однаковим для всіх фрагментів одного пакета і не повторюватися для різних пакетів, поки в обох пакетів не минув час життя.

При розподілі даних пакета, розмір усіх фрагментів, крім останнього, повинен бути кратний 8 байтам. Це дозволяє приділити менше місця в заголовку під поле "Зсув фрагмента".

Другий біт поля Прапори (*More fragments*), якщо дорівнює одиниці, то вказує на те, що даний фрагмент є не останнім у пакеті.

Якщо пакет відправляється без фрагментації, прапор “*More fragments*” встановлюється в 0, а поле Зсув фрагмента – заповнюється нульовими бітами.

Якщо перший біт поля Прапори (*Don’t fragment*) дорівнює одиниці, то фрагментація пакета заборонена. Якщо цей пакет повинен бути переданий через мережу з недостатнім *MTU*, то маршрутизатор змушений буде його відкинути (і сповістити про це відправникові за допомогою протоколу *ICMP*). Цей прапор використовується у випадках, коли відправникові відомо, що в одержувача немає досить ресурсів з відновлення пакетів із фрагментів.

10.6 Протоколи керування міжмережевою взаємодією

В *Internet*, крім *IP* протоколу, що використовується для передачі даних, є кілька протоколів керування, використовуваних на мережному рівні, такі як *ICMP*, *ARP*, *RARP*.

10.6.1 *Протокол ICMP*.

Протокол *ICMP* (*Internet Control Message Protocol*, Протокол керуючих повідомлень інтернет) описаний в *RFC* 792.

Він використовується для повідомлень про помилки або позаштатні ситуації, переданих вузлу-відправникові дейтаграми вузлом-одержувачем або проміжним маршрутизатором.

Хоча повідомлення *ICMP* вкладаються в поле даних *IP*-дейтаграми, тобто *ICMP* як би є протоколом більш високого рівня, чим *IP*, модуль обробки *ICMP*-повідомлень входить у модуль, що реалізує протокол *IP*.

10.6.2 *Протокол ARP*.

Протокол *ARP* (*Address Resolution Protocol*, Протокол вирішення адрес) описаний в *RFC* 826.

При передачі пакетів усередині локальних мереж протоколи канального рівня користуються локальними адресами вузлів. Відправник же може знати тільки *IP*-адресу одержувача. Для того щоб визначити, яка локальна адреса (наприклад, *MAC*-адреса в мережі *Ethernet*) відповідає даній *IP*-адресі, застосовується протокол *ARP*. Цей протокол розроблявся спеціально для *Ethernet*-мереж, але може працювати в будь-яких мережах, які підтримують широкомовну передачу.

Всі вузли, які підтримують протокол *ARP*, ведуть *ARP*-таблицю, що складається із записів <*IP*-адреса; *MAC*-адреса>.

Коли вузлу потрібно визначити локальну адресу іншого вузла, його *ARP*-модуль спочатку шукає його в *ARP*-таблиці, і якщо потрібна адреса не знайдена, то передає широкомовне повідомлення: “Чи знає хто-небудь локальну адресу для *IP* 83.55.77.109? *Я* 83.55.67.101, моя *MAC*-адреса 00:10:B0:41:25:60.”. Вузол, якого розшукують, відповідає (не широкомовно, а прямою передачею): “Так, 83.55.67.101 – це я. Моя *MAC*-адреса 00:10:B0:41:81:61”. При цьому він зберігає пару <*IP*-адреса; *MAC*-адреса> того вузла, що шукав, у своєї *ARP*-таблиці. Нарешті, перший вузол, одержавши відповідь, заносить його у свою *ARP*-таблицю.

Як правило, записи в *ARP*-таблиці мають обмежений час життя (стандарт описує можливі схеми обмеження часу життя і тайм-аутів, але не вимагає їхнього застосування).

Формат повідомлення *ARP* дозволяє використовувати цей протокол у мережах з різним розміром адрес (до 256 біт).

Повідомлення *ARP* не містять *IP*-заголовка і безпосередньо розміщаються в полі даних кадру канального рівня.

10.6.3 *Протокол RARP*.

Reverse Address Resolution Protocol (*RARP*) – зворотний протокол визначення адреси. Іноді виникає зворотна проблема – відома *Ethernet*–адреса, але невідомо, яка *IP*-адреса їй відповідає. Ця проблема виникає, наприклад, при вилученому завантаженні бездискової станції. Як ця станція визначить свою і сусідні *IP*-адреси?

Вона надсилає запит до *RARP*-серверу: Моя *Ethernet*-адреса така, хто знає відповідну *IP*-адресу? *RARP*-сервер відловлює такі запити і шле відповідь.

У цьому протоколі є один істотний недолік – пакети з тим самим запитом розсилаються всім, збільшуючи накладні витрати. Для усунення цього недоліку був запропонований протокол *BOOTP*. На відміну від *RARP*, *BOOTP* використовує *UDP*-повідомлення, які розсилаються тільки маршрутизаторам. Цей протокол також використовується в бездискових станціях, у яких у пам'яті прошита *IP*-адреса виділеного маршрутизатора.

10.7 *CIDR* – безкласова маршрутизація усередині домену

Популярність *Internet* обернулася проти нього. Не стало вистачати адрес. В 1987 році вважалося що 100 000 мереж дуже багато, і це число не швидко буде досягнуто. Однак воно було перевищено в 1996 році. Проблема в тому, що адреси виділяються класами. Багато організацій не використовують усього діапазону адрес, виділеного їй класу.

Клас В, що найбільш часто використовується, занадто великий для багатьох організацій. На основі досвіду видно, що було б не погано, якби клас С мав не 256 машин, а 1024.

Інша проблема – вибухоподібне зростання таблиць маршрутизації. Маршрутизатор не повинен знати про кожну машину в мережі, але повинен знати про кожну мережу. На сьогодні півмільйона адрес класу С було виділено, отже, в таблиці маршрутизації повинно бути не менш півмільйона елементів, кожний з яких показує, як досягти ту або іншу мережу.

Крім цього, багато алгоритмів маршрутизації вимагають, щоб маршрутизатори періодично обмінювалися своїми таблицями. Чим більші ці таблиці, тим більше шансів, що при передачі вони будуть ушкоджені і передані неправильно.

Вихід – у збільшенні ієрархії адрес в Internet. Вказувати країну, область, місто, район, машину. Однак 32 біт не вистачить. Крім цього, Ліхтенштейн, наприклад, буде мати стільки ж адрес, скільки і США.

Так що кожне рішення має свої проблеми. У наш час поширюється рішення на основі протоколу *CIDR*, описаного в *RFC* 1519. Його ідея о— снована на тому, що на сьогодні не використані більше 2 мільйонів мереж класу С, тому можна виділяти за запитом організації кілька послідовних мереж класу С так, щоб покрити необхідне число машин. Наприклад, якщо

організація заявлена 2000 машин, виділити їй 8 послідовних мереж класу С, що дасть 2048 машин.

Відповідно до цього були змінені правила визначення місця для адрес класу С. Світ був поділений на чотири зони. Кожній зоні була виділена частина адрес класу С.

194.0.0.0 – 195.255.255.255 – Європа

198.0.0.0 – 199.255.255.255 – Північна Америка

200.0.0.0 – 201.255.255.255 – Центральна і Південна Америка

202.0.0.0 – 203.255.255.255 – Азія і Тихий Океан

У такий спосіб кожний регіон одержав 32 мільйони адрес для роздачі, а 320 мільйонів адрес класу С з 204.0.0.0. по 223.255.255.255 зарезервовано на майбутнє. Це істотно спростило роботу з таблицями маршрутизації. Будь-який маршрутизатор, одержавши адресу в діапазоні 194.0.0.0 по 195.255.255.255, знає, що його треба переслати по одному з європейських маршрутизаторів.

10.8 *IPv*6

Хоча *CIDR* може продовжити на кілька років існуючу версію *Internet IPv4*, але ясно, що дні її пораховані. Кількість людей, які використовують *Internet*, різко зростає. Якщо раніше це були в основному університети, держустанови, то тепер це комерційні організації, мобільні користувачі і т.п. Під тиском зростання *Internet* інженерний комітет *Internet* почав проект створення нової версії *IP*. Основними цілями цього проекту є:

- працювати з мільярдами машин, навіть при неефективному розподілі адрес;
- скоротити розмір таблиць маршрутизації;
- спростити протоколи, щоб зробити маршрутизацію швидше;
- забезпечити більш високу безпеку, чим існуючий *IP*;
- звернути більше уваги на тип сервісу, особливо для додатків реального часу;
- розширити групову адресацію, дозволивши опис групи;
- дозволити роумінг для хосту без зміни його адреси;
- дозволити еволюцію протоколів у майбутньому;
- дозволити спільне існування як старих, так і нових протоколів.

В 1993 році був опублікований протокол *SIPP - Simple Internet Protocol Plus*, який був прийнятий як *IPv*6.

Перша і головна відмінність *IPv*6 – це більша адреса довжиною 16 байт. Це вирішує одне з головних завдань – необмежене *розширення Internet*; друге – заголовок став простіший (усього 7 полів), що прискорило обробку і маршрутизацію; третє – він краще підтримує варіанти в заголовку, внаслідок чого робота з ним стає більш гнучкою, дозволяючи опускати непотрібні поля і вводити необхідні; четверте – серйозно поліпшена безпека протоколу. Ідентифікація і конфіденційність – це ключові можливості нового *IP*.

Нарешті, істотно поліпшена робота з типом сервісу, особливо з огляду на зростаючий *multimedia* трафік.

Контрольні запитання

1. Які типи адрес виділяються в стеку *TCP/IP*?
2. Яка структура *IP*-адреси?
3. Які функції виконує *IP*-протокол?
4. У чому полягає фрагментація *IP*-пакетів?
5. Для чого використовуються протоколи *ICMP*, *ARP*, *RARP*?
6. Що таке *CIDR*?
7. Для чого було створено *IPv*6?

11 ПРОТОКОЛИ ВНУТРІШНЬОЇ І ЗОВНІШНЬОЇ МАРШРУТИЗАЦІЇ (*RIP, OSPF, BGP*). ПРОТОКОЛИ ТРАНСПОРТНОГО РІВНЯ (*UDP, TCP*)

Алгоритм маршрутизації – це правило призначення вихідної лінії зв'язку (порту) на основі даних, що містяться в заголовку пакета, даних, які описують стан маршрутизатора і мережі в цілому.

Ефективність алгоритмів характеризується:

- часом доставки пакетів;
- навантаженням на мережу;
- витратами ресурсів маршрутизаторів (часу і пам'яті).

Для підвищення ефективності бажано, щоб кожний маршрутизатор мав інформацію як про топологію мережі, так і про стан вузлів і зв'язків між ними.

Класифікація алгоритмів:

1) проста маршрутизація – не змінюється при зміні топології і стану мережі:

1.1) випадкова – передача пакета на будь-який порт, крім вихідного;

1.2) лавинна – передача пакета на всі порти, крім вихідного;

1.3) за попереднім досвідом – за принципом мосту з лічильниками хопів;

2) фіксована маршрутизація – за статично заданими таблицями маршрутизації:

2.1) односпрямована;

2.2) багатоспрямована;

3) адаптивна маршрутизація – це маршрутизація з урахуванням змін стану мережі:

3.1**)** локальна – тільки на основі інформації про стан своїх вихідних каналів і черги пакетів;

3.2) розподілена – на основі інформації, що одержується від інших вузлів (регулярний обмін вузлів таблицями маршрутизації);

3.3) централізована – з виділеним центром маршрутизації, який збирає інформацію про стан вузлів і каналів і розсилає її всім вузлам;

3.4) гібридна – централізована+локальна (якщо шлях у таблиці один, то за цим шляхом, інакше – на основі довжин черг).

Адаптивні алгоритми:

1. Дистанційно-векторні (*Distance Vector Algorithms*) – розсилається вектор з метрик сусідніх мереж – *RIP*.

2. Стан зв'язків (*Link State Algorithms*) – кожний вузол будує повний граф мережі (передаються ребра графа *router–router, router–network*) – *IS-IS, OSPF, NLSP*.

11.1 Протокол *RIP*

Протокол *RIP* (*Routing Information Protocol*) є внутрішнім протоколом маршрутизації дистанційно-векторного типу. Для *IP* існує дві

версії протоколу *RIP*. Протокол *RIPv*1 не підтримує масок. Протокол *RIPv*2 поширює інформацію про маски мереж, тому він більшою мірою відповідає сучасним вимогам.

Як відстань до мережі *RIP* допускає різні види метрик: хопи, метрики, які враховують пропускну здатність, внесені затримки і надійність мереж (тобто відповідним ознакам *T*, *D* і *R* у полі «якість сервісу» *IP*-пакета). У більшості реалізацій *RIP* використовується найпростіша метрика – кількість хопів, тобто кількість проміжних маршрутизаторів, які потрібно пройти пакету мережею призначення.

У вихідному стані в кожному маршрутизаторі автоматично створюється мінімальна таблиця маршрутизації, в якій ураховуються тільки безпосередньо приєднані мережі. Після ініціалізації кожного маршрутизатора він починає посилати своїм сусідам повідомлення протоколу *RIP*, у яких міститься його мінімальна таблиця. Сусідами є ті маршрутизатори, яким даний маршрутизатор безпосередньо може передати *IP*-пакет якою-небудь своєю мережею, не користуючись послугами проміжних маршрутизаторів. *RIP*-повідомлення передаються в пакетах протоколу *UDP* і включають два параметри для кожної мережі: її *IP*-адресу і відстань до неї від маршрутизатора, який передає повідомлення. Після одержання *RIP*-повідомлень від сусідів маршрутизатор нарощує кожне отримане поле метрики на одиницю і запам’ятовує, від якого маршрутизатора і через який порт отримана нова інформація (адреса цього маршрутизатора буде адресою наступного маршрутизатора, якщо цей запис буде внесено в таблицю маршрутизації). Потім маршрутизатор починає порівнювати нову інформацію з тією, яка зберігається в його таблиці маршрутизації.

Протокол *RIP* заміщає запис про яку-небудь мережу тільки в тому випадку, якщо нова інформація має кращу метрику (відстань у хопах менше), чим уже наявна в його таблиці. У результаті про кожну мережу залишається тільки один запис. Якщо ж є декілька рівнозначних відносно відстані шляхів до однієї і тієї ж мережі, то однаково в таблиці залишається один запис, який прийшов до маршрутизатора першим за часом. Для цього правила існує виключення – якщо гірша інформація про яку-небудь мережу прийшла від того маршрутизатора, на підставі повідомлень якого була створено даний запис, то гірша інформація заміняє кращу.

На наступному етапі відбувається розсилання нової, уже немінімальної таблиці сусідам. У цьому повідомленні маршрутизатор розміщає дані про всі відомі йому мережі, як безпосередньо підключені, так і віддалені, про які він довідався з *RIP*-повідомлень. При одержанні цих повідомлень сусідні маршрутизатори обробляють інформацію, яка міститься в них, і на її підставі роблять коректування своїх таблиць маршрутизації.

Для відпрацьовування негативних змін, пов’язаних із втратою якого-небудь маршруту, використовуються два механізми:

1) витікання часу життя маршруту;

2) вказівка спеціальної відстані (нескінченності) до мережі, яка стала недоступною.

Для відпрацьовування першого механізму кожний запис у таблиці маршрутизації має час життя (*TTL - Time To Live*). Якщо за час тайм-ауту не прийде нове маршрутне повідомлення про цей маршрут, то він позначається як недійсний.

Тайм-аут працює в тих випадках, коли маршрутизатор не може послати сусідам повідомлення про маршрут, що відмовив. У випадку наявності можливості посилки повідомлення *RIP*-маршрутизатори вказують нескінченну відстань до мережі, яка обрана рівною 16 хопам (при іншій метриці необхідно вказати маршрутизатору її значення, яке вважається нескінченністю). Одержавши таке повідомлення, маршрутизатор повинен перевірити, чи виходить ця інформація від того ж маршрутизатора, повідомлення якого послужило раніше підставою для запису про дану мережу в таблиці маршрутизації.

Якщо це є тим же маршрутизатором, то маршрут позначається як недоступний.

Обмеження в 15 хопів звужує область застосування протоколу *RIP* до мереж, у яких число проміжних маршрутизаторів не може бути більше 15. Для більш масштабних мереж потрібно застосовувати інші протоколи маршрутизації, наприклад *OSPF*, або розбивати мережу на автономні області.

Головна причина нестабільної роботи маршрутизаторів, що працюють за протоколом *RIP*, пов’язана із принципом роботи дистанційно-векторних алгоритмів – користування інформацією, отриманою з інших рук. Викорінити цю причину повністю не можна, адже сам спосіб побудови таблиць маршрутизації пов’язаний з передачею чужої інформації без вказівки джерела її походження.

При реалізації *RIP* можна виділити наступні режими:

1. Ініціалізація, визначення всіх "живих" інтерфейсів шляхом посилки запитів, одержання таблиць маршрутизації від інших маршрутизаторів. Часто використовуються широкомовні запити.

2. Отримання запиту. Залежно від типу запиту посилається адресатові повна таблиця маршрутизації або проводиться індивідуальна обробка.

3. Отримання відгуку. Проводиться корекція таблиці маршрутизації (видалення, виправлення, додавання).

4. Регулярні корекції. Кожні 30 секунд уся або частина таблиці маршрутизації посилається всім сусіднім маршрутизаторам. Можуть посилатися і спеціальні запити при локальній зміні таблиці.

RIP досить простий протокол, але, на жаль, не позбавлений недоліків:

1. *RIP* не працює з адресами підмереж. Якщо нормальний 16-бітний ідентифікатор ЕОМ класу В не дорівнює 0, *RIP* не може визначити, чи є ненульова частина підмережевим ID або повною *IP*-адресою.

2. *RIP* вимагає багато часу для відновлення зв'язку після перебою в маршрутизаторі (хвилини). У процесі встановлення режиму можливі цикли.

3. Число хопів важливий, але не єдиний параметр маршруту, та і 15 хопів не межа для сучасних мереж.

11.2 Протокол маршрутизації OSPF (Open Shortest Path First)

Протокол *OSPF* є досить сучасною реалізацією алгоритму стану зв'язків (він прийнятий в 1991 році) і має багато особливостей, орієнтованих на застосування у великих гетерогенних мережах. Цей протокол, також, як і *RIP*, належить до внутрішніх протоколів маршрутизації.

Протокол *OSPF*, будучи внутрішнім протоколом маршрутизації, визначає маршрути усередині автономної системи. Автономна система може бути поділена на окремі області, кожна з яких стає об'єктом маршрутизації, а внутрішня структура зовні невидима. Цей прийом дозволяє значно скоротити необхідний обсяг маршрутної бази даних.

В *OSPF* використовується термін опорної мережі (*backbone*) для комунікацій між виділеними областями. Протокол працює лише в межах автономної системи. У межах виділеної області може працювати свій протокол маршрутизації.

У протоколі *OSPF*, як і в *RIP*, вводиться поняття «сусідів» для маршрутизаторів, яким даний маршрутизатор може передати пакет якоюсь своєю мережею, не користуючись послугами проміжних маршрутизаторів. Кожний маршрутизатор зберігає інформацію про те, у якому стані перебуває його сусід. Маршрутизатор покладається на сусідні маршрутизатори і передає їм пакети даних тільки в тому випадку, якщо він впевнений, що вони повністю працездатні. Для з'ясування стану зв'язків маршрутизатори-сусіди досить часто обмінюються короткими повідомленнями *HELLO*.

Для поширення мережею даних про стан зв'язків, маршрутизатори обмінюються повідомленнями іншого типу. Ці повідомлення називаються *Links State Advertisement* (*LSA*) – оголошення про зв'язки маршрутизатора (точніше, про стан зв'язків). *OSPF*-маршрутизатори обмінюються не тільки своїми, але і чужими оголошеннями про зв'язки, одержуючи, зрештою, інформацію про стан усіх зв'язків мережі.

Ця інформація і утворює граф зв'язків мережі, що є однаковим для всіх маршрутизаторів у мережі.

Крім інформації про сусідів, маршрутизатор у своєму оголошенні перераховує *IP*-підмережі, з якими він зв'язаний безпосередньо, тому після одержання інформації про граф зв'язків мережі обчислення маршруту до кожної мережі здійснюється безпосередньо за цим графом за алгоритмом

Дикстри. Більш точно, маршрутизатор обчислює шлях не до конкретної мережі, а до маршрутизатора, до якого ця мережа підключена.

Кожний маршрутизатор має унікальний ідентифікатор, що передається в оголошенні про стани зв'язків. Такий підхід дає можливість не витрачати *IP*-адреси на зв'язок типу «точка-точка» між маршрутизаторами, до яких не підключені робочі станції.

Маршрутизатор обчислює оптимальний маршрут до кожної мережі, але запам'ятовує тільки перший проміжний маршрутизатор з кожного маршруту. Таким чином, результатом обчислень оптимальних маршрутів є список рядків, у яких вказується номер мережі і ідентифікатор маршрутизатора, якому потрібно переслати пакет для цієї мережі. Зазначений список маршрутів і є маршрутною таблицею, але обчислений він на підставі повної інформації про граф зв'язків мережі, а не часткової інформації, як у протоколі *RIP*.

У протоколі *OSPF* підмережі поділяються на три категорії:

1) «хост-мережа», яка являє собою підмережу із однієї адреси;

2) «тупикова мережа», яка являє собою підмережу, підключену тільки до одного маршрутизатора;

3) «транзитна мережа», яка являє собою підмережу, підключену до більш ніж одного маршрутизатора.

Транзитна мережа є для протоколу *OSPF* особливим випадком. У транзитній мережі кілька маршрутизаторів є взаємно і одночасно досяжними. У локальних мережах маршрутизатор може послати одне повідомлення, яке одержать усі його сусіди. Це зменшує навантаження на маршрутизатор, коли він посилає повідомлення для визначення існування зв'язку або оновлених оголошень про сусідів.

Маршрутна таблиця *OSPF* містить у собі:

1) *IP*-адресу місця призначення і маску;

2) тип місця призначення (мережа, граничний маршрутизатор і т.д.);

3) тип функції (можливий набір маршрутизаторів для кожної з функцій *TOS*);

4) область (описує область, зв'язок з якої веде до мети, можливо кілька записів даного типу, якщо області дії граничних маршрутизаторів перекриваються);

5) тип шляху (характеризує шлях як внутрішній, міжобласний або зовнішній, ведучий до *AS*);

6) ціна маршруту до мети;

7) черговий маршрутизатор, куди варто послати дейтаграму;

8) маршрутизатор повідомлень (використовується для міжобласних обмінів і для зв'язків автономних систем один з одним).

Переваги *OSPF*:

1. Для кожної адреси може бути кілька маршрутних таблиць, по одній на кожний вид *IP*-операції (*TOS*).

2. Кожному інтерфейсу присвоюється безрозмірна ціна, яка враховує пропускну здатність, час транспортування повідомлення. Для кожної *IP*-операції може бути присвоєна своя ціна (коефіцієнт якості).

3. При існуванні еквівалентних маршрутів *OSFP* розподіляє потік рівномірно за цими маршрутами.

4. Підтримується адресація підмереж (різні маски для різних маршрутів).

5. При зв'язку «точка-точка» не потрібно *IP*-адреси для кожного з кінців.

6. Застосування мультикастінга замість широкомовних повідомлень знижує завантаження на залучені сегменти.

Недоліки:

1. Важко одержати інформацію про перевагу каналів для вузлів, які підтримують інші протоколи, або зі статичною маршрутизацією.

2. *OSPF* є лише внутрішнім протоколом.

11.3 Протокол *BGP*

Прикордонний шлюзовий протокол *BGP* (*Border Gateway Protocol*) є сьогодні основним протоколом обміну маршрутною інформацією між автономними системами Інтернету.

Протокол *BGP* дозволяє реалізувати маршрутну політику, обумовлену адміністратором *AS*. Політика відбивається в конфігураційних файлах *BGP*. Маршрутна політика це не частина протоколу, вона визначає рішення, коли місце призначення досяжне декількома шляхами, політика відбиває міркування безпеки, економічні інтереси та ін. Кількість мереж у межах однієї *AS* не лімітовано. Один маршрутизатор на багато мереж дозволяє мінімізувати таблицю маршрутів.

BGP використовує три таймери:

1. *Connectretry* (скидається при ініціалізації і корекції; 120 сек);

2. *Holdtime* (запускається при одержанні команд *Update* або *Keepalive*; 90сек);

3. *keepalive* (запускається при посилці повідомлення *Keepalive*; 30сек).

BGP відрізняється від *RIP* і *OSPF* тим, що використовує *TCP* як транспортний протокол. Дві системи, що використовують *BGP*, зв'язуються один з одним і пересилають за допомогою *TCP* повні таблиці маршрутизації. Надалі обмін іде тільки у випадку якихось змін. ЕОМ, яка використовує *BGP*, не обов'язково є маршрутизатором. Повідомлення обробляються тільки після того, як вони повністю отримані.

BGP є протоколом, що орієнтується на вектор відстані. Вектор описується списком *AS* по 16 біт на *AS*. *BGP* регулярно (кожні 30сек) посилає сусідам *TCP*-повідомлення, що підтверджують, що вузол живий (це не теж саме що "*Keepalive*" функція в *TCP*). Якщо два *BGP*-маршрутизатори спробують встановити зв'язок один з одним одночасно,

такі два зв’язки можуть бути встановлені. Така ситуація називається зіткненням, один зі зв’язків повинен бути ліквідований. При встановленні зв’язку маршрутизаторів спочатку робиться спроба реалізувати вищий із протоколів (наприклад, *BGP*-4), якщо один з них не підтримує цю версію, номер версії знижується.

Основним повідомленням протоколу *BGP* є повідомлення *UPDATE* (обновити), за допомогою якого маршрутизатор повідомляє маршрутизатор сусідньої автономної системи про досяжність мереж, яка належить до його власної автономної системи. Сама назва цього повідомлення говорить про те, що це тригерне оголошення, яке посилається сусідові тільки тоді, коли в автономній системі що-небудь різко міняється: з’являються нові мережі або нові шляхи до мереж, або ж навпроти, зникають існуючі мережі або шляхи. В одному повідомленні *UPDATE* можна оголосити про один новий маршрут або анулювати декілька, що перестали існувати. Під маршрутом в *BGP* розуміють послідовність автономних систем, яку потрібно пройти на шляху до зазначеної в адресі мережі.

Протокол *BGP*-4 є вдосконаленою версією (у порівнянні з *BGP*-3). Ця версія дозволяє пересилати інформацію про маршрут у рамках одного *IP*-пакета. Концепція класів мереж і підмереж існує поза рамками цієї версії. Для того щоб пристосуватися до цього, змінені семантика і кодування атрибута *AS_PASS*. Введено новий атрибут *LOCAL_PREF* (ступінь переваги маршруту для власної *AS*), який спрощує процедуру вибору маршруту. Введено нові атрибути *ATOMIC_AGGREGATE* і *AGGREGATOR*, які дозволяють групувати маршрути. Структура даних відбивається і на схемі ухвалення рішення, що має три фази:

1. Обчислення ступеня переваги для кожного маршруту, отриманого від сусідньої *AS*, і передача інформації іншим вузлам місцевої *AS*.
2. Вибір кращого маршруту з наявного числа для кожної точки призначення і укладання результату в *LOC-RIB*.
3. Розсилання інформації з *LOC-RIB* усім сусіднім *AS* згідно з політикою, закладеною в *RIB*. Угруповання маршрутів і редагування маршрутної інформації.

11.4 Протоколи транспортного рівня. Порти

Відправником і одержувачем даних, переданих через мережу, з погляду транспортного рівня є додаток (процес). Як будь-яка програма, процеси створюються і знищуються, на кожному вузлі може виконуватися кілька процесів, а кожний процес може мати кілька точок підключення до мережі. Такі логічні точки (що програмно організуються, як правило, у вигляді черг повідомлень) називаються **портами** (port). Номер порту однозначно ідентифікує процес. Коли вузол одержує дейтаграму транспортного рівня, він направляє її прикладному процесу, використовуючи номер порту, заданий при встановленні зв’язку.

Порти нумеруються позитивними цілими 16-бітовими числами. Різні протоколи транспортного рівня нумерують свої порти незалежно, тобто, наприклад, порт 223 протоколу TCP і порт 223 протоколу UDP зовсім не зв'язані один з одним.

Деякі номери портів задані стандартами. Ці номери виділяються організацією *IANA* (*Internet Assigned Numbers Authority*). У цей час під стандартні порти відведений діапазон від 0 до 1023 (раніше – до 255). Інші порти можуть вільно використовуватися прикладними процесами. Порти в діапазоні від 1024 до 5000 називаються тимчасовими (*ephemeral*). Призначення цих портів не стандартизовано, але *IANA* підтримує інформацію про їх використання.

Пара "порт – *IP*-адреса" називається (у термінології *TCP/IP*) **гніздом** або **сокетом** (***socket***) і однозначно вказує процес у мережі.

11.5 Протокол *UDP*

Протокол *UDP* (*User Datagram Protocol*, Протокол користувальницьких дейтаграм) описаний в *RFC* 768. Він надає прикладним процесам найпростіші послуги транспортного рівня. Дві основні функції *UDP* – розподіл дейтаграм між процесами (на підставі номерів портів) і контроль передачі користувальницьких даних (не тільки заголовка, як у протоколі *IP*). Як і *IP*, *UDP* не гарантує доставку і не підтримує установку з'єднань.

Повідомлення протоколу *UDP* називається **користувальницькою дейтаграмою** (*User datagram*) і складається із заголовка і користувальницьких даних. Заголовок складається із чотирьох 16-бітових полів:

1) порт відправника (може заповнюватися нулями, якщо не використовується);
2) порт одержувача;
3) довжина повідомлення (у байтах);
4) контрольна сума.

Відразу за заголовком ідуть користувальницькі дані.

Нульове значення в полі "Контрольна сума" означає, що контрольна сума не обчислювалася. Для розрахунку контрольної суми до початку дейтаграми приписується псевдозаголовок, який складається з п'яти полів:

1) *IP*-адреса відправника;
2) *IP*-адреса одержувача;
3) нулі (8 біт);
4) протокол (8 біт);
5) довжина дейтаграми (16 біт).

Крім того, до кінця дейтаграми, можливо, додають нульовий байт, щоб його довжина (разом із псевдозаголовком) була кратна 16 бітам. Потім обчислюється контрольна сума (як у протоколі *IP*), і псевдозаголовок відкидається.

11.6 Протокол *TCP*

Протокол *TCP* (*Transmission Control Protocol*, Протокол керування передачею) описаний в *RFC* 793. Він забезпечує надійну передачу потоку даних, використовуючи сервіс передачі дейтаграм протоколу *IP*. Пакети, передані протоколом *TCP*, називаються **сегментами**. Надійність передачі забезпечується за допомогою нумерації байтів потоку і підтверджень прийому. Всі байти вихідного потоку даних нумеруються (цей номер називається номером у послідовності (*sequence number*), і з кожним сегментом передається номер у послідовності його першого байта.

Оскільки два вузли можуть передавати два потоки дані за одним *TCP*-з'єднанням, то для передачі підтверджень одного потоку використовуються сегменти зустрічного потоку. У кожному сегменті передається номер байта у послідовності, який збирається прийняти даний вузол.

Після того як модуль *TCP* передасть сегмент модулю *IP*, він записує його копію в чергу на повторну передачу і запускає таймер для цього сегмента. Коли надійде підтвердження прийому сегмента (тобто буде прийнятий сегмент, у якому буде заявлено, що та сторона готова прийняти байт із номером, більшим за всі номери байтів сегмента, що чекає повторної передачі), сегмент видаляється із черги. Якщо підтвердження не надходить до спрацювання таймера, сегмент відправляється повторно. Сегмент складається із заголовка і поля даних.

Формат заголовка сегмента *TCP* наведений на рис. 11.1.

<table>
<tr><td colspan="3">Порт відправника</td><td>Порт одержувача</td></tr>
<tr><td colspan="4">Номер у послідовності</td></tr>
<tr><td colspan="4">Номер підтвердження</td></tr>
<tr><td>Зсув даних</td><td>Резерв</td><td>Біти керування</td><td>Вікно</td></tr>
<tr><td colspan="3">Контрольна сума</td><td>Покажчик терміновості</td></tr>
<tr><td colspan="2">Опції</td><td colspan="2">Вирівнювання</td></tr>
</table>

Рисунок 11.1 – Формат заголовка сегмента *TCP*

Формат заголовка сегмента *TCP* містить:

1) порт відправника (*Source port*) і порт одержувача (*Destination port*) [16 біт] – номери портів на вузлах;

2) номер у послідовності (*Sequence Number*) [32 біти] – номер у потоці першого байта даних цього сегмента. Якщо встановлено керуючий біт *SYN*, то в цьому полі утримується початковий номер у послідовності (*Initial Sequence Number, ISN*) і перший байт даних сегмента має номер у потоці *ISN*+1;

3) номер підтвердження (*Acknowledgement number*) [32 біти] – номер байта в потоці, очікуваного відправником даного сегмента. При цьому повинен бути встановлений керуючий біт *ACK*;

4. зсув даних (*Data offset*) [4 біти] – кількість 32-бітових слів у заголовку *TCP*-сегмента. Мінімальне значення поля – 5 (20-ти байтовий заголовок) ;

5. резерв (*Reserved*) [6 біт] – повинен бути заповнений нулями;

6. біти керування (*Control bits*) [6 біт] – від старшого до молодшого:

URG (*Urgent Pointer field significant*) – брати до уваги поле Покажчик терміновості;

ACK (*Acknowledgement field significant*) – брати до уваги поле Номер підтвердження;

PSH (*Push function*) – сегмент містить "проштовхнуті" дані;

RST (*Reset the connection*) – перервати зв'язок;

SYN (*Synchronize sequence numbers*) – синхронізувати номери байтів у потоці;

FIN (*No more data from sender*) – відправник більше не буде передавати дані;

7. вікно (*Window*) [16 біт] – розмір вікна – кількість байтів даних, починаючи із зазначеного в полі Номер підтвердження, які відправник даного сегмента готовий прийняти;

8. контрольна сума (*Checksum*) [16 біт] – контрольна сума всього сегмента (заголовка і даних), обчислюється за алгоритмом протоколу *IP*. Як і в *UDP*, перед обчисленням контрольної суми до сегмента приписується псевдозаголовок (тієї ж структури, що і в *UDP*) ;

9. покажчик терміновості (*Urgent Pointer*) [16 біт] – містить номер першого байта, що має звичайний статус терміновості. При цьому повинен бути встановлений керуючий біт *URG*;

10. опції (*Options*) [змінний розмір] – додаткова службова інформація. Подібно опції заголовка *IP*-дейтаграми мають змінну довжину і можуть бути взагалі відсутніми;

11. вирівнювання (*Padding*) – поле, що використовується для доведення розміру заголовка до цілого числа 32-бітових слів.

Щоразу при встановленні з'єднання модуль *TCP* створює структуру даних – Блок керування передачею (*Transmission Control Block, TCB*), що зберігає постійну інформацію про з'єднання (*IP*-адреси, номери портів, покажчики на вхідний і вихідний буфери, черга повторного відправлення і т.д.) і поточні значення змінних, що описують поточний стан з'єднання.

Контрольні запитання

1. Що таке алгоритм маршрутизації?
2. Яка класифікація алгоритмів маршрутизації?
3. Які метрики використовуються в алгоритмах маршрутизації?
4. У чому полягають відмінності алгоритмів *RIP* і *OSPF*?
5. Що таке сокет?
6. Які функції протоколів TCP і UDP?

12 ДІАГНОСТИКА КОМП'ЮТЕРНИХ МЕРЕЖ

Системи аналізу мережі належать до класу інструментальних програмних засобів для моніторингу мережного стану і виявлення деяких типів мережних проблем.

Однією з причин підвищеної уваги до систем аналізу мережі є існування великого кола комерційних і соціальних галузей, що користуються комп'ютерними технологіями. У цих галузях аналіз мережі, необхідний для безперебійної роботи, буде сприйнятий дуже успішно. Очікується, що застосування подібних систем істотно зменшить кількість часу, який тратиться на відновлення працездатності мережі, що істотно полегшить рутинну працю системного адміністратора.

Системи аналізу можуть використовуватися системним адміністратором та користувачем для перевірки і детального аналізу працездатності мереж.

При використанні мереж виникають питання найбільш повного використання їх ресурсів. Для настройки та підтримки функціонування апаратних засобів мережі необхідні діагностичні засоби.

Сучасні засоби тестування мереж

Більшість операційних систем мають у своєму складі програму *ping,* яка дозволяє виявити присутність вузла у мережі та з'ясувати час необхідний для проходження до нього пакета даних.

Програма *Ping* є зручним засобом перевірки працездатності вузла і використовується як первинний засіб діагностики. Але вона має недоліки, оскільки час вимірюється за допомогою таймера операційної системи, точність якого становить 1мс., що дає помилку ± 1мс. Окремі програмні рішення ping взагалі не відображують час, меньший за 10 мс. В той час, коли сучасні локальні мережі, навіть достатньо великі, мають час проходження пакета в межах декількох мікросекунд. Така точність не може задовольнити потреби діагностування.

Це ще один зручний мережний інструмент, що дозволяє визначати IP-адреси маршрутизаторів, які пройдені запитом на шляху до визначеного вузла. **Ping** також дозволяє визначити *IP*-адреса маршрутизаторів, використовуючи параметр запису маршруту в розширеному заголовку IP, однак його можливості обмежені усього дев'ятьма транзитами - максимальним простором, призначеним для адреси у розширеному заголовку. Транзит відбувається кожного разу, коли *IP*-дейтаграма проходить маршрутизатор для переходу в іншу фізичну мережу. Для маршрутів з більш ніж дев'ятма такими переходами використовується програма *Traceroute*.

В основі програми ***Traceroute*** лежить ідея відправлення *UDP*-пакета адресату і поступовій зміні часу життя (*time-to-live, TTL*). Спочатку *TTL*-пакета дорівнює 1, і коли пакет досягає першого маршрутизатора, його ТП.

скидається, і маршрутизатор генерує *ICMP*-пакет зі зведеннями, що перевищується ліміт часу. Тоді початкове значення *TTL* збільшують на 1, так що цього разу *UDP*-пакет досягає наступного маршрутизатора, той теж відсилає *ICMP*-пакет за перевищенням ліміту часу. Сукупність цих *ICMP*-повідомленнь дає список *IP*-адрес, що пройдені на шляху до кінцевого вузла. Коли *TTL* збільшиться настільки, що *UDP*-пакет досягне кінцевого вузла, повертається *ICMP*-повідомлення про недосяжність порту, оскільки на одержувача жоден процес не чекає вашого повідомлення.

Програма *Traceroute* корисна тим, що подає найдокладнішу інформацію про маршрут до конкретного вузла − це часто необхідно при багатоадресній передачі чи проблемах з маршрутизацією.

Програма ***NetStat*** – потужний інструмент, який дозволяє отримати статистичні дані про всі підключення, мережеві адаптери та протоколи.

Програма здатна відображати статистику для протоколів *IP, TCP* та *UDP*. Для кожного протоколу відображається його власна специфічна статистика.

Для *IP* статистика повертає:

- Отримано пакетів
- Отримано помилок у заголовках
- Отримано помилок в адресах
- Спрямовано датаграм
- Отримано невідомих протоколів
- Відкинуто отриманих пакетів
- Діставлено отриманих пакетів
- Запитів на висновок
- Відкинуто маршрутів
- Відкинуто вихідних пакетів
- Вихідних пакетів без маршруту
- Потрібна зборка
- Успішна зборка
- Перебоїв при зборці
- Успішно фрагментовано датаграм
- Перебоїв при фрагментації датаграм
- Створено фрагментів

Для *TCP*:

- Підключень активно відкрито
- Підключень пасивно відкрито
- Перебоїв при підключенні
- Скинуто підключень
- Поточних підключень
- Отримано сегментів
- Відправлено сегментів

- Перевідправлено сегментів

Для *UDP*:

- Отримано дейтаграм
- Відсутність портів
- Помилки при одержанні
- Відправлено датаграм

Програма *NetStat*, яка входить до стандартного комплекту поставки ОС *Windows9x*, не відображає статистики для протоколу *ICMP*.

Також програма відображує з`єднання для протоколів *TCP* та *UDP*.

Контрольні запитання

1. Ким можуть використовуватися системи аналізу мережі?
2. Яке основне призначення програми *Ping?*
3. Які основні недоліки прорами *Ping?*
4. Яке основне призначення програми *Traceroute?*
5. Я ке основне призначення програми *NetStat?*

13 ПРОЄКТУВАННЯ ДІАГНОСТИЧНОЇ ПРОГРАМИ З ВИКОРИСТАННЯМ ПРОТОКОЛУ *TCP/IP*

Протокол *TCP* надає транспортні послуги, які відрізняються від послуг *UDP*. Замість ненадійної доставки дейтаграм без встановлення з'єднань він забезпечує гарантовану доставку з установленням з'єднань у вигляді байтових потоків.

Протокол *TCP* використовується в тих випадках, коли потрібна надійна доставка повідомлень. Він звільняє прикладні процеси від необхідності використовувати тайм-аути і повторні передачі для забезпечення надійності. Найбільш типовими прикладними процесами, які використовують *TCP*, є *FTP* (*File Transfer Protocol* – протокол передачі файлів) і *TELNET*. Крім того, *TCP* використовує система *X-Window, rcp* (*remote copy* - віддалене копіювання) та інші "*r*-команди". Розширені можливості TCP надаються не задарма. Реалізація *TCP* вимагає великої продуктивності процесора і великої пропускної здатності мережі. Внутрішня структура модуля *TCP* набагато складніша структури модуля *UDP*.

Коли прикладний процес починає використовувати *TCP*, то модуль *TCP* на машині клієнта і модуль *TCP* на машині серверу починають спілкуватися. Ці два кінцевих модулі *TCP* підтримують інформацію про стан з'єднання, який називається віртуальним каналом. Цей віртуальний канал споживає ресурси обох кінцевих модулів *TCP*. Канал є дуплексним; дані можуть одночасно передаватися в обох напрямках. Один прикладний процес пише дані в *TCP*-порт, вони проходять мережею, і інший прикладний процес читає їх зі свого *TCP*-порту.

Протокол *TCP* розбиває потік байтів на пакети; він не зберігає меж між записами. Наприклад, якщо один прикладний процес здійснює 5 записів у *TCP*-порт, то прикладний процес на іншому кінці віртуального каналу може виконати 10 читань для того, щоб одержати всі дані. Але цей же процес може одержати всі дані відразу, зробивши тільки одну операцію читання. Не існує залежності між числом і розміром записуваних повідомлень, з одного боку і числом і розміром зчитувальних повідомлень – з іншого.

Протокол *TCP* вимагає, щоб усі відправлені дані були підтверджені прийнятою стороною. Він використовує тайм-аути і повторні передачі для забезпечення надійної доставки. Відправникові дозволяється передавати деяку кількість даних, не чекаючи підтвердження приймання у раніше відправлених даних. Таким чином, між відправленими і підтвердженими даними існує вікно вже відправлених, але ще непідтверджених даних. Кількість байтів, які можна передавати без підтвердження, називається розміром вікна. Як правило, розмір вікна встановлюється в стартових файлах мережного програмного забезпечення. Оскільки *TCP*-канал є дуплексним, то підтвердження для даних, що йдуть в одному напрямку, можуть передаватися разом з даними, що йдуть у протилежному напрямку.

Приймачі на обох сторонах віртуального каналу виконують керування потоком переданих даних для того, щоб не допускати переповнення буферів.

Тестування досяжності призначення і його станів.

Протоколи *TCP/IP* забезпечують засіб, який допомагає мережним адміністраторам або користувачам ідентифікувати проблеми в мережі. Найчастіше використовується засіб налагодження викликів *ICMP* повідомлень – запит відгуку і відповідь відгуку. Сервер або шлюз посилає повідомлення запиту відгуку, указаного місцем призначення. Будь-яка машина, яка одержала запит відгуку, генерує відповідь відгуку і повертає його первісному відправникові. Цей запит містить необов'язкову область даних; відповідь містить копію даних, посланих у запиті. Запит відгуку і пов'язана з ним відповідь можуть бути використаними для перевірки досяжності призначення і здатності відповідати на запити. Запит відгуку і відповідь на нього передаються в *IP*-дейтаграмах. Успішне приймання відповіді свідчить про працездатність основних частин транспортної системи. По-перше, програмне забезпечення *IP* на машині джерела зробило маршрутизацію дейтаграми. По-друге, проміжні шлюзи між джерелом і одержувачем коректно маршрутизують дейтаграми. По-третє, машина одержувача працює (принаймні, вона обробляє переривання) і програмне забезпечення як *IP*, так і *ICMP* виконує свої функції. І нарешті, таблиці маршрутів у шлюзах назад по всій дорозі коректні.

У багатьох системах команда, яку користувачі викликають для посилки запиту відгуку *ICMP*, називається *ping*. Ускладнені версії цієї програми посилають серії запитів відгуку *ICMP*, приймають відповіді і видають статистику про втрати дейтаграм. Вони дозволяють користувачеві вказувати довжину даних, які посилають, і інтервали часу між запитами. Менш складні версії просто надсилають запит відгуку *ICMP* і чекають відповіді.

Хоча *IP* є механізмом ненадійної доставки, дейтаграми не знищуються просто так. Щораз, коли помилка заважає шлюзу зробити маршрутизацію або доставку дейтаграми, шлюз посилає повідомлення про недосяжність призначення назад його джерелу, а потім знищує дейтаграму. Помилки недосяжності мережі звичайно є наслідком помилок маршрутизації; помилки недосяжності серверу – наслідок помилок при доставці.

Призначення можуть бути недосяжними через те, що устаткування було тимчасово непрацездатне, або через те, що відправник указав неіснуючу адресу призначення, або (у рідких випадках) через те, що в шлюзі не зазначено маршруту до мережі призначення. Відзначимо, що хоча шлюзи повідомляють про виявлені помилки, вони можуть не знати про всі помилки доставки. Наприклад, якщо машина одержувача приєднана до мережі *Ethernet*, то мережне устаткування не надає підтвердження одержань дейтаграми. Тому шлюз може продовжувати

надсилати пакети призначенню навіть після того, як воно відключено, не одержуючи при цьому ніякої інформації про те, що пакети не доставляються.

13.1 Організація функціонування системи

Програма розбивається на 2 частині: клієнтську і серверну, кожна з них виконує свою функцію.

Клієнтська частина працює таким чином:

1) при запуску програми виконується функція *FormCreate(Sender: TObject)* – ініціалізує сокет;

2) функція *Button1Click(Sender: TObject)* – відкриває ініціалізований сокет;

3) при одержанні даних спрацьовує функція *TCPClient1Data(Sender: TObject; Socket: TSocket)* – виводить інформацію від серверу;

4) при закритті програми виконується функція *FormDestroy(Sender: TObject)* – у ній сокет знищується.

Серверна частина програми знаходиться в режимі очікування повідомлення від клієнта і при його одержанні відсилає необхідну інформацію назад.

Більш докладно це можна описати так:

- при запуску програми виконується функція *FormCreate(Sender: TObject)* – вона ініціалізує сокет на приймання даних;
- виконання функції *Button1Click(Sender: TObject)* задає сокету порт для прослуховування;
- функція *Button2Click(Sender: TObject)* – закриває порт;
- при спробі зовнішнього приєднання до даного порту спрацьовує функція *TCPServer1Accept(Sender: TObject; Socket: TSocket)*. У цій функції відбувається збір інформації для клієнта і відправка;
- при закритті програми виконується функція знищення сокету *FormDestroy(Sender: TObject)*.

13.1.1 Опис функцій та дій серверної частини програми.

1. ***TTY(Msg: string) –*** функція відображення даних від клієнта.

```
procedure TForm1.TTY(Msg: string);
begin
  with Memo1.Lines do
    begin
if Count > 100 then
       Delete(0);
     Add(Msg);
    end;
end;
```

2. ***TCPClient1Error(Sender: TObject; Error: integer; Msg: string)*** – функція оброблювача помилки.

```
procedure TForm1.TCPClient1Error(Sender: TObject; Error: integer;
Msg: string);
begin
  MessageDlg(Msg, mtError, [mbOK], 0);
end;
```

3. ***TCPClient1Data(Sender: TObject; Socket: TSocket)*** – подія на приймання даних.

```
procedure TForm1.TCPClient1Data(Sender: TObject; Socket: TSocket);
begin
  TTY(TCPClient1.Read);
end;
```

4. ***TCPClient1Connect(Sender: TObject; Socket: TSocket)*** – подія на підключення до клієнта.

```
procedure TForm1.TCPClient1Connect(Sender: TObject; Socket:
TSocket);
begin
  Memo1.Lines.Clear;
  with TCPClient1 do
    begin
      StatusBar1.SimpleText:= 'Connected on local port
'+SocketToPort(LocalSocket);
    end;
end;
```

5. ***FormCreate(Sender: TObject);*** – функція створення форми і ініціалізація сокету.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   TCPClient1:= TTCPClient.Create(Self);//створюємо об'єкт
  with TCPClient1 do
    begin
      OnError:= TCPClient1Error;//встановлення оброблювачів подій
      OnData:= TCPClient1Data;
 OnConnect:= TCPClient1Connect;
 OnClose:= TCPClient1Close;
      TTY('WinSocket Version: '+Version);//виводимо на екран
//інформацію про версії бібліотек, адресу, ім'я хоста
      TTY('Description: '+Description);
      TTY('SystemStatus: '+SystemStatus);
      TTY(IntToStr(MaxUDPSize));
      TTY('');
      TTY('Local Host Name: '+LocalHostName);
      TTY('Local Host Address: '+LocalHostAddress);
```

```
TTY(‘‘);    end; end;
```

6. ***FormDestroy(Sender: TObject);*** – закриття додатка.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
TCPClient1.Free;
end;
```

7. ***Button1Click(Sender: TObject);*** – відкриття сокету на відсилання даних.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
with TCPClient1 do
   begin
     Host:= Edit1.Text; //призначаємо порт і адресу для відсилання
     Port:= Edit2.Text;
     StatusBar1.SimpleText:= ‘Trying to connect to ‘+Host+’ at port
‘+Port;
     Try      Open;     Except     end;
     if SocketState = ssOpen then
      begin
        Edit1.Enabled:= false;
        Edit2.Enabled:= false;
        Button1.Enabled:= false;
        Button2.Enabled:= true;
      end;    end; end;
```

8. ***Button2Click(Sender: TObject);*** – закриття з’єднання.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  TCPClient1.Close;
  Edit1.Enabled:= true;
  Edit2.Enabled:= true;
  Button1.Enabled:= true;
  Button2.Enabled:= false;
  Memo1.Lines.Clear;
  StatusBar1.SimpleText:= ‘‘;
end;
```

9. ***TCPClient1Close(Sender: TObject; Socket: TSocket)*** – закриття сокету.

```
procedure TForm1.TCPClient1Close(Sender: TObject; Socket: TSocket);
begin
  TCPClient1.Close;
  Edit1.Enabled:= true;
  Edit2.Enabled:= true;
  Button1.Enabled:= true;
  Button2.Enabled:= false;
  StatusBar1.SimpleText:= ‘Connection closed by server!’;
```

```
  TTY('Connection closed by server!');
end;
```

13.2.1 Опис функцій та дій клієнтської частини програми.

1. ***TTY(Msg: string)*** – функція відображення даних від серверу.

```
procedure TForm1.TTY(Msg: string);
begin
  with Memo1.Lines do
    begin
      if Count > 100 then
        Delete(0);
      Add(Msg);
    end;
end;
```

2. ***TCPServer1Accept(Sender: TObject; Socket: TSocket);*** – функція оброблювач події коли до серверу хтось підключився.

```
procedure TForm1.TCPServer1Accept(Sender: TObject; Socket:
TSocket);
  var
    tmp : array [0..100] of char;
    lpMemoryStatus : TMemoryStatus;
begin
  with TCPServer1 do
    begin
      if Clients.Count > StrToInt(Edit3.Text) then
        begin
          Write(Socket, 'Sorry! TTCPServer Demo reached max client
limit...');
          Disconnect(Socket);
        end
      else
        begin //одержуємо дані про ресурси пам'яті комп'ютера
          lpMemoryStatus.dwLength := SizeOf(lpMemoryStatus);
          GlobalMemoryStatus(lpMemoryStatus);
          with lpMemoryStatus do begin
//відсилаємо отриману інформацію клієнтові
  Write('Вільно пам'яті:' + IntToStr(dwMemoryLoad) + '%');
  Write('ОЗУ:'+Format('%0.0f Мбайт',[dwTotalPhys div 1024 /1024]));
  Write('ОЗУ вільно:'+ Format('%0.3f Мбайт',[dwAvailPhys div 1024 /
1024]));
  Write('Файл підкачування всього:'+Format('%0.0f
Мбайт',[dwTotalPageFile div 1024 /1024]));
  Write('Файл підкачування свобдно:'+ Format('%0.0f Мбайт',
[dwAvailPageFile div 1024 / 1024]));
  end;
```

```
 StatusBar1.SimpleText:= IntToStr(Clients.Count)+' Client(s) connected';
      end;    end; end;
```

3. ***FormCreate(Sender: TObject);*** – створення форми і ініціалізація сокету на приймання даних.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
TCPServer1:= TTCPServer.Create(Self);
  with TCPServer1 do
    begin
 //установка оброблювачів подій
          OnAccept:= TCPServer1Accept;
      OnClose:= TCPServer1Close;
      Port := '1';
      Open;
    end;
  Button1Click(Button1);
end;
```

4. ***FormDestroy(Sender: TObject);*** – закриття додатка, видалення сокетів.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
TCPServer1.Free;
end;
```

5. ***Button1Click(Sender: TObject);*** – функція запуску сокету на прослуховування порту.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
with TCPServer1 do
   begin
//задаємо порт
     Port:= Edit1.Text;
//відкриваємо сокет
     Open;
     if SocketState = ssListening then
       begin
         StatusBar1.SimpleText:= LocalHostAddress+' Listening on
port: '+Port;
         Edit1.Enabled:= false;
         Button1.Enabled:= false;
         Button2.Enabled:= true;
       end;    end; end;
```

6. ***Button2Click(Sender: TObject);*** – зупиняємо прослуховування порту.

```
procedure TForm1.Button2Click(Sender: TObject);
```

```
begin
  TCPServer1.Close;
  Edit1.Enabled:= true;
  Button1.Enabled:= true;
  Button2.Enabled:= false;
  StatusBar1.SimpleText:= ‘‘;
end;
```

13.2.2 Опис допоміжних функцій та їх дій в програмі.

1. ***TTCPClient.Create(AOwner: TComponent);*** – конструктор клієнтського об’єкта.

```
constructor TTCPClient.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FHandle:= AllocateHWnd(WndProc);
  FProtocol:= IPPROTO_TCP; //протокол
  FType:= SOCK_STREAM; //тип пакетів
end;
```

2. ***TTCPClient.Destroy;*** – деструктор.

```
destructor TTCPClient.Destroy;
begin
  Close;
  DeallocateHWnd(FHandle); //знищуємо покажчик на об’єкт
  inherited Destroy; end;
```

3. ***TTCPClient.Open;*** – відкриття клієнтського сокету.

```
procedure TTCPClient.Open;
var
  SockAddrIn: TSockAddrIn;
  SockOpt: LongBool;
  EventMask: longint;
begin
  if (FSocketState <> ssClosed) then
    Exit;
//заповнюємо структру необхідну для відправлення даних
  if not GetSockAddrIn(FHost, FPort, SockAddrIn) then
    Exit;
//створення сокету
  FLocalSocket:= socket(PF_INET, FType, 0);
  if FLocalSocket = INVALID_SOCKET then
    begin
      SocketError(WSAGetLastError);
      Exit;
    end;

  EventMask:= (FD_CONNECT or FD_READ or FD_CLOSE);
```

```
  if WSAASyncSelect(FLocalSocket, FHandle, WM_ASYNCSELECT,
EventMask) <> 0 then
    begin
     SocketError(WSAGetLastError);
     closesocket(FLocalSocket);
     Exit;
    end;
//установка опцій для сокету
SockOpt:= true; {Enable OOB Data inline}
  if setsockopt(FLocalSocket, SOL_SOCKET, SO_OOBINLINE,
PChar(@SockOpt), SizeOf(SockOpt)) <> 0 then
    begin
     SocketError(WSAGetLastError);
     closesocket(FLocalSocket);
     Exit;
    end;
//прив'язка локального сокету до структури даних
  if connect(FLocalSocket, SockAddrIn, SizeOf(SockAddrIn)) <> 0 then
    begin
     if WSAGetLastError <> WSAEWOULDBLOCK then
      begin
       SocketError(WSAGetLastError);
       closesocket(FLocalSocket);
       Exit;
      end;
    end;
 FSocketState:= ssOpen;
end;
```

4. ***TTCPClient.Close;*** – закриття сокету.

```
procedure TTCPClient.Close;
begin
  if (FSocketState = ssNotStarted) or (FSocketState = ssClosed) then
    Exit;
  SocketClose(FLocalSocket, FHandle);
  if FLocalSocket = INVALID_SOCKET then
    FSocketState:= ssClosed;
end;
```

5. ***TTCPClient.Write(Data: string);*** – запис даних у сокет.

```
procedure TTCPClient.Write(Data: string);
begin
//запис у сокет
SocketWrite(FLocalSocket, 0, Data);
end;
```

6. ***TTCPClient.Read: string;*** – читання даних із сокету.

```
function TTCPClient.Read: string;
begin
//читання із сокету
 Result:= SocketRead(FLocalSocket, 0);
end;
```

7. ***TTCPServer.Open;*** – відкриття серверного сокету.

```
procedure TTCPServer.Open;
var
  SockAddrIn: TSockAddrIn;
  SockOpt: LongBool;
begin
  if (FSocketState <> ssClosed) then
    Exit;
//заповнюємо структуру даних
  if not GetAnySockAddrIn(FPort, SockAddrIn) then
    Exit;

//створення сокету
FLocalSocket:= socket(PF_INET, FType, 0);
  if FLocalSocket = INVALID_SOCKET then
    begin
      SocketError(WSAGetLastError);
      Exit;
    end;

  if WSAASyncSelect(FLocalSocket, FHandle, WM_ASYNCSELECT,
FD_ACCEPT) <> 0 then
      begin
        SocketError(WSAGetLastError);
        closesocket(FLocalSocket);
        Exit;
      end;
//прив'язка сокету і структури
if bind(FLocalSocket, SockAddrIn, SizeOf(SockAddrIn)) <> 0 then
      begin
        SocketError(WSAGetLastError);
        closesocket(FLocalSocket);
        Exit;
      end;
if listen(FLocalSocket, 5) <> 0 then
      begin
        SocketError(WSAGetLastError);
        closesocket(FLocalSocket);
```

```
      Exit;
    end;
  FSocketState:= ssListening;
end;
```

8. ***SocketWrite(Socket: TSocket; Flag: integer; Data: string);*** – функція запису даних у сокет.

```
procedure SocketWrite(Socket: TSocket; Flag: integer; Data: string);
var
  TotSent, ToSend, Sent, ErrorLoop: integer;
begin
  if Data <> '' then
    begin
      ErrorLoop:= 0;
      TotSent:= 0;
      ToSend:= Length(Data);
      repeat
//відсилаємо дані
    Sent:= send(Socket, Data[TotSent+1], (ToSend-TotSent), Flag);
        if Sent = SOCKET_ERROR then //якщо помилка
          begin
            Inc(ErrorLoop);
            if WSAGetLastError <> WSAEWOULDBLOCK then
              begin
                SocketError(WSAGetLastError);
                Exit;
              end;
          end
        else
          Inc(TotSent, Sent);
      until (TotSent >= ToSend) or (ErrorLoop > MAX_LOOP);
    end;
end;
```

9. ***SocketRead(Socket: TSocket; Flag: integer): string;*** – читання даних із сокету.

```
function SocketRead(Socket: TSocket; Flag: integer): string;
var
  Received: longint;
begin
  Result:= '';
//читання даних із сокету
  Received:= recv(Socket, FReadBuffer, SizeOf(TReadBuffer), Flag);
  if Received = SOCKET_ERROR then //якщо помилка
```

```
    begin
      if WSAGetLastError <> WSAEWOULDBLOCK then
        SocketError(WSAGetLastError);
    end
  else
    begin
      SetLength(Result, Received); //установка розміру масиву
      Move(FReadBuffer, Result[1], Received);
    end;
end;
```

10. ***SockAddrInToPort(SockAddrIn: TSockAddrIn): string;*** – одержання порту зі структури сокету.

```
function SockAddrInToPort(SockAddrIn: TSockAddrIn): string;
begin
Result:= IntToStr(ntohs(SockAddrIn.sin_port));
end;
```

11. ***SockAddrInToAddress(SockAddrIn: TSockAddrIn): string;*** – одержання адреси зі структури сокету.

```
function TCustomWSocket.SockAddrInToAddress(SockAddrIn:
TSockAddrIn): string;
begin
Result:= inet_ntoa(SockAddrIn.sin_addr);
end;
```

13.2 Результати тестування

При запуску системи першим вмикаємо програму-сервер. Отриману форму заповнюємо *IP*-адресою ПК, даємо номер порту та вмикаємо „Соединиться”.

У формі отримаємо повідомлення про версію *WinSock* (рис. 13.1).

При запуску клієнтської частини програми з’являється форма із трьома елементами керування (рис. 13.2):

- поле для введення порту для прослуховування;
- дві кнопки що включають / виключають прослуховування порту ПК, отримавши запит на передачу, формують дані для передачі:
- незайнята частина пам’яті, %;
- загальний осяг ОЗП ;
- незайнята частина ОЗП ;
- осяг файлу підкачки;
- незайнята частина файлу підкачки.

Сформувавши файл даних, ПК передає їх. ПК-клієнт отримує дані і виводить їх на екран. Це і є перевірка каналу зв’язку двох ПК з передачею

важливих даних про характеристики пам’яті ПК.

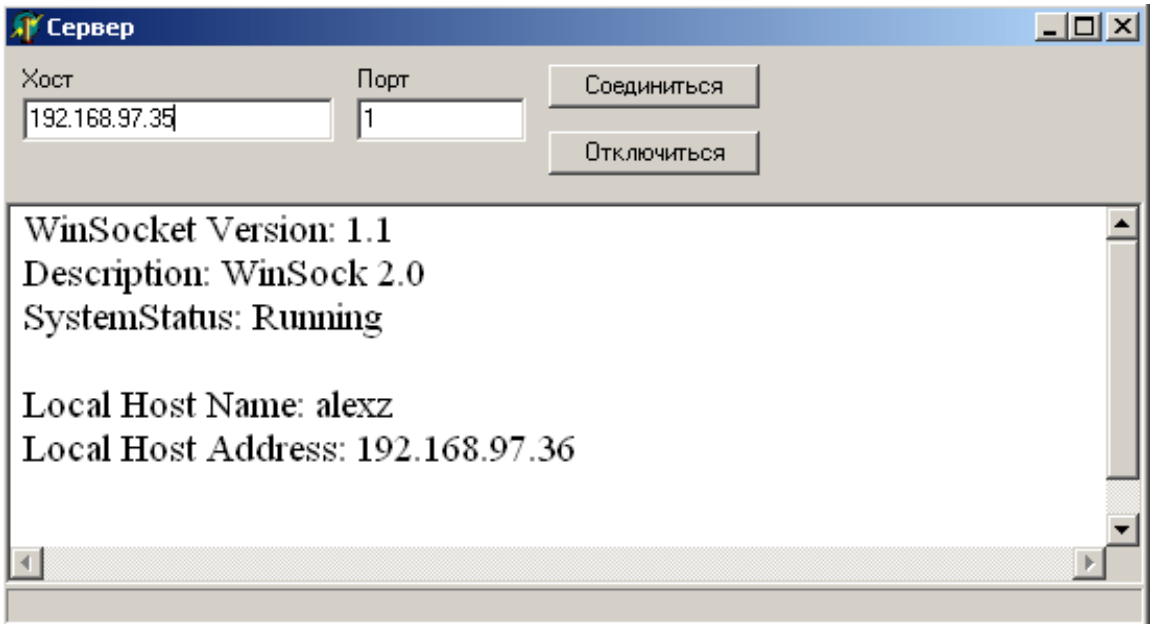

Рисунок 13.1 – Робота серверу

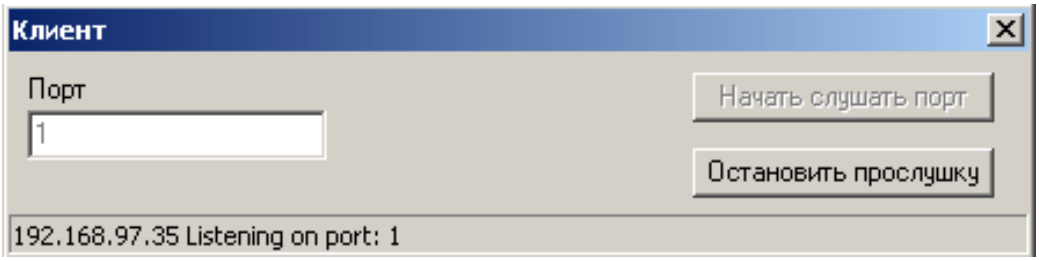

Рисунок 13.2 – Робота клієнта

Результати роботи системи програм подані на рис. 13.3.

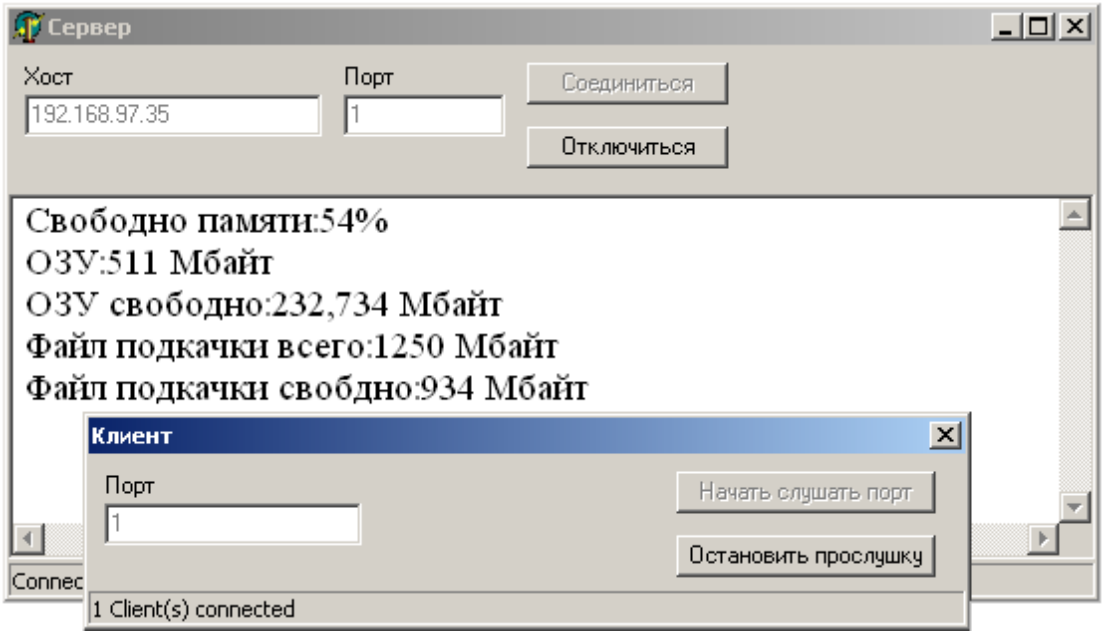

Рисунок 13.3 – Результати роботи системи

Контрольні запитання

1. Які послуги надає протокол *TCP* ?
2. В яких випадках використовується протокол *TCP?*
3. Що таке віртуальний канал?
4. Яким є віртуальний канал?
5. Чи вимагає протокол *TCP*, щоб всі відправлені дані були підтверджені прийнятою стороною?
6. Що використовує протокол *TCP* для забезпечення надійної доставки?
7. Як називається кількість байтів, які можна передавати без підтвердження?
8. Який засіб налагодження викликів ICMP найчастіше використовується?
9. Що таке *ping*?

14 ПРОЄКТУВАННЯ ЗАСОБІВ ДІАГНОСТИКИ З БІБЛІОТЕКОЮ *ICS*

14.1 Особливості протоколу HTTP

HTTP (від англ. *Hyper Text Transfer Protocol* – «протокол передачі гіпертексту») – мережний протокол прикладного рівня для передачі файлів. У стеку *TCP/IP* для *HTTP* зарезервовані порти 80 і 8080 транспортних протоколів *TCP* і *UDP* (на практиці використовується тільки перший).

Основним призначенням *HTTP* є передача веб-сторінок (текстових файлів з розміткою *HTML*), хоча за допомогою його з успіхом передаються і інші файли, які пов'язані і не пов'язані з веб-сторінками (зображення і додатки).

Протокол *HTTP* забезпечує передачу з віддалених серверів на локальний комп'ютер документів, які мають у собі код розмітки гіпертексту, написаного на мові *HTML* або *XML*. Даний протокол орієнтований поперед усього на падання інформації програмам перегляду веб-сторінок, веб-браузерам, найбільш відомими з яких є такі програми, як *Microsoft Internet Explorer* та Opera.

Тільки з використанням протоколу *HTTP* здійснюється відправка запитів віддаленим *HTTP*-серверам мережі Інтернет та обробка їх відгуків; окрім цього, *HTTP* дозволяє використовувати для виклику ресурсів Всесвітньої мережі адрес стандарту доменної системи імен (*DNS, Domain Name System*), тобто позначення, які називаються *URL* (*Uniform Resource Locator*) виду *http://www.domain.zone/page.htm(.html)*.

Кожен *HTTP* запит/відповідь складається із трьох частин:

1. стартовий рядок;
2. заголовки;
3. тіло повідомлення, що містить дані запиту, запитуваний ресурс або опис проблеми, якщо запит не був виконаний.

Стартові рядки розрізняються для запиту і відповіді. Рядок запиту виглядає так:

‹метод› ‹*URL*› *HTTP*/‹версія›,

де ‹метод› може бути:

- *OPTIONS* – повертає методи *HTTP*, які підтримуються сервером. Цей метод може служити для визначення можливостей веб-серверу.
- *GET* – запитує вміст зазначеного ресурсу. Запитуваний ресурс може приймати параметри (наприклад, розшукуваний рядок). Вони передаються в рядку *URL* (наприклад: *http://www.example.net/resource?param1=value1&param2=value2*). Відповідно до стандарту *HTTP* запити типу *GET* вважаються ідемпотентними, тобто багаторазове

повторення того самого запиту *GET* повинно приводити до однакових результатів (за умови, що сам ресурс не змінився за час між запитами). Це дозволяє кешувати відповіді на запити *GET*.

- *HEAD* – аналогічний методу *GET*, за винятком того, що у відповіді серверу відсутнє тіло. Це корисно для добування позначки-інформації, заданої в заголовках відповіді, без пересилання всього вмісту;
- *POST* – передає дані користувача (наприклад, з *HTML*-форми) заданому ресурсу. Наприклад, у блогах відвідувачі звичайно можуть вводити свої коментарі до записів в *HTML*–форму, після чого вони передаються серверу методом *POST* і він поміщає їх на сторінку. При цьому передані дані (у прикладі із блогами – це текст коментаря) включаються в тіло запиту. На відміну від методу *GET*, метод *POST* не вважається ідемпотентним, тобто багаторазове повторення тих самих запитів *POST* може повертати різні результати (наприклад, після кожного відправлення коментарю буде з'являтися одна копія цього коментарю);
- *PUT* – завантажує зазначений ресурс на сервер;
- *DELETE* – видаляє зазначений ресурс;
- *TRACE* – повертає отриманий запит так, що клієнт може побачити, що проміжні сервери додають або змінюють у запиті;
- *CONNECT* – використовується разом із проксі-серверами, які можуть динамічно перемикатися в тунельний режим *SSL*.

В основному використаються методи *GET* і *POST*.

Перший рядок відповіді виглядає так:

HTTP/‹версія› ‹код статусу› ‹опис статусу›

Найбільш типові статуси:

* 200 *OK* – запит виконаний успішно;
* 403 *Forbidden* – доступ до запитаного ресурсу заборонений;
* 404 *Not Found* – запитаний ресурс не знайдений.

14.2 Узагальнений алгоритм клієнт-серверної програми

Розроблена програма є клієнт-серверною, тому нижче буде наведено два алгоритми роботи відповідно для клієнта і для серверу.

Схема алгоритму клієнта:

1. Виведення на екран вікна програми.
2. Очищення вікна логу.
3. Визначення частоти процесора двома способами для подальшого використання при розрахунках.
4. Виведення повідомлення про необхідність запустити сервер перед початком роботи клієнту.
5. Перед початком під'єднання до серверу заблокувати всі кнопки.
6. Ініціалізація бібліотеки сокетів.

7. Створення сокету.
8. Заповнення адреси сокету.
9. Початок вимірювання часу двома способами.
10. Посилання даних.
11. Отримання повідомлення назад.
12. Закінчення вимірювання часу.
13. Повторення у циклі 3 рази пунктів 6 – 12.
14. Розрахунок середнього часу посилання-прийманняу пакетів.
15. Виведення результатів, розрахованих двома способами, у лог.
16. Розблокування кнопок.

Схема алгоритму серверу:

1. Виведення на екран вікна програми.
2. Очищення вікна логу.
3. Виведення повідомлення, що сервер запущено.
4. Заблокування всіх кнопок до кінця роботи з клієнтом.
5. Ініціалізація бібліотеки сокетів.
6. Створення сокету.
7. Заповнення адреси сокету.
8. Очікування прийманняу даних.
9. Приймання даних.
10. Посилання даних у зворотному напрямку.
11. Повторення у циклі 3 рази пунктів 8 – 10.
12. Розблокування кнопок та виведення повідомлення у лог, що сервер закінчив роботу.

14.3 Проектування алгоритму програми – клієнт з компонентами бібліотеки ICS

Алгоритм складається з узагальнених дій алгоритму та необхідних компонентів бібліотеки *ICS*, які дозволяють реалізувати систему тестування.

Алгоритм роботи клієнта:

1. Завантажуємо та виводимо на екран вікна програми.
2. Виводимо повідомлення про початок роботи.
3. Визначаємо частоту лічильника тактів.
   QueryPerformanceFrequency(Freq);
4. Після натиснення кнопки *GetButton* – блокуємо кнопки.
   GetButton.Enabled := FALSE;
   AbortButton.Enabled := TRUE;
5. Виводимо повідомлення про те, що було розпочато спробу старту клієнта.
   InfoLabel.Caption := 'Loading';

6. Ініціалізуємо бібліотеку сокетів (версія 1.01).

WSAStartup($101, Data)

7. Виконуємо перевірку, чи не виникло помилок при ініціалізації бібліотеки.

if (WSAStartup($101, Data) = 0) then

8. Виводимо у лог повідомлення про успішну ініціалізацію бібліотеки або про помилку.

WriteLogMessage('Sockets library initializated successfully.');
WriteLogMessage('[Error] Sockets library initializattion error!');

9. Виконуємо перевірку, чи не виникло помилок при створенні сокету.

if (S <> Invalid_Socket) then

10. Виводимо у лог повідомлення про успішне створення сокету або про помилку.

WriteLogMessage('Socket created successfully.');
WriteLogMessage('[Error] Can not create socket!');

11. Читаємо адресу серверу та символ для передачі з форми програми.

HttpCli1.URL := URLEdit.Text;

12. Робимо спробу підключитися до серверу.

13. Виконуємо перевірку, чи не виникло помилок при підключенні до серверу.

on E: EHttpException do begin

14. Виводимо у лог повідомлення про успішне підключення до серверу або про помилку.

InfoLabel.Caption := 'Received ' +
IntToStr(HttpCli1.RcvdStream.Size) + ' bytes';

15. Запам'ятовуємо у змінній стартове значення часу.

```
asm
  dw 310Fh
  mov TimerLo, eax
end;
QueryPerformanceCounter(tm1);
```

16. Передаємо символ на сервер (S – сокет, Buf – дані для передачі).

HttpCli1.Get;

17. Виконуємо перевірку, чи не виникло помилок при передачі символу на сервер.

if (NumBytesSend = SOCKET_ERROR) then

18. Виводимо у лог повідомлення про успішну передачу.

InfoLabel.Caption := 'Received ' +

IntToStr(HttpCli1.RcvdStream.Size) + ‘ bytes’;

19. Запам’ятовуємо у змінній кінцеве значення часу.

```
QueryPerformanceCounter(tm2);
asm
  dw 310Fh
  sub eax, TimerLo
  mov TimerLo, eax
end;
```

20. Розраховуємо затримку приймання-передачі у секундах та записуємо значення до масиву.

```
Tmp:= (TimerLo/1000000);
rlt:=((tm2-tm1)/hfreq)*1000;
```

21. Закриваємо бібліотеку сокетів.

WSACleanup;

22. Виконуємо перевірку, чи не виникло помилок при закритті бібліотеки.

if (WSACleanup = 0) then

23. Виводимо повідомлення у лог про успішне закриття бібліотеки або повідомлення про помилку.

```
WriteLogMessage(‘Client stoped.’);
WriteLogMessage(‘Can not close socket library.’);
```

24. Виводимо два розрахованих значення у лог.

```
Memo1.Lines.Add(‘Затраченное время:  ‘ + FloatToStr(rlt) + ‘ мс’);
Memo1.Lines.Add(‘Затраченное время (asm): ‘ + FloatToStr(Tmp) + ‘ мс’);
```

25. Розблокуємо кнопки форми.

14.4 Проектування алгоритму програми – сервер з компонентами бібліотеки *ICS*

Алгоритм складається з дій узагальненого алгоритму та необхідних компонент бібліотеки *ICS*, які дозволяють реалізувати систему тестування.

Алгоритм роботи серверу:

1. Завантажуємо та виводимо на екран вікна програми.
2. Виводимо повідомлення про готовність серверу.
3. Після натиснення кнопки *Start* – блокуємо всі кнопки.
4. Виводимо повідомлення про те, що було розпочато спробу старту серверу.
5. Ініціалізуємо бібліотеку сокетів (версія 1.01).

WSAStartup($101, Data);

6. Виконуємо перевірку, чи не виникло помилок при ініціалізації бібліотеки.

if (WSAStartup($101, Data) = 0) then

7. Виводимо у лог повідомлення про успішну ініціалізацію бібліотеки або про помилку.

WriteLogMessage('Sockets library initializated successfully.');

WriteLogMessage('[Error] Sockets library initializattion error!');

8. Виконуємо перевірку, чи не виникло помилок при створенні сокету.

if (S <> Invalid_Socket) then

9. Виводимо у лог повідомлення про успішне створення сокету або про помилку.

10. Виконуємо перевірку, чи не виникло помилок при прив'язці до адреси.

11. Виводимо у лог повідомлення про успішну прив'язку до адреси або про помилку.

12. Переводимо сервер у режим приймання – чекаємо, доки клієнт отримає дані.

13. Виводимо у лог повідомлення про наявність символу.

14. Передаємо символ у зворотному напрямку.

NumBytesSend := Send(AcceptedSock, Buf, SizeOf(Buf), 0);

15. Виконуємо перевірку, чи не виникло помилок при передачі.

if (NumBytesSend = SOCKET_ERROR) then

16. Виводимо у лог повідомлення про те, що символ було передано або повідомлення про помилку.

17. Закриваємо бібліотеку сокетів.

WSACleanup;

18. Виконуємо перевірку, чи не виникло помилок при закритті бібліотеки.

if (WSACleanup = 0) then

19. Виводимо у лог повідомлення про те, що бібліотеку було закрито, або помилку.

WriteLogMessage('Server stoped.');

WriteLogMessage('Can not close socket library.');

20. Розблокуємо кнопки форми.

Контрольні запитання

1. Як називається мережний протокол прикладного рівня для передачі файлів?
2. З скількох частин складається кожен *HTTP* запит/відповідь?
3. Як виглядає рядок запиту *HTTP* ?
4. Як виглядає рядок відповіді *HTTP*?
5. Назвіть основні пункти узагальненого алгоритму клієнт-серверної програми.

15 ПРОЄКТУВАННЯ ЗАСОБІВ ДІАГНОСТИКИ З БІБЛІОТЕКОЮ *INDY*. БІБЛІОТЕКА *INDY*

Бібліотека *Indy* розроблена для використання потоків. Побудова серверів і клієнтів в *Indy* подібна побудові серверів і клієнтів в *Unix*.

Зазвичай сервери в *Unix* мають один або кілька процесів прослуховування, які спостерігають за запитами користувачів. Для кожного клієнта, який потребує обслуговування, створюється розгалуження (*fork*) процесу. Це робить програмування дуже простим, тому що кожний процес обслуговує тільки одного клієнта.

Сервери *Indy* працюють подібним чином. *Windows*, на відміну від *Unix*, не робить розгалуження, а створює новий потік. Сервери *Indy* створюють новий потік для кожного клієнтського з'єднання, а також потік прослуховування, що ізольований від головного кодового потоку програми. Потік прослуховування поєднує вхідні клієнтські запити. Для кожного клієнта, якому відповідають, створюється новий потік для його обслуговування. Необхідні події збуджуються в контексті даного потоку.

15.1 Методологія Indy

Indy відрізняється від інших сокетних компонентів. Майже всі інші компоненти працюють у неблокувальному режимі, асинхронно. Вони хочуть, щоб користувач реагував на події, створював машину станів і часто виконував цикли очікування. Наприклад, з іншими компонентами, коли ви робите з'єднання, то повинні очікувати подію з'єднання або крутити цикл очікування, поки факт з'єднання не буде встановлено. В *Indy* викликається метод *Connect* і очікується повернення з нього. Якщо з'єднання буде успішне, то буде отримано повідомлення про закінчення з'єднання. В протилежному випадку буде викликано виключення.

Робота з *Indy* аналогічна роботі з файлами. *Indy* дозволяє помістити весь код в одне місце замість створення різних оброблювачів подій. *Indy* є більш простим у використанні, розроблений для роботи з потоками.

15.2 Основні характеристики *Indy*

Основні характеристики ***Indy***

- *Indy* використовує *API* для блокування сокетів.
- *Indy* не орієнтований на події. *Indy* має події, але для інформаційних потреб, де вони необов'язкові.
- *Indy* розроблений для використання кодових потоків, проте може працювати без їх використання.
- Програмування в *Indy* – це лінійне програмування.

- *Indy* має високий рівень абстрагування. Більшість сокет-компонентів не дуже ефективно ізолюють програміста від стека. Більшість сокет-компонентів замість ізоляції від стека, навпаки, занурюють його в складності створення обгорток навколо цього в *Delphi / C++ Builder*.

15.3 Огляд клієнтів

Indy розроблено для створення високого рівня абстракції. Складності і деталізація *TCP/IP* стека сховані від програміста.

Типова клієнт - сесія в *Indy* виглядає так:

```
with IndyClient do begin
  Host := 'postcodes.atozedsoftware.com'; // Host to call
  Port := 6000; // Port to call the server on
  Connect;
  try
    // Do your communication
  finally
    Disconnect;
  end;
end;
```

15.4 Огляд серверів

Компоненти серверів *Indy* створюють потік прослуховування, що ізольований від головного кодового потоку програми. Потік прослуховування з'єднує вхідні клієнтські запити. Для кожного клієнта, якому відповідають, створюється новий потік для його обслуговування. Необхідні події збуджуються в контексті даного потоку.

15.5 Потоки

Для реалізації функціональності використовуються потоки. *Indy* дуже інтенсивно використовує потоки для реалізації серверів. Потоки так само використаються і у клієнтах. Неблокувальні сокети також можуть використовувати потоки, але вони вимагають деякої додаткової обробки і їх переваги губляться в порівнянні з блокувальними сокетами.

Бібліотека *Indy* має ряд функцій, які дозволяють тестувати мережу алгоритмами системи клієнт-сервер. Для нашої програми відібрані наступні функції:

IdHTTP1.Host – формує шлях до *HTTP*-серверу;

QueryPerformanceFrequency – обчислює частоту лічильника;

QueryPerformanceCounter – обчислює час у тактах;

IdHTTP1.Get – вилучає пакети з серверу;

Application.ProcessMessages – дозволяє отримувати наступний пакет з серверу;

AResponseInfo.ResponseNo – встановлює час очікування відповіді від клієнта;

AResponseInfo.ContentText – сервер передає зміст *HTML* документу клієнту;

AResponseInfo.CloseSession – закриває обмін інформацією.

Алгоритми тестування клієнт-серверної системи

Узагальнений алгоритм тестування системи:

1. Програма-клієнт видає запит до програми-серверу.
2. Програма-сервер приймає запит, виконує його та надсилає відповідь програмі-клієнта.
3. Клієнт отримує відповідь та реєструє час передання даних.

Переходимо до проєктування алгоритмів програм клієнт та сервер.

Алгоритм програми-клієнту:

- створення форми для посилання запиту:

```
procedure TForm1.Action;
begin
 IdHTTP1.Get('http://'+IdHTTP1.Host);
end;
```

- передача імені серверу, з яким треба встановити зв'язок:

IdHTTP1.Host:=LabeledEdit1.Text;

- обчислення частоти - КГц

```
QueryPerformanceFrequency(f);
freq:=f/1000;
```

- обчислення часу - мс

```
Result:=IntToStr(Round(t/freq));
```

- згідно з показником кількості запитів починається цикл:

✓ обчислення часу до з'єднання із сервером:

```
for i:=1 to SpinEdit1.Value do
begin
 QueryPerformanceCounter(s);
```

- з'єднання із сервером

Action;

✓ виконання передання-приймання інформації:

Application.ProcessMessages;

✓ обчислення часу після з'єднання із сервером та часу на виконання операції як різницю між отриманими результатами

total:=total+(f-s);

✓ запис до протоколу часу на виконання

Memo1.Lines.Add('Итерация '+IntToStr(i)+' время='+*GetMs(f-s)*+'мс');

end;

✓ обчислення середнього часу виконання

Memo1.Lines.Add(‘Среднее время ‘+*GetMs(Round(total/SpinEdit1.Value))*+’мс’);

end;

Алгоритм програми-серверу:

- завантаження *HTTP* - серверу
- Очікування запиту від клієнта:

Memo1.Lines.Add(‘[‘+*TimeToStr(now)*+’] Запросы получены. Адрес клиента:’+*AThread.Connection.Socket.Binding.PeerIP*);

- формування даних, що будуть передані:

AResponseInfo.ContentText:=‘<html><body><h1>Test Server</h1></body></html>‘;

- закриття сесії передачі даних

AResponseInfo.CloseSession();

15.6 Результати тестування для бібліотеки *ICS*

Програма *Server* запускається першою на тому ПК, зв’язок з яким ми перевіряємо з даного ПК і на якому буде запущена програма *Client*.

Для бібліотеки *ICS* повинна з’явитися форма, наведена рис. 15.1.

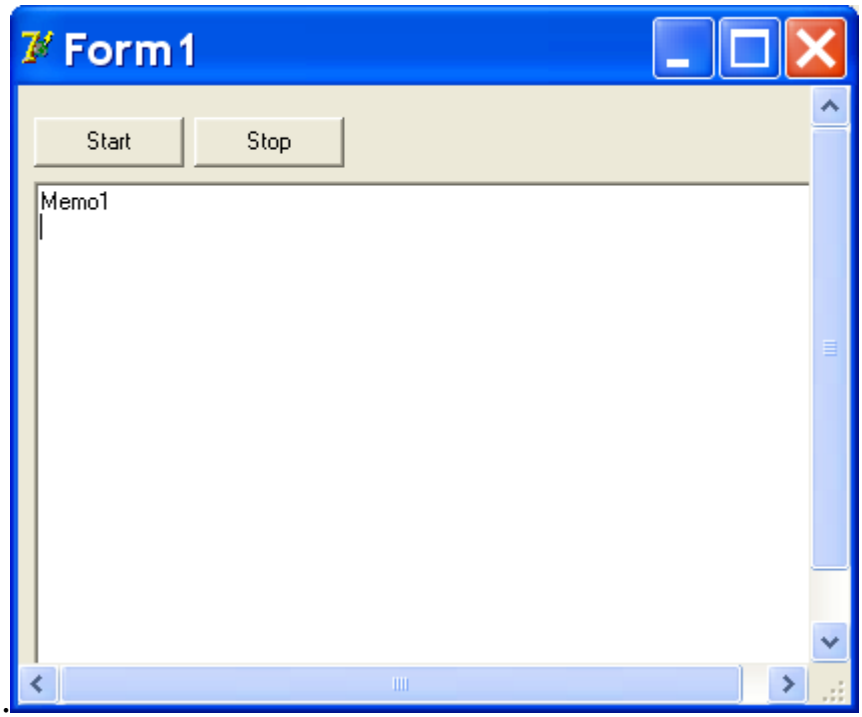

Рисунок 15.1 – Сервер

Після запуску форми „*Start*”отримаємо результат, наведений на рис. 15.2.

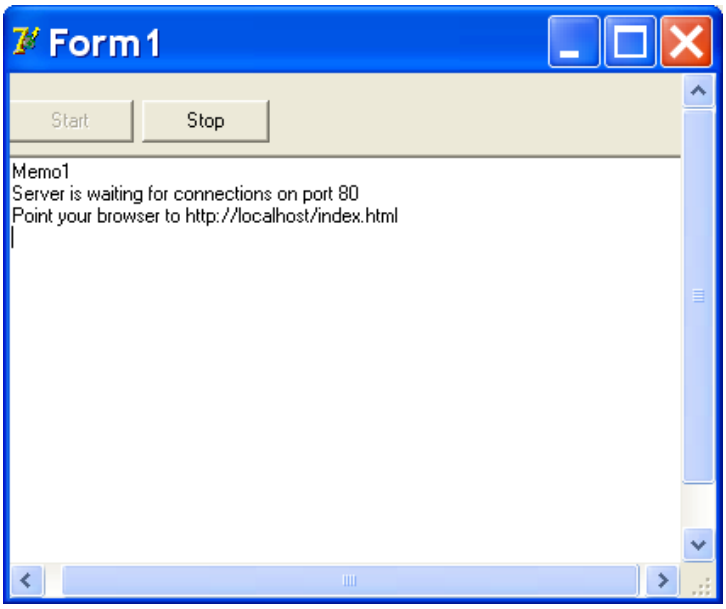

Рисунок 15.2 – Форма після запуску

Сервер готовий для роботи з клієнтом.

Після запуску програми клієнт повинен отримувати форму, наведену на рис.15.3.

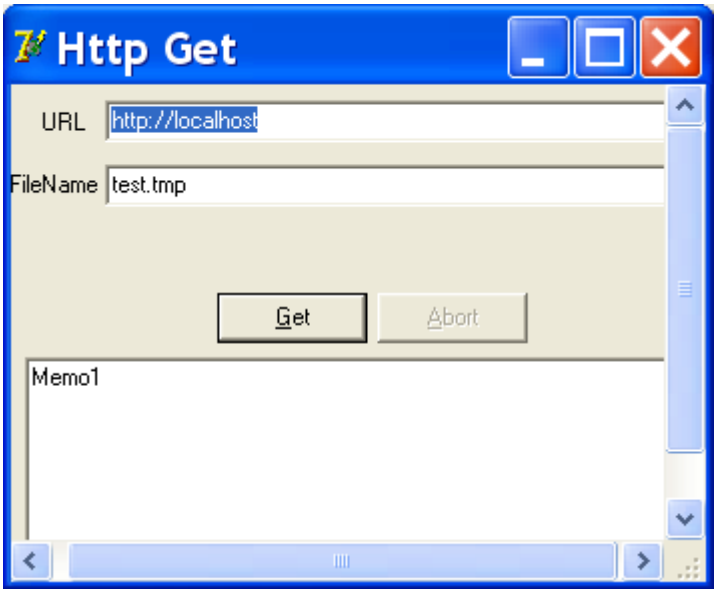

Рисунок 15.3 – Клієнт

Система готова для тестування. В нашому випадку ім'я серверу внесено. За необхідності можливе введення довільної адреси.

Виконуємо тестування, натиснувши на кнопку *Get*.

Результат роботи системи наведено на рис. 15.4.

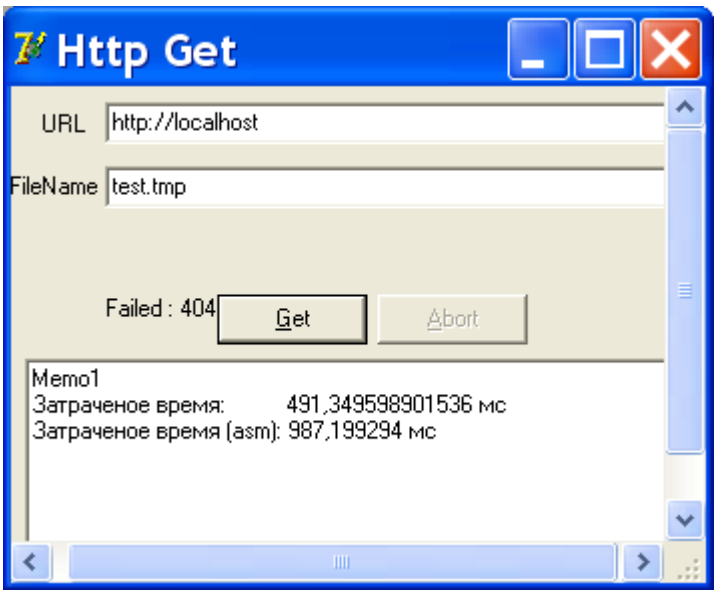

Рисунок 15.4 – Результат роботи системи

Якщо тестування виконано і мережа працездатна, то у формі буде повідомлення у вигляді подвійного часу проходження пакета. Замір двома методами дає можливість порівняння, що підвищує достовірність.

Відповідно на формі серверу – рис. 15.5:

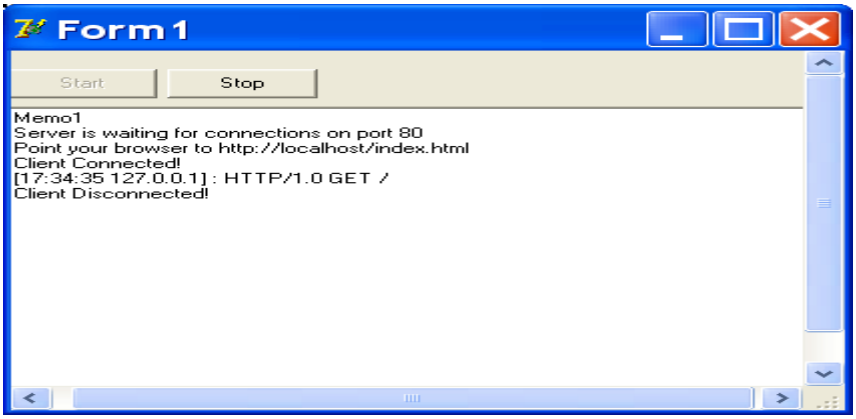

Рисунок 15.5 – Сервер.

Таким чином, за допомогою описаних програм можливо виконати тестування мережі з визначенням її працездатності та часу проходження пакета.

15.7 Результати тестування для бібліотеки *Indy*

Спочатку запускаємо програму- сервер та отримуємо форму, наведену на рис. 15.6.

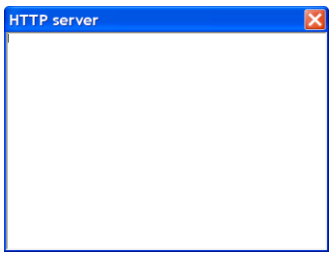

Рисунок. 15.6. – Початок роботи програми-сервер.

Запускаємо програму-клієнт і отримуємо форму, наведену на рис. 15.7. Для подальшої роботи вводимо ім'я хосту та кількість посилок. Натискаємо на СТАРТ. Отримаємо результати роботи системи тестування для форми клієнта (рис. 15.8) та результат роботи системи для форми серверу (рис. 15.9).

На кожний подвійний прохід отримали час у мс.

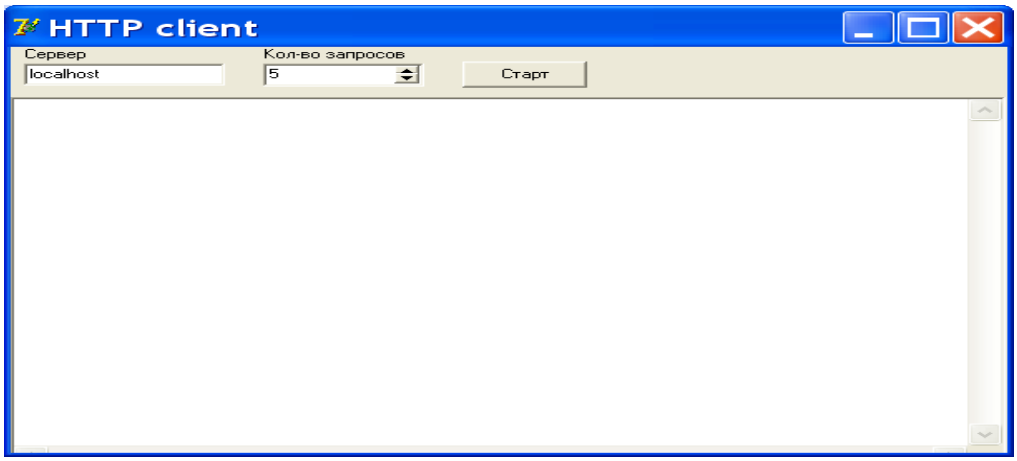

Рисунок 15.7 – Початок роботи програми-клієнт

Результати роботи системи:

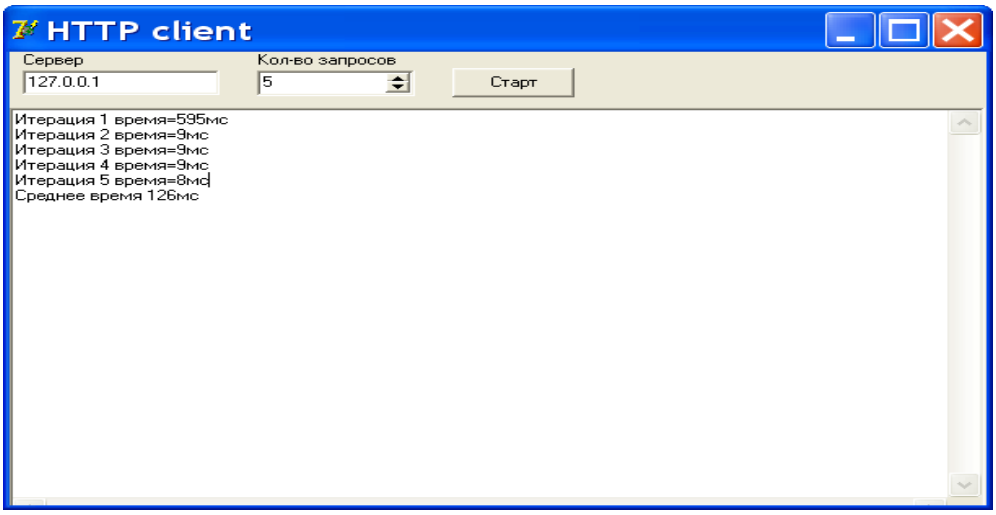

Рисунок 15.8 – Результати роботи системи для програми – клієнт

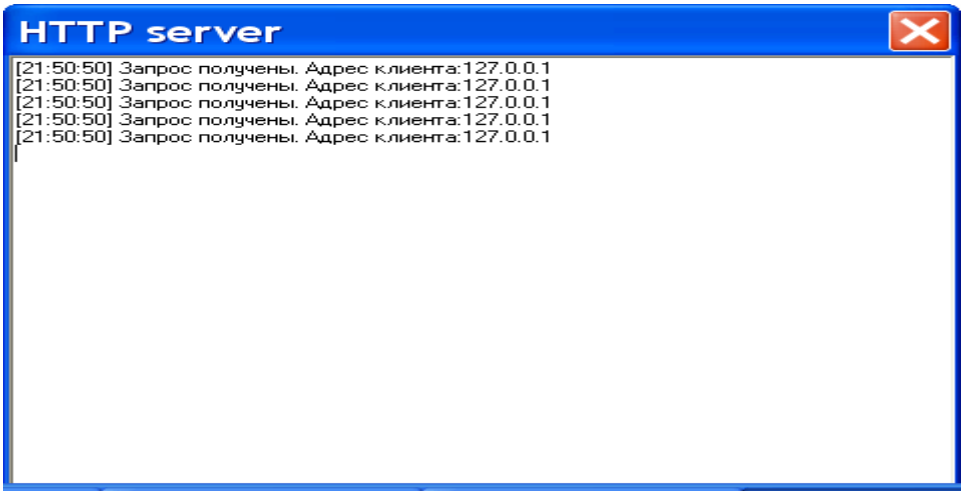

Рисунок 15.9 – Результати роботи системи для програми-сервер

Коли отримаємо на клієнтській формі час подвійного проходу пакета, то це буде результат тестування. Якщо час збігається з можливим для даного каналу двох ПК, то результат позитивний – мережа працездатна.

Контрольні запитання

1. Для використання чого розроблена бібліотека *Indy*?
2. Як працюють сервери *Indy*?
3. Чи дозволяє *Indy* помістити весь код в одне місце замість створення різних оброблювачів подій?
4. Які основні характеристики *Indy?*
5. Назвіть функції бібліотека *Indy*, які дозволяють тестувати мережу алгоритмами системи клієнт – сервер?
6. Який алгоритм тестування клієнт-серверної системи?

16 ПРОЄКТУВАННЯ ПРОГРАМИ ТИПУ УТИЛІТИ *IPCONFIG*

16.1 Робота утиліти *IPconfig*

Робота утиліти *IP-config* наведена на рис. 16.1.

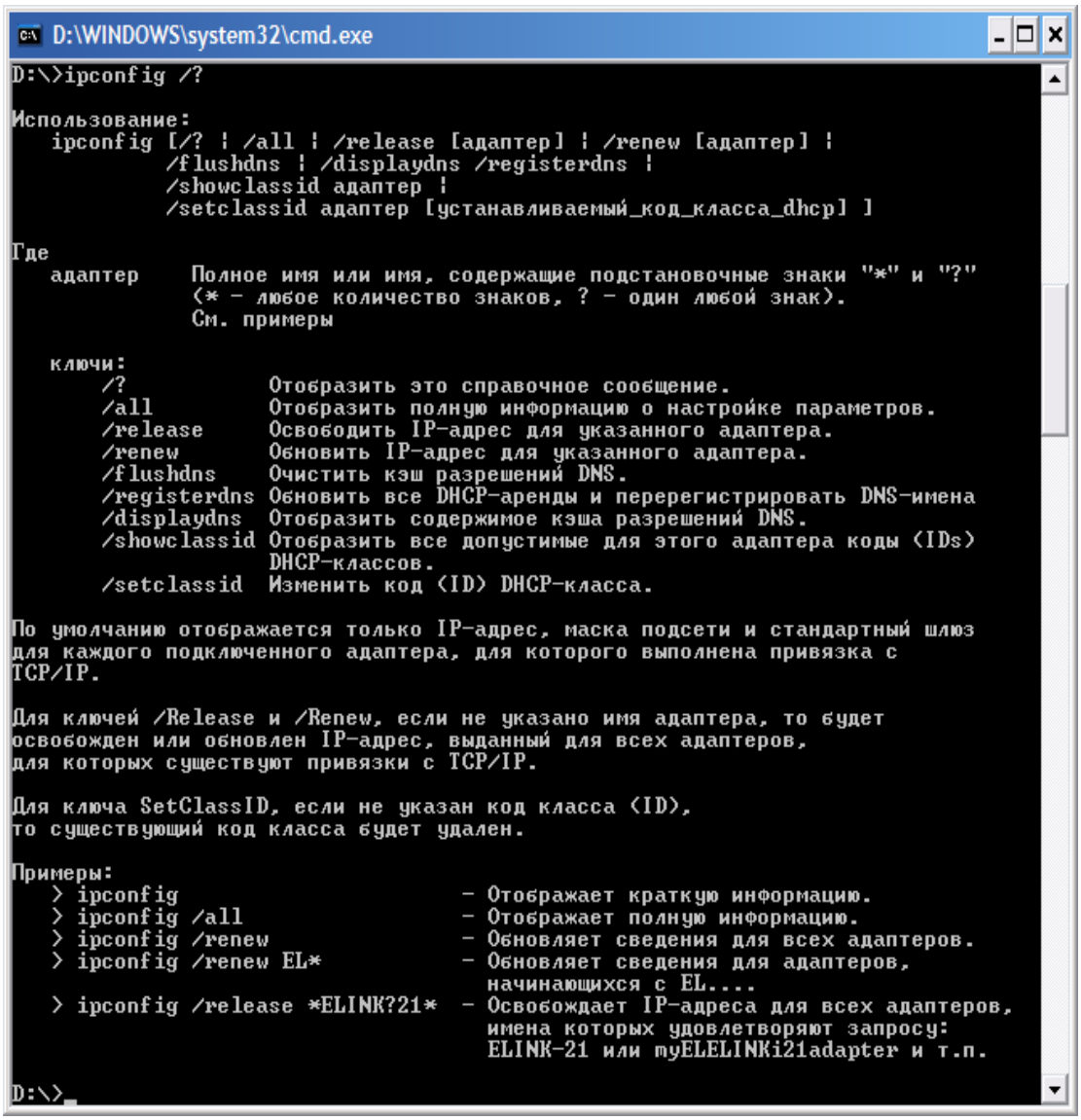

Рисунок 16.1 – Робота системної утиліти ***IPconfig***

З цього рисунку видно, які ключі має утиліта, а саме:

/all – Відобразити повну інформацію про настроювання параметрів.

/release – Звільнити *IP*-адресу для зазначеного адаптера.

/renew – Обновити *IP*-адресу для зазначеного адаптера.

/*flushdns* – Очистити кеш вирішень *DNS*.

/*registerdns* – Обновити всі *DHCP* оренди і перереєструвати *DNS* імена.

/*displaydns* – Відобразити вміст кеша дозволів *DNS*.

/*showclassid* – Відобразити всі припустимі для цього адаптера коди (*IDs) DHCP*-класів.

/*setclassid* – Змінити код (*ID*) *DHCP*-класу.

16.2 Склад інформаційної програми та алгоритми роботи її частин

Розроблена програма складається з таких функцій (головних):

- *PageControl1Change()*;
- *GetIPInfo()*;
- *FormShow()*;
- *GetAdapterInfo()*.

Алгоритми роботи складових програми

1) Перевірка роботи мережі:

```
//перевірка інформації про мережі - результат в Err
Err:=GetNetworkParams(nil, FixedlnfoSize);
if (Err<>O) and (Err<>ERROR_BUFFER_OVERFLOW) then
//якщо відбулася помилка, вивід повідомлення і вихід
begin
HostNameLabel.Caption:= 'Error';
exit;
end;
```

2) Запис інформації про мережу:

```
//якщо помилки немає, тоді записуємо інформацію про мережі в
pFixedlnfo
    pFixedlnfo:=PFIXED_INFO(GlobalAlloc(GPTR, FixedlnfoSize));
    GetNetworkParams(pFixedlnfo, FixedlnfoSize);
```

Спочатку викликається функція *GetNetworkParams* із двома параметрами. Ця функція повертає інформацію про мережу. Але для одержання цих даних нам потрібно знати їх розмір. Щоб це зробити, потрібно спочатку перший параметр встановити рівним *nil*. Другий параметр є змінною, куди буде записаний розмір необхідної пам'яті для одержання повної інформації. Результат виконання функції записується в змінну *Err*, що дає змогу перевірити наявність помилок. Якщо значення

змінної не дорівнює 0 і не дорівнює константі *ERROR_BUFFER_OVERFLOW*, то це є ознакою наявності помилки. В такому випадку необхідно вивести про це повідомлення і вийти із процедури. Наявність такої помилки свідчить про те, що мережна карта не настроєна.

Після цього виділяємо пам'ять для структури *pFixedlnfo* типу *PFIXED_INFO*, куди ми будемо записувати отриману інформацію.

Пам'ять виділяється відповідно до результатів першого виклику функції *GetNetworkParams*.

Далі знову викликається функція *GetNetworkParams*, тільки тепер як перший параметр зазначається структура *pFixedlnfo*.

Все необхідне ми одержали, і воно знаходиться в структурі *pFixedlnfo*.

3) Читання інформації з структури:

розглянемо детально цю структуру *pFixedlnfo*:

- *pFixedlnfo.HostName* – ім'я вашого комп'ютера; Наведемо відразу частину коду із програми:

```
//одержання ім'я ПК
HostNameLabel.Caption:=StrPas(pFixedInfo.HostName);
DNSListBox.Items.Clear;
DNSListBox.Items.Add(StrPas(pFixedInfo.DnsServerList.IpAddress.S));
```

- *pFixedlnfo.DnsServerList.IpAddress* – адреса *DNS*-серверу. Якщо такий сервер не один, то потрібно викликати *pFixedlnfo.DnsServerList.Next*, щоб одержати доступ до наступного. Як результат виклику *Next* до нас повернеться змінна типу *PIP_ADDR_STRING*, через яку і можна одержати наступну адресу *DNS*-серверу.
- *pFixedinfo.NodeType* – тип вузла. Тут зберігається ціле число. Якщо воно дорівнює 1, то, значить, тип вузла *Broadcast*, 2 – *Peer to peer*, 4 – *Mixed*, 8 – *Hybrid*;
- *pFixedinfo.Scopeid* – ідентифікатор *NetBIOS* (*NetBIOS Scope ID*);
- *pFixedlnfo.EnableRouting* – чи включена маршрутизація *IP*. Якщо тут зберігається *true*, то маршрутизація включена, інакше відключена;
- *pixedinfo.EnableProxy* – чи включений *WINS Proxy*. Якщо тут знаходиться *true*, то *Proxy*-сервер включено, інакше відключено.

4) Отримання імені ПК:

```
//отримуємо назву комп'ютера
 HostNameLabel.Caption:=StrPas(pFixedInfo.HostName);
 DNSListBox.Items.Clear;
DNSListBox.Items.Add(StrPas(pFixedInfo.DnsServerList.IpAddress.S));
```

5) Отримання *DNS*:

```
 pAddrStr:=pFixedInfo.DnsServerList.Next;
while (pAddrStr<>nil) do
  begin
   DNSListBox.Items.Add(StrPas(pAddrStr.IpAddress.S));
```

```
 pAddrStr:=pAddrStr.Next;
end;
```
6) Отримання типу вузла:
```
//визначаємо тип вузла
case pFixedInfo.NodeType of
 1: NodeTypeLabel.Caption:= 'Broadcast';
 2: NodeTypeLabel.Caption:= 'Peer to peer';
 4: NodeTypeLabel.Caption:= 'Mixed';
 8: NodeTypeLabel.Caption:= 'Hybrid';
end;
NetBIOSScopeLabel.Caption:=pFixedInfo.ScopeId;
```
7) Отримання стану маршрутизації:
```
//перевірка на включення маршрутизації IP
if pFixedInfo.EnableRouting>0 then
 IPRoutingLabel.Caption:= 'Yes'
else
 IPRoutingLabel.Caption:= 'No';
```
8) Отримання стану проксі-серверу:
```
//перевірка на включення проксі-серверу
if pFixedInfo.EnableProxy>0 then
 WINSProxyLabel.Caption:= 'Yes'
else
 WINSProxyLabel.Caption:= 'No';
if pFixedInfo.EnableDns>0 then
 NetBIOSResolutionLabel.Caption:= 'Yes'
else
 NetBIOSResolutionLabel.Caption:= 'No';
```
9) Отримання інформації про пристрої:
для цього виконується виклик функції *ForraShow()*:

procedure TSystemlnfoForm.ForraShow(Sender: TObject);
яка має 2 структури:
pAdapterlnfo, pAdapt:PIP_ADAPTER__INFO;
pAddrStr: PIP__ADDR_STRING;
алгоритм функції:

1. Очищуємо список пристроїв:
```
AdapterCB.Items.Clear;
```
2. Одержати кількість пристроїв:
```
AdapterlnfoSize:=0;
Err:=GetAdaptersInfo(nil, AdapterlnfoSize);
```
Якщо відбулася помилка, тоді:
```
if (Erro) and (Err<>ERROR__BUFFER_OVERFLOW) then
begin AdapterCB.Items.Add('Error'); exit; end;
```
3. Одержати інформацію про пристрої:

pAdapterlnfo := PIP_ADAPTER_INFO(GlobalAlloc(GPTR, AdapterlnfoSize)

GetAdaptersInfo(pAdapterlnfo, AdapterlnfoSize);

pAdapt := pAdapterlnfo;

4. Перевіряємо тип отриманого адаптера:

while pAdaptonil do
begin
case pAdapt.Type_ of
MIB IF TYPE ETHERNET:
AdapterCB.Items.Add('Ethernet adapter '+pAdapt.AdapterName);
MIB_IFJTYPEJTOKENRING:
AdapterCB.Items.Add('Token Ring adapter '+pAdapt.AdapterName);
MIB_IF_TYPE__FDDI:
AdapterCB.Items.Add('FDDI adapter +pAdapt.AdapterName);
MIB_IFJTYPE_PPP:
AdapterCB.Items.Add('PPP adapter +pAdapt.AdapterNarae);
MXB_IF_TYPE_LOOPBACK:
AdapterCB.Items.Addf'Loopback adapter '+pAdapt.AdapterName);
MIB_IF_TYPE_SLIP:
AdapterCB.Items.Add('Slip adapter '+pAdapt.AdapterName};
MIB_IF_TYPE_OTHER:
AdapterCB.Items.Add('Other adapter '+pAdapt.AdapterName);end;
pAdapt := pAdapt.Next;end;
GlobalFree(Cardinal(pFixedlnfo));end;

Головним у цій процедурі є виклик функції *GetAdaptersinfo*. Перший раз вона викликається з нульовим першим параметром. Це змушує ОС повідомити нам, скільки пам'яті необхідно для зберігання інформації про встановлені пристрої. Цю інформацію ми одержуємо через змінну, зазначену в другому параметрі. Після цього виділяємо необхідну кількість пам'яті. Пам'ять виділяється у глобальній пам'яті машини за допомогою функції *GlobalAlloc*. Вдруге функція *GetAdaptersinfo* викликається вже з усіма заданими параметрами. Перший параметр місить показчик на виділену пам'ять, а другий параметр вказує на кількість цієї пам'яті.

Після одержання необхідної інформації про встановлені пристрої заповнюємо список *comboBox* іменами знайдених мережних плат (ці імена знаходяться в *pAdapt.Adapter.Name*). Додаємо до імені адаптера його тип, який зберігається в структурі *pAdapt* і може приймати такі значення:

- *MIB_IF_TYPE_ETHERNET* – *Ethernet* мережний адаптер;
- *MIB_IF_TYPE_TOKENRING* – адаптер *Token Ring*;
- *MIB_IFJTYPE_FDDI* – адаптер *FDDI*;
- *MIB_IF_TYPE_PPP* – *PPP*-адаптер;
- *MIB_IF_JTYPE_LQOPBACK* – адаптер *LoopBack*;
- *MIB_IF_TYPE_SLIP* – *Slip*-адаптер;
- *MIB_IF_TYPE_OTHER* – інше.

Тепер у списку *ComboBox* знаходяться імена всіх знайдених пристроїв.

5. Отримання інформації про кількість встановлених пристроїв. Для цього треба відслідковувати подію, коли користувач вибирає якийсь елемент зі списку і виводить інформацію щодо цього елемента. Щоб піймати таку подію, необхідно створити оброблювач *OnChange.* У цьому оброблювачі необхідно одержати кількість установлених пристроїв. Для цього викликаємо функцію *GetAdaptersinfo* з нульовим першим параметром, щоб довідатися про кількість необхідної пам'яті для зберігання інформації про пристрої:

AdapterInfoSize:=0;

Err:=GetAdaptersInfo(nil, AdapterlnfoSize);

Далі знову одержуємо список пристроїв:

pAdapterlnfo := PIP_ADAPTER_INFO(GlobalAlloc[GPTR, AdapterlnfoSize));

GetAdaptersInfo(pAdapterInfo, AdapterlnfoSize);

pAdapt := pAdapterlnfo;

Після цього організується цикл *while*, у якому відбувається перевірка всіх назв з отриманого списку пристроїв з тим ім'ям, який вибрав користувач. Як тільки цей запис знайдено, потрібно вивести інформацію зі структури *pAdapterInfo*:

IP_ADAPTER_INFO = record

Next: PIP_ADAPTER_INFO;

Combolndex: DWORD;

AdapterName: array [0..MAX_ADAPTER_NAME_LENGTH + 3] of Char;

Description: array [0..MAX_ADAPTER_DESCRIPTION_LENGTH + 3] of Char;

AddressLength: UINT;

Address: array [0..MAX_ADAPTER_ADDRESS_LENGTH - 1] of BYTE;

Index: DWORD;

Typi_: UINT;

DhcpEnabled: UINT;

CurrentlpAddress: PIP_ADDR_STRING;

IpAddressList: IP_ADDR_STRING;

GatewayList: IP_ADDR_STRING;

DhcpServer: IP_ADDR_STRING;

HaveWins: BOOL;

PrimaryWinsServer: IP_ADDR_STRING;

SecondaryWinsServer: IP_ADDR_STRING;

LeaseObtained: time__t;

LeaseExpires: time_t;

end;

Структура містить такі значення:

- *Comboindex* – індекс пристрою.
- *AdapterName* – ім’я мережного адаптера.
- *Description* – текстовий опис адаптера.
- *AddressLength* – довжина фізичної (*MAC*) адреси.
- *Address* – масив символів, у якому зберігається сама адреса. Довжина масиву визначається попереднім параметром.
- *mypi* – тип адаптера.
- *DhcpEnabled* – вказує на використання *DHCP*.
- *CurrentlpAddress* – поточна *IP*-адреса.
- *GatewayList* – список шлюзів.
- *DHCPServer* – *IP*-адреса *DHCP*-серверу.
- *HaveWins* — інформація про використаня *WINS*-серверу.
- *PrimaryWinsServer* – головний *WINS*-сервер.
- *SecondaryWinsServer* — допоміжний *WINS*-сервер.

16.3 Результати роботи програми

Організована таким чином програма дозволяє, наприклад, отримати інформацію такого вигляду (рис. 16.2, 16.3).

Програма реалізує наступні функції:

- визначає назву ПК, *IP*-адресу *DNS*-серверу;
- *IP*- та фізичну адресу ПК, назву мережної плати;
- визначає маску та шлюз.

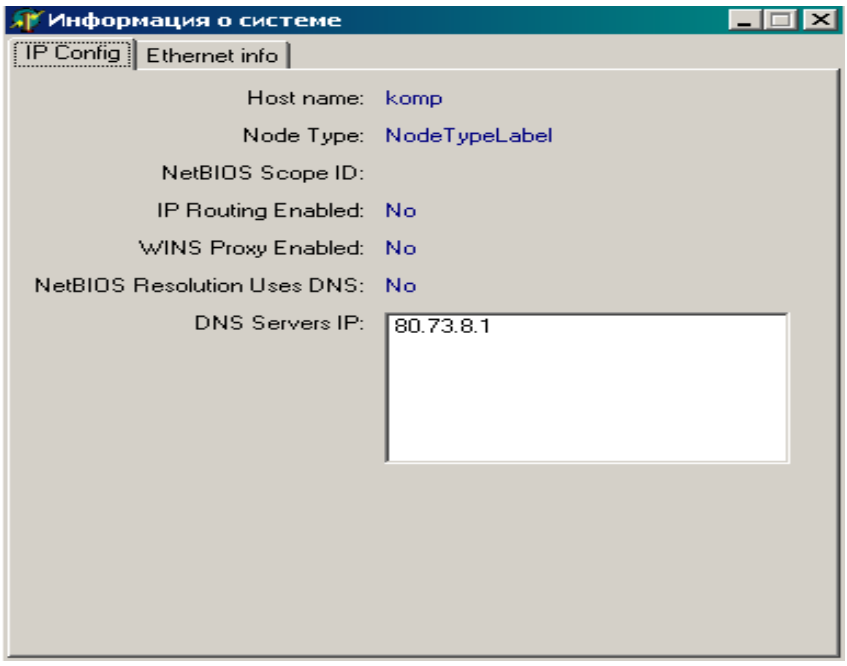

Рисунок 16.2 – Вікно розробленої програми (перша закладка)

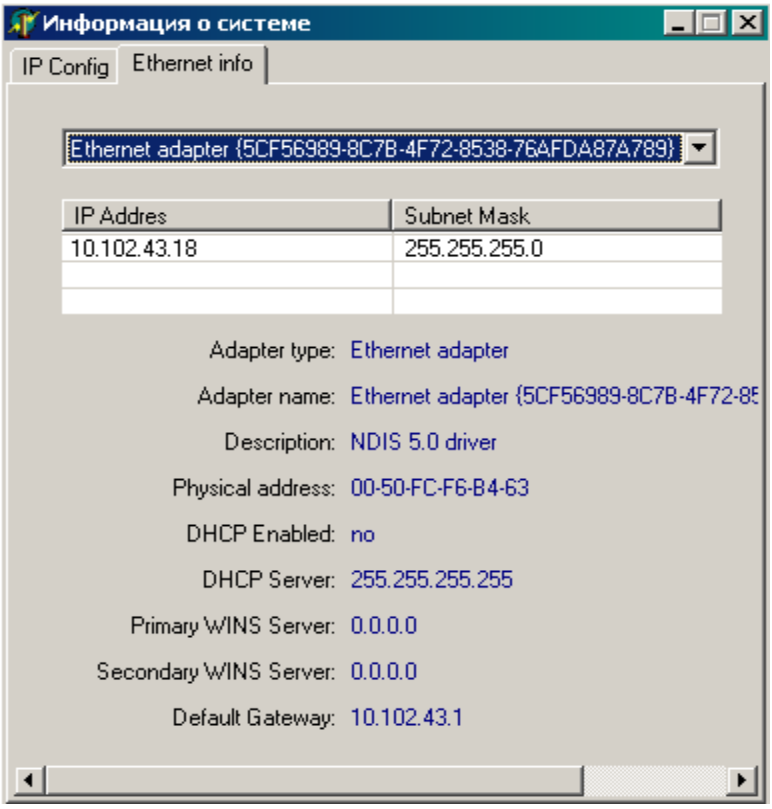

Рисунок 16.3 – Вікно розробленої програми (друга закладка)

17 ПРОЄКТУВАННЯ ПРОГРАМИ ТИПУ УТИЛІТИ *WHOIS*

Проектування програми тестування каналу зв'язку між двома ПК за допомогою протоколу *WHOIS* є клієнт-серверною системою, яка передбачає необхідність проєктування двох різних програм:

1. Програма – клієнт. Це активна програма системи, яка здійснює запит ПК з БД, тобто програмі – серверу, вимірює час проходження запиту – відповіді і виводить результат перевірки працездатності каналу.

2. Програма – сервер. Ця програма запускається першою в системі і очікує запиту від програми – клієнта. Отримавши запит, дає відповідь.

Розглянемо протокол *WhoIs* як складову системи.

17.1 Особливості протоколу *WhoIs*

WHOIS (від англ. *who is* – «хто такий») – мережний протокол прикладного рівня, який базується на протоколі *TCP* (порт 43), призначений для запиту віддалених баз даних (БД). Протокол має на увазі архітектуру «клієнт – сервер». Протокол *WHOIS* в основному використовується для доступу до публічних БД реєстраторів *IP*-адрес і реєстраторів доменних імен. Поточна версія цього протоколу описана в *RFC* 3912. Частіше використовуються *WHOIS*-клієнти командного рядка. Оскільки для багатьох користувачів командний рядок недоступний або незручний, то для таких клієнтів зроблені *Web*-форми, доступні користувачам за протоколом *HTTP*. Також є і *WHOIS*s-клієнти з графічним інтерфейсом.

Бази даних, що мають *WHOIS*-інтерфейс, бувають централізованими і розподіленими. У першому випадку один *WHOIS*-сервер містить повну БД і відповідає на запити, що стосуються всіх реєстраторів. За такою схемою побудований *WHOIS*-сервер для домену *org*. У другому випадку центральний *WHOIS*-сервер не містить повну БД і лише перенаправляє користувача на *WHOIS*-сервер відповідного реєстратора. За такою схемою працює *WHOIS* для домену *com*. Коли *WHOIS*-клієнт «вміє» розпізнавати таке перенаправлення, він сам запрошує потрібний периферійний *WHOIS*-сервер, інакше користувачу доводиться робити це вручну. У протоколі *WHOIS* не передбачено розрізнення централізованої і розподіленої моделей.

Формат повернених даних не регламентований протоколом. У різних реєстраторів можуть бути різні формати записів (у розподіленій моделі).

17.2 Узагальнений алгоритм тестування системи

1. Запуск програми-серверу:

1.1) створення компонента для обробки запитів клієнта;

1.2) відповідь на запити клієнта.

2. Запуск програми-клієнта;

2.1) введення адреси серверу;

2.2) запит до програми-серверу з вимірюванням часу відповіді програми-серверу на запит програми-клієнта;

2.3) виконання пункту 2.2 5 разів;

2.4) відображення на екрані отриманих результатів виміряння часу та середнього часу виконання 5-х запитів.

Опис алгоритму програми-клієнт

Алгоритм:

1) опис функцій та змінних, що використовуються;

2) створюємо діалогове вікно - *Form1: TForm1;*

3) отримуємо частоту лічильника по натискуванню клавіші – *QueryPerformanceFrequency(f);*

4) задаємо адресу серверу – *IdWhois1.Host:=LabeledEdit1.Text;*

5) отримуємо значення лічильника в даний момент часу – *QueryPerformanceCounter(start);*

6) переводимо компонент до активного стану – *Action*;

7) виконуємо обмін даними – *IdWhois1.WhoIs(LabeledEdit2.Text);*

8) переводимо компонент до неактивного стану – *IdWhois1.Disconnect;*

9) отримання значення лічильника в даний момент часу – *QueryPerformanceCounter(finish);*

10) підраховуємо загальний час обміну даними – *Result:=IntToStr(round(t/freq));*

11) виводимо результат на екран – *Memo1.Lines.Add*(‘Ітерація №’+*IntToStr(i)*+’ час ‘+*GetMs(finish-start)*+’мс’);

Опис алгоритму програми-сервер

Алгоритм:

1) запуск програми-серверу;

2) створення компонента і передача йому керування.

Результати роботи програм

У результаті роботи створюється діалогове вікно програми-клієнта (рис 17.1.) та діалогове вікно програми-серверу (рис. 17.2) з відображенням отриманих результатів.

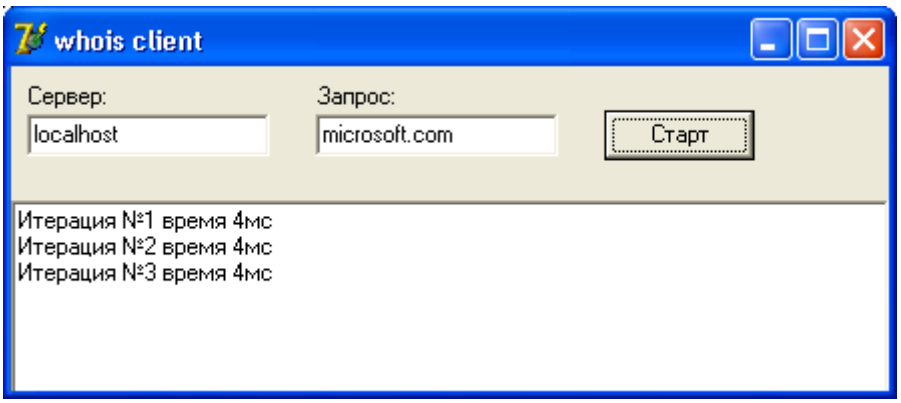

Рисунок 17.1 – Діалогове вікно програми-клієнт

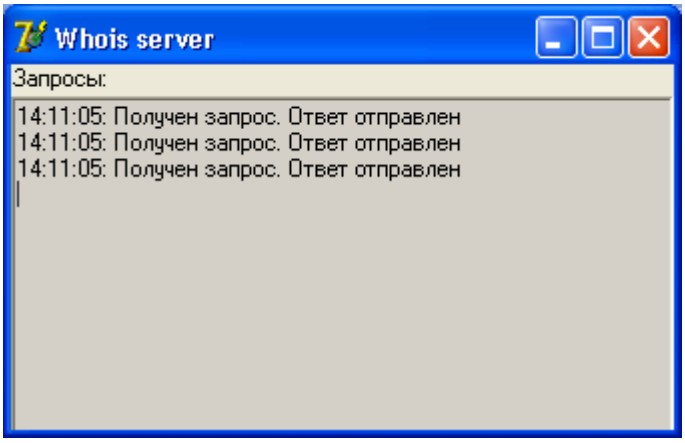

Рисунок 17.2 – Діалогове вікно програми-сервер

18 ПРОЄКТУВАННЯ СИСТЕМИ ТЕСТУВАННЯ МЕРЕЖ З ПРОТОКОЛОМ *POP3*

18.1 Особливості протоколу *POP3*

У деяких невеликих вузлах Інтернет буває непрактично підтримувати систему передачі повідомлень (*MTS – Message Transport System*). Робоча станція може не мати достатніх ресурсів для забезпечення безперервної роботи *SMTP*-серверу. Для "домашніх ПК" занадто дорого підтримувати зв'язок з Інтернет цілодобово.

Але доступ до електронної пошти необхідний як для таких малих вузлів, так і індивідуальних ПК. Для розв'язання цієї проблеми розроблено протокол *POP3* (*Post Office Protocol - Version* 3, *RFC*-1939). Цей протокол забезпечує доступ вузла до базового поштового серверу.

POP3 не ставить за мету надання широкого списку маніпуляцій з поштою, він лише одержує і стирає поштові повідомлення.

Коли користувач ПК-клієнта хоче послати повідомлення, він встановлює *SMTP*-зв'язок з поштовим сервером безпосередньо і посилає все, що потрібно через нього. При цьому ПК *POP3*-сервер не обов'язково є поштовим сервером.

18.2 Взаємодія за протоколом *POP3*

Перед роботою через протокол *POP3*-сервер прослуховує порт 110. Коли клієнт хоче використати цей протокол, він повинен створити *TCP*-з'єднання із сервером. Коли з'єднання встановлено, сервер відправляє запрошення. Потім клієнт і *POP3*-сервер обмінюються інформацією, поки з'єднання не буде закрите або перерване.

Команди *POP3* складаються із ключових слів. Усі команди закінчуються парою *CRLF*. Ключові слова і аргументи складаються із *ASCII* символів. Ключове слово і аргументи розділені одиночним пробілом. Ключове слово складається від 3-х до 4-х символів, а аргумент може бути довжиною до 40-ка символів.

Відповіді в *POP3* складаються з індикатора стану і ключового слова, за яким може випливати додаткова інформація. Відповідь закінчується парою *CRLF*. Існує тільки два індикатори стану: "+*OK*" – позитивний і "-*ERR*" – негативний.

Відповіді на деякі команди можуть складатися з декількох рядків. У цих випадках кожний рядок розділений парою *CRLF*, а кінець відповіді закінчується *ASCII* символом 46 (".") і парою *CRLF*.

POP3-сесія складається з декількох режимів. Як тільки з'єднання із сервером було встановлено і сервер відправив запрошення, сесія переходить у режим *AUTHORIZATION* (авторизація). У цьому режимі клієнт повинен ідентифікувати себе на сервері. Після успішної ідентифікації сесія переходить у режим *TRANSACTION* (передача). У цьому режимі клієнт запитує сервер про можливість виконання певних команд. Коли клієнт відправляє команду *QUIT*, сесія переходить у режим *UPDATE*. У цьому режимі *POP3*-сервер звільняє всі зайняті ресурси і завершує роботу. Після цього *TCP*-з'єднання закривається.

У *POP3*-серверу може бути *INACTIVITY AUTOLOGOUT* таймер. Цей таймер фіксує певний інтервал, наприклад 10 хвилин. Це значить, що якщо клієнт і сервер не взаємодіють один з одним на протязі інтервалу, то сервер автоматично перериває з'єднання і при цьому не переходить у режим *UPDATE*.

18.3 Режим авторизації

Як тільки буде встановлене *TCP*-з'єднання з *POP3*-сервером, він відправляє запрошення, що закінчується парою *CRLF*, наприклад:

S: +OK POP3 server ready

Тепер *POP3*-сесія перебуває в режимі *AUTHORIZATION*. Клієнт повинен ідентифікувати себе на сервері, використовуючи команди *USER* і *PASS*. Спочатку треба відправити команду *USER*, для як аргумент вказати ім'я користувача. Якщо сервер відповідає позитивно, то тепер необхідно відправити команду *PASS*, за якою іде пароль. Якщо після відправлення команди *USER* або *PASS* сервер відповідає негативно, то можна спробувати авторизуватися знову або вийти із сесії за допомогою команди *QUIT*. Після успішної авторизації сервер відкриває і блокує *maildrop* (поштова скринька). У відповіді на команду *PASS* сервер повідомляє скільки повідомлень перебуває в поштовій скриньці і передає їхній загальний розмір.

Команди *USER* та *PASS* мають наступний формат:

USER [ім'я] – команда передачі серверу імені користувача;

Аргументи: [ім'я] – рядок, що вказує ім'я поштової скриньки;

PASS [пароль] – команда передачі паролю;

Аргументи: [пароль] – пароль для поштової скриньки.

Приклад обміном повідомлень між клієнтом та сервером в режимі авторизації при правильному та некоректному введені паролю:

C: USER MyName
S: +OK MyName знайдено

C: PASS my_cool_password
S: +OK MyName `s maildrop has 2 messages (320 octets)
...
C: USER MyName
S: +OK MyName знайдено
C: PASS my_wrong_password
S: -ERR maildrop already locked

Закриття сесії можна виконати, використовуючи команду *QUIT*.

18.4 Режим передачі

Після успішної ідентифікації користувача на сервері *POP3* сесія переходить у режим *TRANSACTION*, де користувач може передавати наступні команди. Після кожної з таких команд надходить відповідь серверу. Доступні команди в цьому режимі:

- *STAT*

Аргументи: немає.

Опис: У відповідь на виклик команди сервер видає позитивну відповідь "*+OK*", за яким йде кількість повідомлень у поштовій скриньці і їхній загальний розмір у символах. Повідомлення, які позначені для видалення, не враховуються у відповіді серверу.

Можливі відповіді:

+OK n s

Приклад:

C: STAT

S: +OK 2 320

- *LIST* [повідомлення]

Аргументи: [повідомлення] – номер повідомлення (необов'язковий аргумент).

Опис: Якщо був переданий аргумент, то сервер видає інформацію про зазначене повідомлення. Якщо аргумент не був переданий, то сервер видає інформацію про всі повідомлення, що знаходяться у поштовій скриньці. Повідомлення, позначені для видалення, не перераховуються.

Можливі відповіді:

+OK scan listing follows

-ERR no such message

Приклад:

C: LIST

S: +OK 2 messages (320 octets)

S: 1 120

S: 2 200

S: .

...

C: LIST 2

S: +OK 2 200

• *RETR* [повідомлення]
Аргументи: [повідомлення] – номер повідомлення.

Опис: Після позитивної відповіді сервер передає зміст повідомлення.

Можливі відповіді:

+OK message follows

-ERR no such message

Приклад:

C: RETR 1

S: +OK 120 octets

S:

S: .

• *DELE* [повідомлення]
Аргументи: [повідомлення] – номер повідомлення.

Опис: *POP3*-сервер позначає зазначене повідомлення як вилучене, але не видаляє його, поки сесія не перейде в режим *UPDATE*.

Можливі відповіді:

+OK message deleted

-ERR no such message

Приклад:

C: DELE 1

S: +OK message 1 deleted

...

C: DELE 2

S: -ERR message 2 already deleted

• *NOOP*

Аргументи: немає

Опис: *POP3*-сервер нічого не робить і відповідає позитивно.

Можливі відповіді:

+OK

Приклад:

C: NOOP

S: +OK

• *RSET*

Аргументи: немає.

Опис: Якщо які-небудь повідомлення були позначені для видалення, то з них знімається ця мітка.

Можливі відповіді:

+OK

Приклад:

C: RSET

S: +OK maildrop has 2 messages (320 octets)

Коли клієнт передає команду *QUIT* у режимі *TRANSACTION*, то сесія переходить у режим *UPDATE*. В цьому режимі сервер видаляє всі повідомлення, позначені для видалення. Після цього *TCP*-з'єднання закривається.

Існують і додаткові *POP3*-команди, що дають більшу волю при роботі з повідомленнями: *TOP*, *UIDL* та ін.

18.5 Організація системи тестування

Розробка складається з двох програм – клієнтської та серверної, що демонструють реалізацію протоколу *POP3* для отримання електронних поштових листів (повідомлень). Також ці програми відображають час приймання та передачі повідомлення.

Серверний додаток реалізує такі функції:

• надає доступ до ресурсу "поштова скринька" клієнтським додаткам;

• відображає всі повідомлення, що знаходяться в "поштовій скринці";

- надає можливість додавання нових повідомлень у скриньку та їх збереження у файлі;
- функціонує за протоколом *POP3*.

Клієнтський додаток виконує такі функції:

- налагоджує зв'язок з *POP3*-сервером;
- організує отримання листів з серверу та їх перегляд;
- обчислює та відображує час приймання та передачі повідомлення;
- функціонує за протоколом *POP3*.

Як відомо в ОС *Windows* здійснюється програмування подій, тобто відповідні дії програми відбуваються внаслідок визначених подій. Обидві програми – серверна та клієнтська – написані в середовищі розробки *Delphi7* та використовують стандартні компоненти *TIdPOP3* і *TIdPOP3Server* (закладка *Indy*) для роботи в мережі за протоколом *POP3*.

Алгоритм програми – сервер

Серверна програма налічує три головні функції:

- *FormActivate();*
- *IdPOP3Server1RETR();*
- *IdPOP3Server1STAT();*

Ці функції обробляють повідомлення активації головної форми, запиту на отримання заданого повідомлення та запиту на отримання кількості повідомлень в поштовій скринці відповідно до протоколу *POP3* (елемент керування *TIdPOP3Server*).

Алгоритм роботи функції *FormActivate()* наведено на рис 18.1.

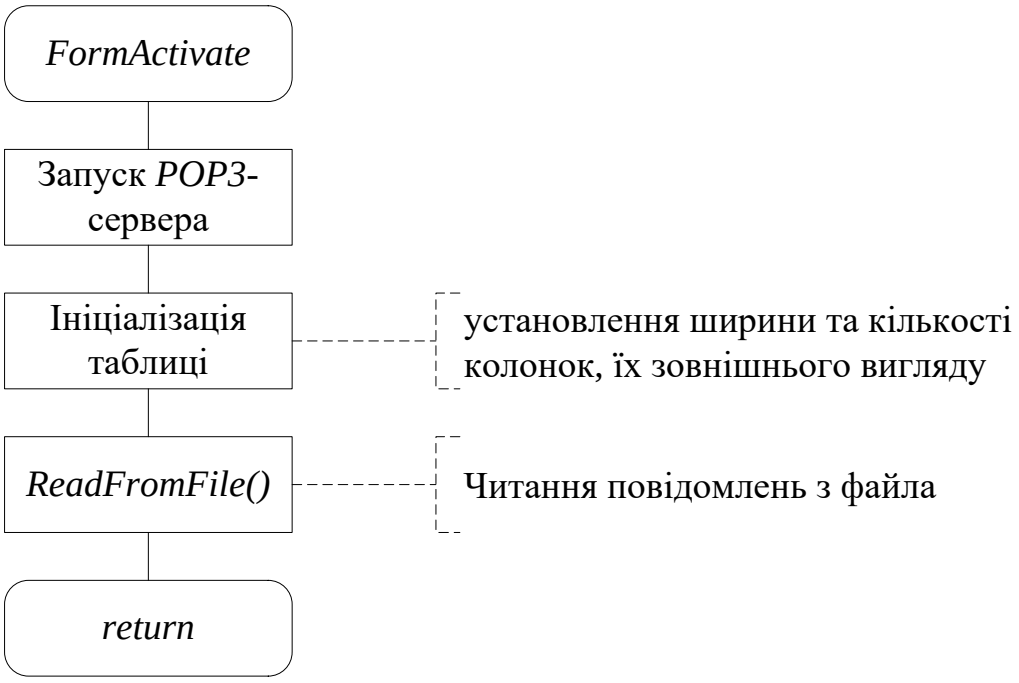

Рисунок 18.1 – Алгоритм роботи функції *FormActivate()*

Алгоритм роботи функції *IdPOP3Server1RETR()* наведено на рис. 18.2.

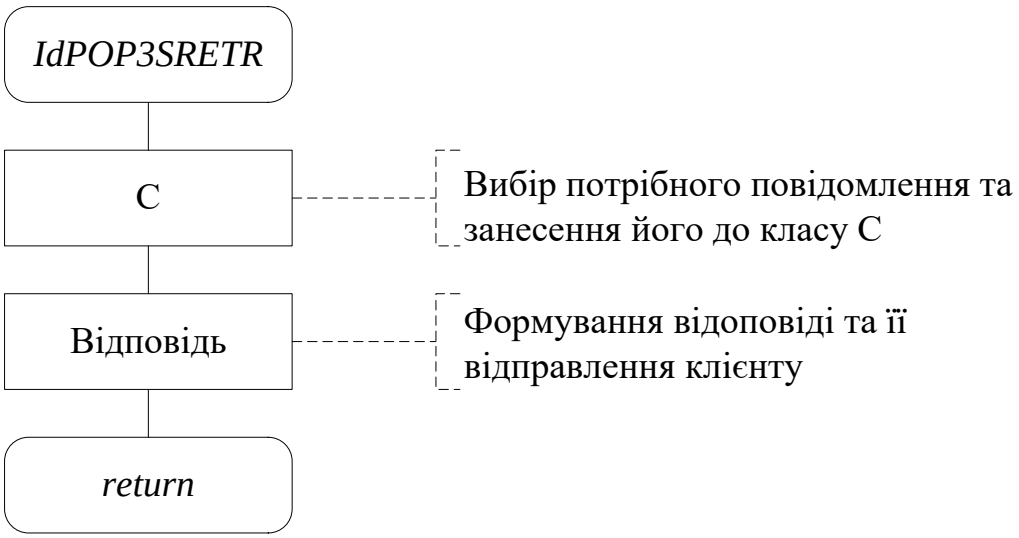

Рисунок 18.2 – Алгоритм роботи функції *IdPOP3Server1RETR()*

Алгоритм роботи функції *IdPOP3Server1STAT()* наведено на рис. 18.3.

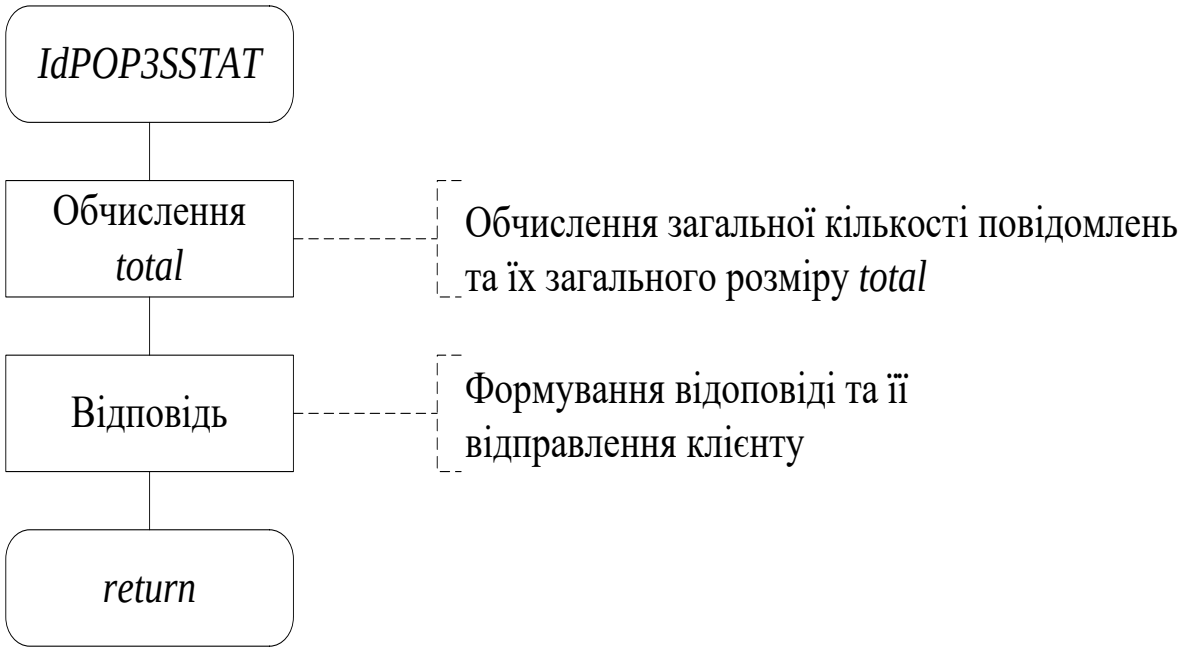

Рисунок 18.3 – Алгоритм роботи функції *IdPOP3Server1STAT()*

18.6 Структура програми з описом функцій складових частин і зв'язків між ними

Як вже було зазначено, програма складається з декількох функцій, три з яких є головними:

- Функція *FormActivate()* – обробник повідомлення активації головного вікна додатка, в якому запускається сервер *POP3*, що поданий екземпляром класу *TIdPop3Server*. Це означає, що сервер *POP3* переходить в активний стан, тобто очікує на підключення клієнтів (за замовчуванням номер порту – 110).

Після цього ініціалізується та заповнюється таблиця і список *Messages* даними з файла *mail_info.dat*, що містить всі листи, збережені на сервері.

Опис роботи процедури *TfrmMain.FormActivate(Sender: TObject):*

1. Встановлюємо прослуховування *POP3*-серверу, тобто очікуємо на підключення клієнтів:

IdPop3Server1.Active:=True;

editPort.Text:=IntToStr(IdPOP3Server1.DefaultPort);

2. Ініціалізація зовнішнього вигляду таблиці:

sGridTable.Cells[0,0]:='№';

sGridTable.Cells[1,0]:='От кого';

sGridTable.Cells[2,0]:= 'Кому';

sGridTable.Cells[3,0]:='Тема';

sGridTable.Cells[4,0]:= 'Текст сообщения';

sGridTable.Cells[5,0]:= 'Длина';

3. Задання ширини стовбців:

*sGridTable.ColWidths[0]:=*25;

*sGridTable.ColWidths[3]:=*110;

*sGridTable.ColWidths[4]:=*300;

*sGridTable.ColWidths[5]:=*43;

4. Читання повідомлень з фала:

ReadFromFile('mail_info.dat');

end;

- Функція *IdPOP3Server1RETR()* – обробник команди *RETR* елемента керування *IdPop3Server1*, що викликається коли клієнтська програма запитує конкретне повідомлення (лист), вказуючи його номер. У цій функції реалізується відправлення листа та його розміру відповідно до формату протоколу *POP3*.

Опис процедури *TfrmMain.IdPOP3Server1RETR (ASender: TIdCommand; AMessageNum: Integer)*:

1. Отримання повідомлення з номером *AMessageNum:*

C:=Messages.Items[AMessageNum-1];

2. Відправлення розміру повідомлення клієнту:

ASender.Thread.Connection.WriteLn('+OK '+IntToStr(C.msg.Len)+' octets');

3. Відправлення тексту адресату (від кого лист):

ASender.Thread.Connection.WriteLn('From: '+C.msg.FromMail+' ');

ASender.Thread.Connection.WriteLn('To: '+C.msg.ToMail+' ');

4. Відправлення теми листа:

ASender.Thread.Connection.WriteLn('Subject: '+C.msg.Subject+' ');

5. Відправлення самого повідомлення (листа):

ASender.Thread.Connection.WriteLn('');

ASender.Thread.Connection.WriteLn(C.msg.BodyText);

ASender.Thread.Connection.WriteLn('.');

end;

- Функція *IdPOP3Server1STAT()* – обробник команди *STAT* елемента керування *IdPop3Server1*, що викликається, коли клієнтська програма запитує стан поштової скриньки. В тілі цієї функції формується відповідь клієнту, в якій міститься кількість повідомлень у поштовій скринці та їх загальний розмір у символах.

Опис процедури *TfrmMain.IdPOP3Server1STAT (ASender: TIdCommand)*:

1. Підрахунок загального розміру всіх повідомлень, що збережені на *POP3*-сервері:

total:=0;

for i := 0 to Messages.Count - 1 do

begin

```
  C:=Messages.Items[i];
  total:=total+C.msg.Len;
 end;
```

2. Відправлення клієнту розміру - *total:*

```
 ASender.Thread.Connection.WriteLn('+OK
'+IntToStr(Messages.Count)+' '+IntToStr(total));
end;
```

Також треба зазначити, що всі повідомлення зберігаються в списку *Messages*, об'явленому як:

Messages : TList; Список усіх повідомленьсообщений

Повідомленя мають формат, що відповідає структурі:

TMailBoxEntry = packed record

FromMail, ToMail: string[20];	адреси джерел та відправників
Subject : string[20];	тема листа
BodyText : string[200];	тіло листа (повідомлення)
Len : integer;	довжина листа (повідомлення)

Алгоритм програми-клієнт

Клієнтська програма складається з декількох функцій, але як основну можна виділити лише одну – обробник повідомлення натискання кнопки *buttonConnect*. Схема алгоритму даної функції наведена на рис. 18.4.

Для роботи за протоколом *POP3* використовується елемент *IdPOP3Client*, що поданий у програмі екземпляром *POP*. Саме через нього здійснюється обмін даними між клієнтською та серверною програмою. При натисканні на кнопку "Соединение" викликається обробник *buttonConnect()*, в якому виконуються такі дії:

1) підключення до серверу;
2) отримання кількості повідомлень, що зберігаються на сервері;
3) циклічний запит усіх повідомлень з обчисленням часу приймання/передачі;
4) відображення отриманих даних;
5) від'єднання від серверу.


buttonConnect
Connected
Перевірка підключення до POP3-серверу
POP.Disconnect
Відключення від POP3-серверу
QueryPerformanceFrequency
Отримання частоти роботи таймерf- hfreq
Ініціалізація POP-клієнту
Заповнення парметрів з'єднання: ім'я та порт серверf
POP.Connect
Підключення до POP3-серверf
!Connected
Перевірка підключення до POP3-серверf
return
Отримання загальної кількості повідомлень на сервері
k=1; k<=i; i++
time1
Отримання часу початку запиту - time1
POP.Retrieve (k,IdMessage)
Запит на отримання повідомлення з номером k
time2
Отримання часу отримання відповіді - time2
Обчислення часу прийому/передачі
A
B
C

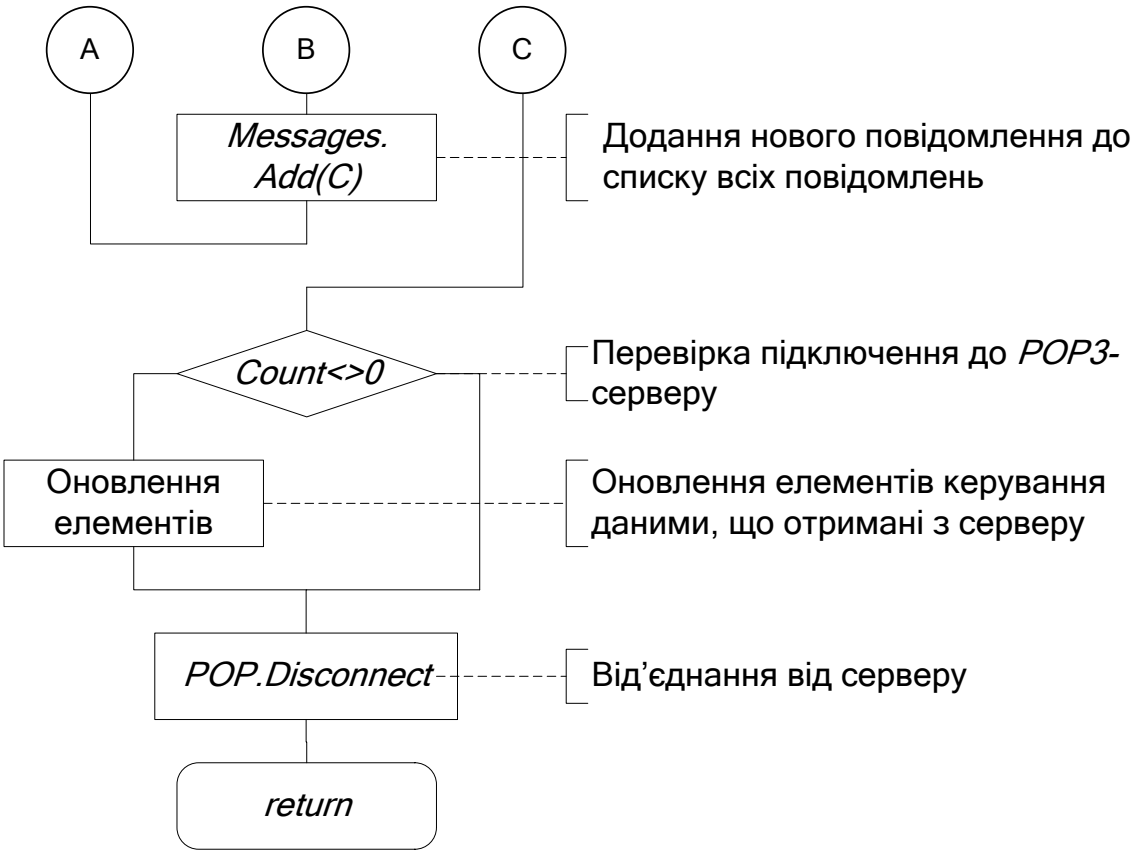

Рисунок 18.4 – Алгоритм роботи функції buttonConnect()

Опис процедури *TMainForm.buttonConnectClick(Sender: TObject)*:

1. Перевірка підключення клієнта:

```
if POP.Connected then
begin
   Відключення від серверу
  POP.Disconnect;
end;
```

2. Отримання частоти лічильника (hfreq – частота в секундах)

```
QueryPerformanceFrequency(hfreq);
```

3. Задання параметрів підключення:

<им'я серевера>, <№ порту>

```
POP.Host := editServerHost.Text;  'localhost';
POP.Port := StrToInt(editPort.Text);  110;
```

4. Підключення до POP3-сервера:

POP.Connect;

5. Якщо підключення пройшло вдало:

if POP.Connected then

begin

6. Очищення списку отриманих повідомлень:

Messages.Clear;

7. Одержуємо загальне число повідомлень у скринці – i:

i:=POP.CheckMessages;

labelTotalMsgs.Caption:=IntToStr(i);

8. Послідовно отримуємо всі повідомлення з скринки:

for k := 1 to i do

IdMessage.Clear;

9. Запам'ятовуємо початковий час – *time1:*

QueryPerformanceCounter(time1);

10. Одержуємо k-те повідомлення:

POP.Retrieve(k,IdMessage);

11. Запам'ятовуємо кінцевий час – *time2*:

QueryPerformanceCounter(time2);

12. Обчислення часу запиту/одержання повідомлення:

*rt:=((time2-time1)/hfreq)*1000;*

13. Занесення отриманої інформації до структури:

C:=TMsgClass.Create;

C.msg.FromMail:=IdMessage.From.Text;

C.msg.Subject:=IdMessage.Subject;

C.msg.BodyText:=IdMessage.Body.GetText;

C.msg.time:=rt;

14. Додавання повідомлення до списку:

Messages.Add(C);

15. Від'єднання від серверу:

POP.Disconnect;

Результати роботи програми

Програма *Server* запускається першою на тому ПК, зв'язок з якою ми перевіряємо з даного ПК, на якому буде запущена програма *Client*. Після її запуску з'являється форма, наведена на рис. 18.5.

Клієнтський додаток виконує такі функції:

- налагоджує зв'язок з *POP3*-сервером;
- організує отримання листів з сервера та їх перегляд;
- обчислює та відображує час прийому/передачі повідомлення;
- функціонує за протоколом *POP3*.

Головне вікно програми-клієнта наведено на рис. 18.6.

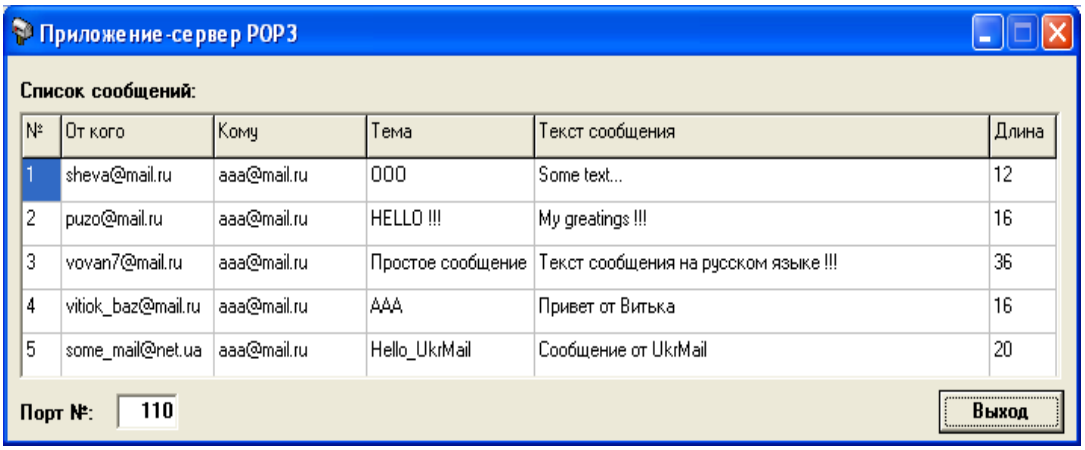

Рисунок 18.5– Головне вікно програми-серверу

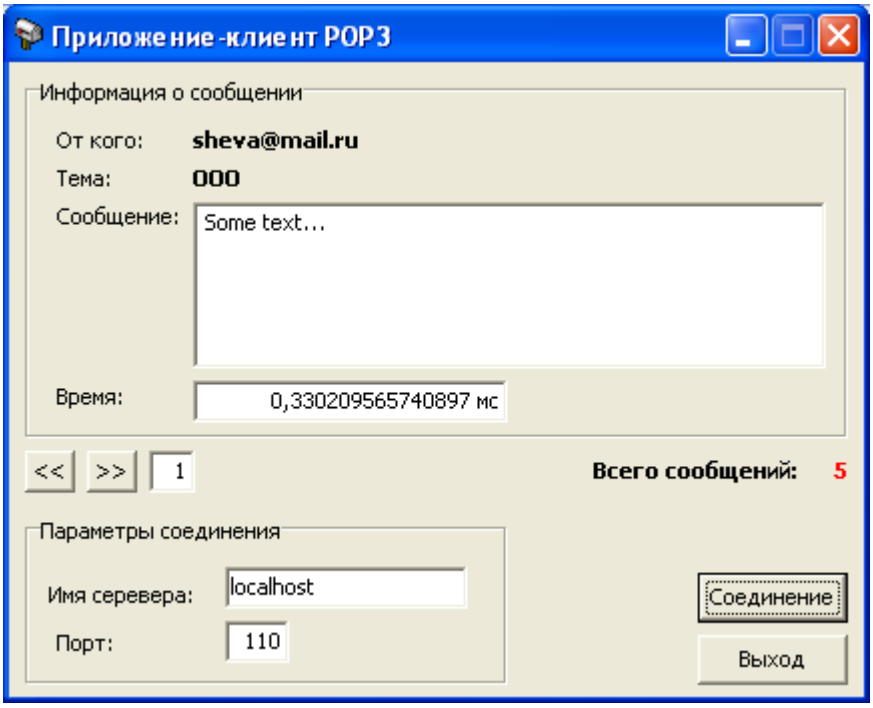

Рисунок 18.6 – Головне вікно програми-клієнта

Таким чином, ми маємо працездатну систему тестування мережі з визначенням її працездатності та часу проходження пакета.

Контрольні запитання

1. Які особливості протоколу *WHOIS?*
2. Де в основному використовується протокол *WHOIS* ?
3. Якими бувають бази даних, що мають *WHOIS*-інтерфейс?
4. Чи передбачено у протоколі *WHOIS* розрізнення централізованої і розподіленої моделей?
5. З чого складаються команди *POP3*?
6. Чим закінчуються команди *POP3?*

19 ПРОЄКТУВАННЯ СИСТЕМИ ТЕСТУВАННЯ МЕРЕЖІ З ПРОТОКОЛАМИ *TCP* ТА *FTP* З ВИЗНАЧЕННЯМ ЧАСУ ПОДВІЙНОГО ПРОХОДУ

19.1 Механізм сокетів

Для забезпечення мережних комунікацій використовують сокети. Сокет – це кінцева точка мережних комунікацій. Кожен, хто використовує сонет, має тип і асоційований з ним процес. Сокети існують всередині комунікаційних доменів. Домени – це абстракції, які мають конкретну структуру адресації і безліч протоколів, що визначають різні типи сокетів усередині домену. Прикладами комунікаційних доменів можуть бути: *UNIX*-домен, *Internet*-домен і т.д.

В *Internet*-домені сокет – це комбінація *IP*-адреси і номера порту, що однозначно визначає окремий мережний процес у всій глобальній мережі *Internet*. Два сокети, один для хоста-отримувача, інший для хоста-відправника, визначають з'єднання для протоколів, орієнтованих на встановлення зв'язку, таких, як *TCP*.

Для створення сокета використовується системний виклик *socket*.

s = socket(domain, type, protocol);

Цей виклик ґрунтується на інформації про комунікаційні домени і типи сокета. Для використання особливостей *Internet* значення параметрів повинні бути такими:

communication domain – *AF_INET* (*Internet* протоколи).

type of the socket – *SOCK_STREAM*; Цей тип забезпечує послідовний, надійний, орієнтований на встановлення двостороннього зв'язку потік байтів.

Нижче наведено короткий опис інших типів сокетов:

Datagramm socket – підтримує двосторонній потік даних. Не гарантується, що цей потік буде послідовним, надійним і що дані не будуть дублюватися. Важливою характеристикою даного сокета є те, що границі запису даних визначені.

Raw socket – забезпечує можливість користувальницького доступу до комунікаційних протоколів, які розташовані нижче та підтримують сокет-абстракції. Такі сокети звичайно є дейтаграмоорієнтованими.

Функція *socket* створює кінцеву точку для комунікацій і повертає файловий дескриптор, що посилається на сокет або "–1" у випадку помилки. Даний дескриптор використовується надалі для встановлення зв'язку.

Для створення сокета типу *stream* із протоколом *TCP*, що забезпечує комунікаційну підтримку, виклик функції *socket* повинен бути таким:

s = socket(AF_INET, SOCK_STREAM, 0);

19.2 Прив’язка до локальних імен

Сокет створюється без імені. Поки із сокетом не буде пов’язане ім’я, вилучені процеси не мають можливості посилатися на нього і, отже, на даному сокеті не може бути отримано ніяких повідомлень. Комунікаційні процеси використовують для даних цілей асоціації. В *Internet*-домені асоціація складається з локальної вилученої адреси і з локального вилученого порту. У більшості доменів асоціація повинна бути унікальною.

В *Internet*-домені зв’язування сокета і імені може бути досить складним, але, на щастя, звичайно немає необхідності спеціально прив’язувати адресу і номер порту до сокета, тому що функції *connect* і *send* автоматично зв’яжуть даний сокет з підходящою адресою, якщо це не було зроблено до їхнього виклику.

Для зв’язування сокета з адресою і номером порту використовують системний виклик *bind*:

bind(s, name, namelen);

Ім’я *name* – це рядок байтів змінної довжини, що інтерпретується підтримуваним протоколом. Інтерпретація може розрізнятися в різних комунікаційних доменах.

19.3 Встановлення зв’язку

З боку клієнта зв’язок встановлюється за допомогою стандартної функції

connect:

error = connect(s, serveraddr, serveraddrlen);

яка ініціює встановлення зв’язку на сокеті, використовуючи дескриптор сокета *s* і інформацію зі структури *serveraddr*, що має тип *sockaddr_in* та містить адресу серверу і номер порту, з яким треба встановити зв’язок. Якщо сокет не був пов’язаний з адресою, *connect* автоматично викличе системну функцію *bind*.

Connect повертає 0, якщо виклик пройшов успішно. Повернута величина «–1» вказує на те, що в процесі встановлення зв’язку відбулася якась помилка. У випадку успішного виклику функції процес може працювати з дескриптором сокета, використовуючи функції *read* і *write*, і закривати канал, використовуючи функцію *close*.

З боку серверу процес встановлення зв’язку є складнішим.

Коли сервер бажає запропонувати один зі своїх сервісів, він зв’язує сокет з загальновідомою адресою, що асоціюється з даним сервісом, і пасивно слухає цей сокет. Для цих цілей використовується системний виклик *listen*:

error = listen(s, qlength);

де ***s*** – це дескриптор сокета, а ***qlength*** – це максимальна кількість запитів на встановлення зв'язку, які можуть стояти в черзі, очікуючи обробки сервером; ця кількість може бути обмеженою особливостями системи.

Коли сервер одержує запит від клієнта і приймає рішення про встановлення зв'язку, він створює новий сокет і зв'язує його з асоціацією. Для *Internet*-домену це означає той же самий номер порту. Для цієї мети використається системний виклик *accept*:

newsock = accept(s, clientaddr, clientaddrlen);

Сокет, асоційований клієнтом, і сокет, що був повернутий функцією *accept*, використовуються для встановлення зв'язку між сервером і клієнтом.

19.4 Передача даних Закривання сокетів

Коли зв'язок встановлений за допомогою різних функцій, може початися процес передачі даних. За наявності зв'язку користувач може посилати і одержувати повідомлення за допомогою функцій *read* і *write*:

write(s, buf, sizeof(buf));
read(s, buf, sizeof(buf));

Виклики *send i recv* практично ідентичні *read* і *write*, за винятком того, що додається аргумент прапорів.

send(s, buf, sizeof(buf), flags);
recv(s, buf, sizeof(buf), flags);

Можуть бути зазначені один або більше прапорів за допомогою ненульових значень, таких, як наступні:

- ***MSG_OOB*** – посилати/одержувати дані, характерні для сокетів типу *stream*.
- ***MSG_PEEK*** – переглядати дані без читання. Якщо використовується у функції *recv*, то будь-які присутні дані повертаються користувачеві, але самі дані залишаються як "непрочитані". Наступний *read* або *recv*, викликаний на даному сокеті, поверне прочитані минулого разу дані.
- ***MSG_DONTROUTE*** – посилати дані без маршрутизації пакетів. (Використовується тільки процесами, що управляють таблицями маршрутизації.

19.5 Закривання сокетів

Коли взаємодіючі модулі вирішують припинити передачу даних і закрити сеанс зв'язку, вони обмінюються тристороннім рукостисканням з сегментами, що містять установлений біт "Від відправника більше немає даних" (цей біт ще називається *FIN* біт).

Якщо сокет більше не використовується, процес може закрити його за допомогою функції *close*, викликавши її з відповідним дескриптором сокета:

close(s);

Якщо дані були асоційовані із сокетом, що обіцяє доставку (сокет типу *stream*), система буде намагатися здійснити передачу цих даних. Тим не менш, після закінчення досить таки тривалого проміжку часу, якщо дані усе ще не доставлені, вони будуть відкинуті. Якщо користувальницький процес бажає припинити будь-яку передачу даних, він може зробити це за допомогою виклику *shutdown* на даному сокеті для його закриття. Виклик *shutdown* викликає "моментальне" відкидання всіх даних по черзі. Формат виклику такий:

shutdown(s, how);

де *how* має одне з наступних значень:

- 0 – якщо користувач більше не бажає читати дані;
- 1 – якщо дані більше не будуть посилати;
- 2 – якщо дані не будуть посилати та не будуть читати.

19.6 Проектування програм тестування з протоколом *TCP*

Проектування системи виконується за допомогою трьох бібліотек *WinSock2, Indy, ICS.*

Алгоритми тестування клієнт-серверної системи з бібліотекою *Winsock2*

Алгоритм роботи програми-серверу, що використовує бібліотеку WinSocket:

1. Ініціалізация бібліотеки роботи з сокетами:
   WSA_Startup;
2. Перевірка на помилки:
   GetLastError = 0;
3. Створення сокета:
   s = socket(domain, type, protocol);
4. Зв’язуваня сокета з адресою і портом:
   bind(s, name, namelen);
5. Ініціалізация серверного сокета і очікування клієнта:
   error = listen(s, qlength);
6. Після встановлення зв’язку с клієнтом, робота с ним:
   newsock = accept(s, clientaddr, clientaddrlen);
7. Ініціалізація циклу приймання повідомлення від клієнтського сокета.
8. Закриття сокета:
   Close;

19.7 Алгоритм роботи програми клієнта, що використовує бібліотеку *WinSocket*

1. Ініціалізація бібліотеки роботи з сокетами:
   WSA_Startup;
2. Перевірка на помилки:
   GetLastError = 0;
3. Створення сокета:
   s = socket(domain, type, protocol);
4. Зв'язування сокета з адресою і портом:
   bind(s, name, namelen);
5. З'єднання з серверним сокетом:
   error = connect(s, serveraddr, serveraddrlen);
6. Ініціалізація циклу приймання повідомлення від серверного сокета;
7. Закриття сокета:
   Close;

19.8 Алгоритми тестування клієнт-серверної системи з бібліотекою *ICS*

Алгоритм роботи серверу ICS:

1. Запускаємо сервер:

```
WSocket_gethostname(strHostName);
Display('Server: I am "' + strHostName + '"');
Display('Server: IP: ' + LocalIPList.Text);
WSocketServer.Proto      := 'tcp';              { Use TCP protocol  }
WSocketServer.Port       := '1111';{'telnet';}    { Use telnet port   }
WSocketServer.Addr       := LocalIPList.Text;    { Use any interface }
WSocketServer.ClientClass := TTcpSrvClient;      { Use our component }
WSocketServer.Listen;
```

2. В *OnClientConnect* оброблюємо підключення клієнта:

```
with Client as TTcpSrvClient do begin
    OnDataAvailable    := WSocketServerDataAvailable;
    OnDataSent         := WSocketServerDataSent;
end;
```

3. В *WSocketServerDataAvailable* оброблюємо отримання даних від клієнта і посилаємо у відповідь підтвердження:

```
with Sender as TWSocket do      begin
Display('Server: MESSAGE from Client ['+PeerAddr+']: '  +
ReceiveStr);
SendLine('accept request');  bWasAccepted := true;
ImServerIn.Picture.LoadFromFile(STATE_RECEIVE);
TimerServerIn.Enabled := true;      end;
```

4. Зупинення серверу.

Алгоритм роботы клієнта *ICS*:

1. Ініціалізація настроювань сокета:

```
WSocket.Proto      := 'tcp';                  { Use TCP protocol  }
WSocket.Port       := '1111'; {'telnet';}     { Use telnet port   }
WSocket.Addr       := LocalIPList.Text;
```

2. Зв'єднання:

```
WSocket.Connect;                              { Start litening    }
```

3. Посилаємо в циклі задане число пакетів;
4. Фіксування часу відправки;

В *WsocketDataAvailable* обробка отримання даних від серверу:

```
Display('Client: MESSAGE from Server: ' + WSocket.ReceiveStr);
acceptTime := FormTest.GetTimerValue;
```

4. Фіксування часу приймання;
5. Зупинка клієнта та закриття сокетів.

Команди для роботи з *ICS*.

Server:

1. *Property string Proto* – поле задання протоколу обміну;
2. *Property string Port* – поле задання порта сокета;
3. *Property string Addr* – поле задання адреси сокета;
4. *Function Listen* – функція перемикання сокета в режим очікування клієнтів;
5. *Procedure SendLine(str: string);* – посилання рядка сокету.

Client:

1. *Property string Host* – поле задання адреси серверу;
2. *Property string Port* – поле задання порту сокета;
3. Function Connect – функція перемикання сокета в режим з'єднання з сервером;
4. *Function ReceiveStr : string* – отримання рядка з черги сокета.

19.9 Алгоритми тестування клієнт-серверної системи з бібліотекою *Indy*

Алгоритм роботи серверу *Indy:*

1. Запуск серверу:

```
IdTCPServer.Bindings.Items[0].IP := strIP;
IdTCPServer.Bindings.Items[0].Port := 1111;
IdTCPServer.Active := true;
```

2. В *IdTCPServerConnect* обробка підключення клієнта:

```
Display('Server [STATUS]: Connected from: ' +
AThread.Connection.Socket.Binding.PeerIP);
iCount := iCount+1;
Display('Server: There is now ' + IntToStr(iCount) + ' clients
connected.');
```

3. В *IdTCPServerExecute* обробка отриманих даних від клієнта та відправка у відповідь підтвердження:

```
Try   str := AThread.Connection.ReadLn();
  except       on E: Exception do AThread.Connection.Disconnect;  end;
  if AThread.Connection.Connected then
  begin      Display('Server [MESSAGE]:  ' + str);
   AThread.Connection.WriteLn('accept request');
```

4. Зупинка серверу.

Алгоритм роботи клієнта *Indy*:

1. Ініціалізація налаштувань сокета:

```
strIP := LocalIPList.CommaText;
    if CheckClientIP.Checked then
      strIP       := IPEdit.Text;
    IdTCPClient.Host  := strIP;
    IdTCPClient.Port  := 1111;
```

2. З'єднання:

```
IdTCPClient.Connect;
```

3. Відправка у циклі заданого числа пакетів.
4. Фіксування часу відправки.
5. В *TClientThread.Execute* обробка отриманих даних від серверу:

```
strReceive := ClientSocket.ReadLn().
```

6. Фіксування часу приймання.
7. Зупинка клієнта, закриття сокета.

Команди для роботи з Indy.

1. *Property TAddrList Bindings* – поле задання списку адрес налаштувань сокета.
2. *Property boolean Active* – поле задання активності сокета. Після зміни параметра в *true* сокет переходить у режим активності.
3. *Procedure WriteLn (str: string);* – відправка рядка.
4. *Function ReadLn : string* – отримання рядка з черги сокета.

19.10 Результати тестування програми з *TCP*

Програма розроблена в середовищі *Delphi* і виконує тестування мережних компонентів для роботи з механізмом гнізд (*sockets*) та протоколів *FTP* та *TCP*.

Реалізовано тестування таких наборів компонентів:

- *ICS Sockets(FTP);*
- *WinSockets(FTP);*
- *Indy Sockets(FTP);*
- *ICS Sockets(TCP);*

- *WinSockets(TCP);*
- *Indy Sockets(TCP);*

19.11 Робота з програмою

Загружаємо файл *TestLoader.exe* та маємо вікно програми, де є дві кнопки «Протокол *TCP*» та «Протокол *FTP*» (рис. 19.1).

«Протокол *TCP*» – для роботи с протоколом *TCP*.

«Протокол *FTP*» – для роботи с протоколом *FTP*.

Рисунок 19.1 – Вікно селектора тесту

Для роботи з протоколом *TCP* потрібно натиснути відповідну кнопку, після чого з'явиться меню (рис. 19.2).

Рисунок 19.2 – Вікно селектора тесту

Позначення кнопок меню є таким:

- *Socket TCP* – виклик вікна, на якому розташовані елементи керування для здійснення тестування як Серверу так і Клієнта (за допомогою *TCP Socket*).
- *Socket ICS* – виклик вікна, на якому розташовані елементи керування для здійснення тестування як Серверу так і Клієнта (за допомогою *ICS Socket*).
- *Socket Indy* – виклик вікна, на якому розташовані елементи керування для здійснення тестування як Серверу так і Клієнта (за допомогою *Indy Socket*).

Кожне вікно має однакову структуру, тобто набір кнопок та їх призначення і розташування. Тому розглянемо детально призначення кожної кнопки та кожного вікна на прикладі вікна, яке з'являється при натисканні кнопки «*ICS Socket*» (рис. 19.3).

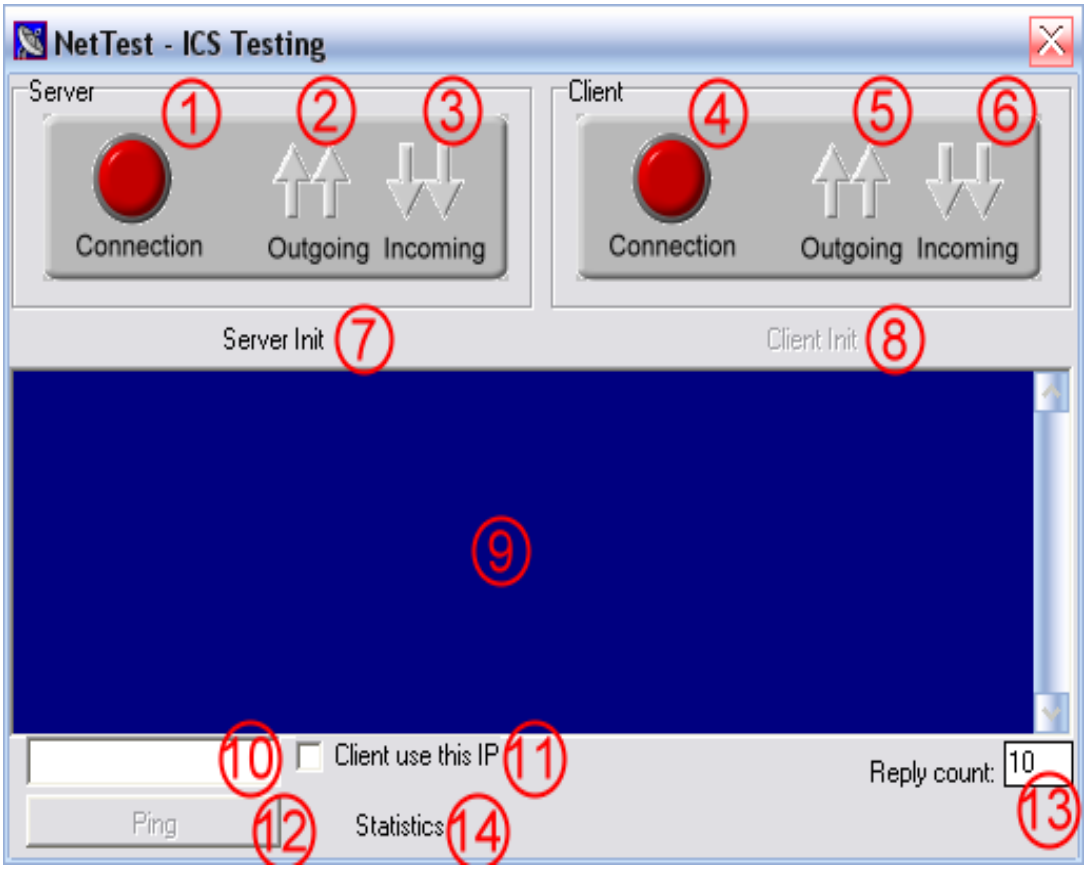

Рисунок 19.3 – Вікно тестування

На рисунок 19.3 розміщено 13 кнопок, значення яких наведено нижче.

1. Індикатор з'єднання з клієнтом.
2. Індикатор вихідного трафіку серверу.
3. Індикатор вхідного трафіку серверу.
4. Індикатор з'єднання з сервером.
5. Індикатор вихідного трафіку клієнта.

6. Індикатор вхідного трафіку клієнта.
7. Кнопка ініціалізації серверу.
8. Кнопка ініціалізації клієнта.
9. Поле ходу роботи.
10. Поле завдання віддаленої адреси серверу (для клієнта).
11. Вмикач використання віддаленої адреси.
12. Кнопка початку посилання пакетів на сервер.
13. Поле введення кількості посилань.
14. Кнопка виклику форми статистики.

Після запуску додатка, з'являється головна форма, де потрібно вибрати протокол тестування (рис. 19.2). Після цього маємо ще одне вікно, де подано селектор вибору діалогу тестування. Активуйте потрібну форму тестування шляхом натискання однієї з кнопок селектора. Інтерфейс форми тестування поданий на рис. 19.3.

Сценарій тестування 1 (без вказування віддаленої адреси серверу).

Натисніть кнопку *Server Init*. Після цього в полі ходу роботи має з'явитися текст:

Server: I am "XXXXX"
Server: IP: 172.18.16.149

Server: Waiting for clients...

Натисніть кнопку *Client Init*. Після цього відбудеться з'єднання каналів і у вікні порядку роботи з'явиться такий текст:

Client: Connected to server
Server: Client connected. Remote: 172.18.16.149/2034 *Local*: 172.18.16.149/1111
Server: There is now 1 clients connected.

Введіть потрібну кількість повторів відправки пакетів та натисніть кнопку *Ping*. Після чого тест перейде в активну фазу і результат походження пакетів можливо відслідковувати шляхом аналіза індикаторів вхідного та вихідного трафиків.

Сценарій тестування 2 (з визначенням віддаленого серверу).

Запустіть два екземпляри додатку на різних комп'ютерах. На одному з них ініціалізуйте серверну частину, на другом – клієнтську частину та введіть *IP*-адресу серверної машини и ставте перемикач використання віддаленої адреси в положення «ВКЛ». Натисніть кнопку *Ping* и аналізуйте результати тестів.

Примітка 1. Програма передбачає комбінування технологій. Тому використання однойменної технології необов'язково. Наприклад, для серверної части обрана реалізація *ICS*, а для клієнтської *Indy*. Це дає більш широкі можливості для тестування компонентів та мережі.

Примітка 2. Час обміну пакетами вимірюється залежно від частоти процесора, як найбільш швидкої та точної одиниці вимірювання подій комп'ютера. Тому результат поданий в умовних одиницях виміру.

19.11.1 Перегляд результатів тесту (статистика)

На вікні проведення тестування натисніть кнопку *Statistics*. у результаті з'явиться форма, подана на рис 19.4.

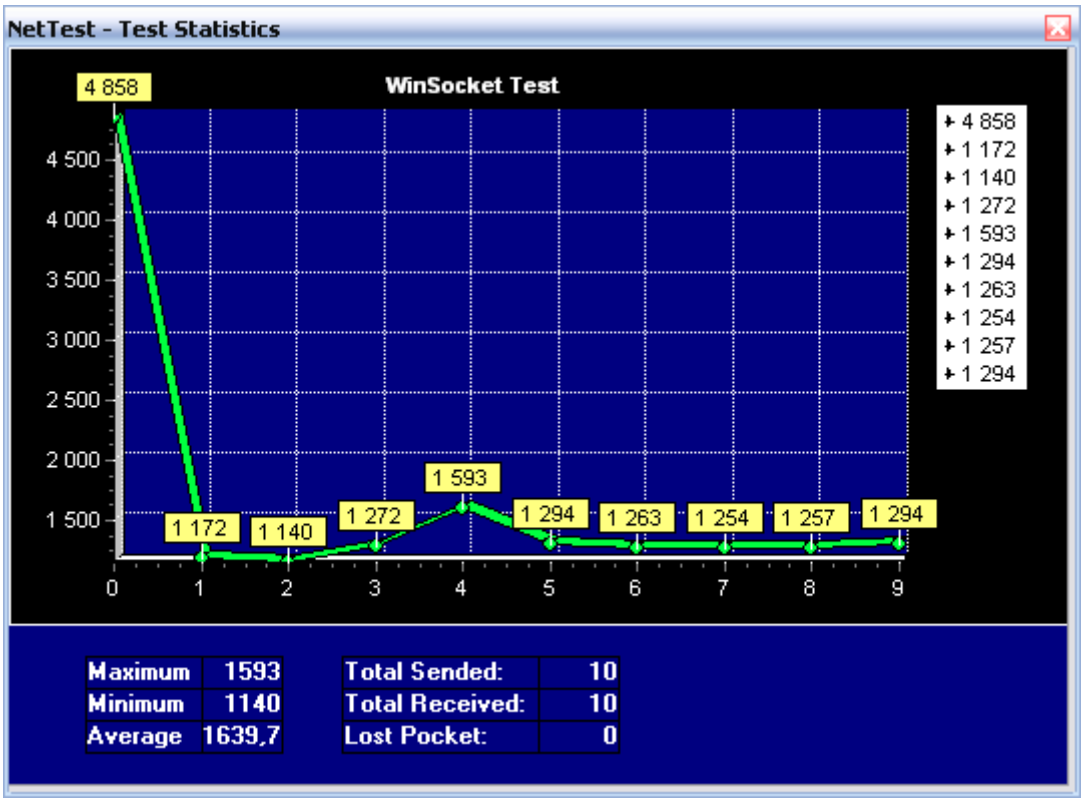

Рисунок 19.4 – Форма статистики

На формі подано дані за результатами тесту. Графік відображає динаміку затримки передавання пакета. Також виведені максимальні, мінімальні і середні значення.

19.11.2 Приклад роботи програми (у вигляді логу)

ICS Testing.
Server: I am "ANGEL"
Server: IP: 172.18.16.149

Server: Waiting for clients...
Client: Connected to server
Server: Client connected. Remote: 172.18.16.149/2034 *Local*: 172.18.16.149/1111
Server: There is now 1 clients connected.
Client: MESSAGE from Server: Welcome to OverByte ICS TcpSrv
==========1==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 606
==========2==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 534
==========3==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 556
==========4==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 518
==========5==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 517
==========6==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 520
==========7==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 653
==========8==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 486
==========9==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request
time := 488
==========10==========
Server: MESSAGE from Client [172.18.16.149]: post request
Client: MESSAGE from Server: accept request

time := 509

19.12 Проектування програм тестування з протоколом *FTP*

Програми клієнт та сервер з бібліотекою *ICS*

Алгоритм роботи програми-клієнта:

1. Виведення на екран вікна програми.
2. Визначення частоти процесору для подальшого використання при розрахунках.
3. Ініціалізація бібліотеки *ICS*.
4. Ініціалізація клієнт-компоненту *FTPClient*.

```
type
 TForm1 = class(TForm)
   FtpClient1: TFtpClient;
```

5. Указання адреси, ім'я користувача та пароля клієнта:

```
FtpClient.UserName      := 'Angel';
FtpClient.PassWord      := 'Pass111';
FtpClient.HostDirName   := HostPathEdit.Text;
FtpClient.HostFileName  := HostFileEdit.Text;
FtpClient.LocalFileName := LocalFileEdit.Text;
FtpClient.Binary        := true;
FtpClient.Passive       := false;
```

6. Початок вимірювання часу двома способами.
7. Відправка даних.
8. Закінчення вимірювання часу.
9. Виведення отриманих результатів.

Алгоритм роботи серверу:

1. Завантаження та виведення на екран вікна програми.
2. Ініціалізація бібліотеки сокетів:

```
WSAStartup($101, Data);
```

3. Переведення серверу в режим очікування:

```
FtpServer1.Start;
Logit('FTP запущен');
```

4. Приймання, розпізнавання та відповідні дії на вхідні дані.
5. Закриваємо бібліотеку сокетів:

```
WSACleanup;
```

Команди для роботи з *ICS*:

Server:

1. *Property string Port* – поле задання порту сокета.
2. *Procedure Start* – функція перемикання сокета в режим очікування клієнтів.
3. *Procedure Stop* – функція перемикання сокета в режим зупинки.
4. *Procedure DisconnectAll* – функція відключення всіх клієнтів.

Client:

1. *Property string UserName* – поле задання імені користувача.
2. *Property string PassWord* – поле задання пароля для входу на сервер.
3. *Property string HostDirName* – поле задання директорії серверу.
4. *Property string HostFileName* – поле задання файлу серверу.
5. *Property string LocalFileName* – поле задання файлу клієнта.
6. *Property string Binary* – поле задання двійкового формату передачі даних.
7. *Procedure ExecuteCmd(TCommand, TTransmitAsync)* – виконання команди, де *TCommand* – тип команди, *TTransmitAsync* – режим синхронізації при виконанні.

Програми клієнт та сервер з бібліотекою *Indy*

Алгоритм роботи клієнта:

1. Ініціалізація сокета:

IdFTP1.Host := EditHost.Text;

2. Виконання з'єднання:

IdFTP1.Connect();

3. Фіксування часу відправлення.
4. Посилання файлу на сервер:

IdFtp1.Put(strFile, strDest);

5. В *IdFTP1WorkEnd* виконується обробка завершення трансферу.
6. Фіксація часу завершення відправлення.

Алгоритм роботи серверу:

1. Заповнення поля конфігураційного компонента.
2. Переведення до активного стану.
3. Обробка запитів клієнтів. Отримання вмісту директорій.

3.1. Команди на отримання файлів:

```
procedure TForm1.IdFTPServerRetrieveFile(ASender:
TIdFTPServerThread;
  const AFileName: String; var VStream: TStream);
begin
   VStream :=
TFileStream.Create('C:\Temp\'+AFileName,fmOpenRead);
end;
```

3.2. Команди на завантаження файлів:

procedure TForm1.IdFTPServerStoreFile(ASender: TIdFTPServerThread;
const AFileName: String; AAppend: Boolean; var VStream: TStream);
begin
VStream := TFileStream.Create(AFileName,fmCreate);
end;

4. Зупинка серверу.

Команди для роботи з *Indy*

1. *Property string Host* – поле задання адреси серверного сонета.
2. *Procedure Connect();* – з'єднання із сервером.
3. *Procedure Put(DestFile: string; SourceFile: string);* – посилання файлу на сервер, де *DestFile* – рядок назви файлу для приймання, *SourceFile* – рядок назви файлу для відправлення.

Результати тестування програми з FTP

Для роботи з протоколом *FTP* потрібно натиснути відповідну кнопку, після чого з'явиться меню (рис. 19.5).

Позначення кожного пункту меню:

- *Server FTP (ICS)* – виклик вікна, на якому розташовані елементи керування для здійснення тестування Серверу (за допомогою *ICS FTP Socket*);
- *Client FTP (ICS)* – виклик вікна, на якому розташовані елементи керування для здійснення тестування Клієнта (за допомогою *ICS FTP Socket*);
- *Server FTP (Indy)* – виклик вікна, на якому розташовані елементи керування для здійснення тестування Серверу (за допомогою *Indy FTP Socket*);
- *Client FTP (Indy)* – виклик вікна, на якому розташовані елементи керування для здійснення тестування Клієнта (за допомогою *Indy FTP Socket*);
- *Client - Server(WinSocket)* – виклик вікна, на якому розташовані елементи керування для здійснення тестування як Серверу так і Клієнта(за допомогою *WinSocket FTP*).

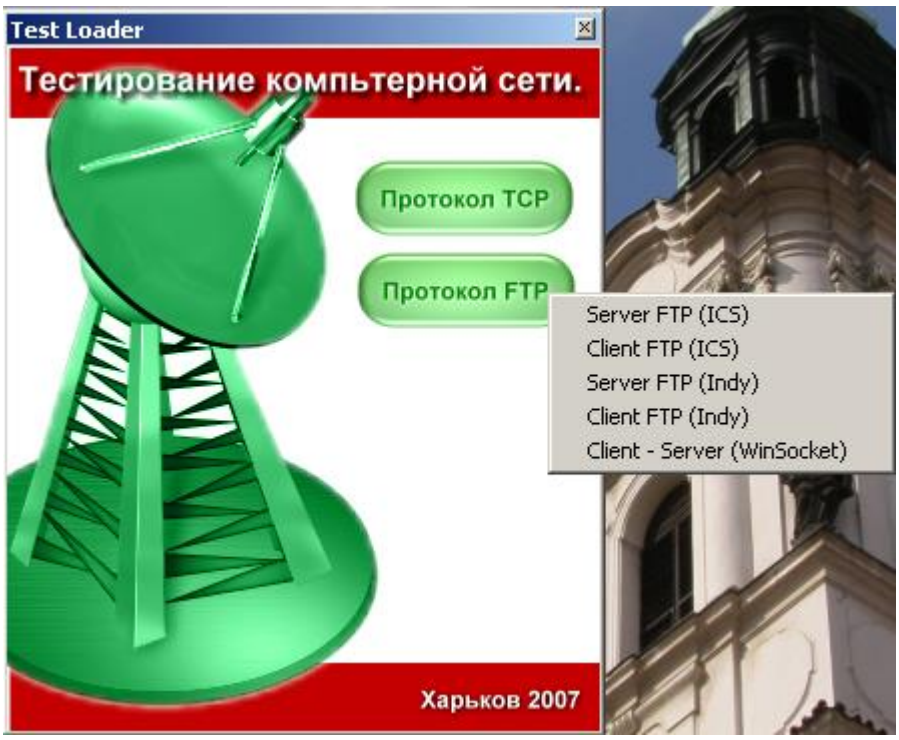

Рисунок 19.5 – Вікно селектора тесту

При натисканні *Server FTP (ICS)* з'явиться вікно (рис 19.6).

Сервер приймає команди клієнтів та виконує їх. Протокол роботи серверу відображується на синьому полі виведенні ходу роботи. Для запуску роботи серверу потрібно вибрати *File* → *Start Server*.

Для запуску роботи програми-клієнта, натисніть у вікні (рис 19.5) пункт меню *Client FTP (ICS)*. При натисканні *Client FTP (ICS)* з'явиться вікно (рис . 19.7).

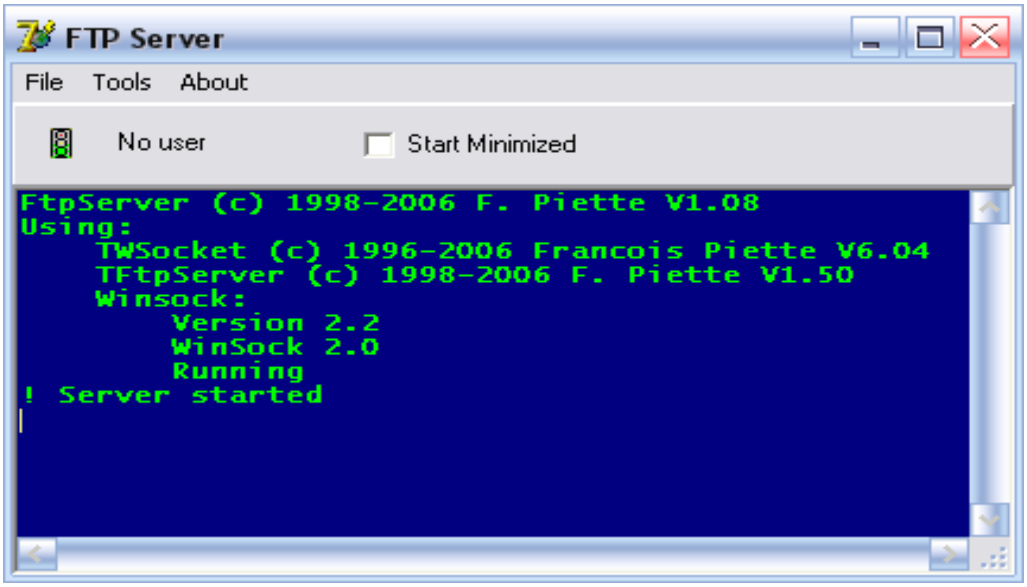

Рисунок 19.6 – Вікно тестування програми-серверу

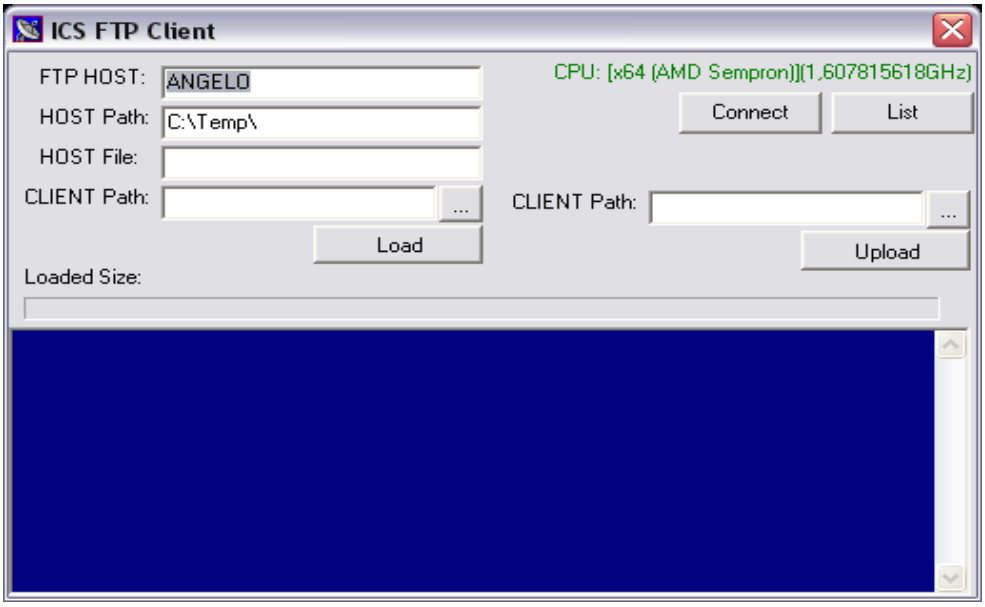

Рисунок 19.7 – Вікно тестування кліента (ICS)

Порядок проведення тестування є таким:

- для з’єднання з сервером треба натиснути «*Connect*»;
- для отримання списку файлів, розташованих на сервері, треба натиснути кнопку «*List*»;
- в рядок «*Host File*» потрібно ввести ім’я файла для завантаження;
- в рядок «*CLIENT Path*» (ліва частина вікна) потрібно ввести шлях збереження завантаженого файла;
- потрібно натиснути «*Load*» для початку завантаження;
- для завантаження файлів на Сервер потрібно обрати файл за допомогою рядка «*CLIENT Path*» (права частина вікна) та натиснути кнопку «*Upload*».

При натисканні *Server FTP (Indy)* у вікні (рис. 19.5) з’явиться вікно (рис. 19.6).

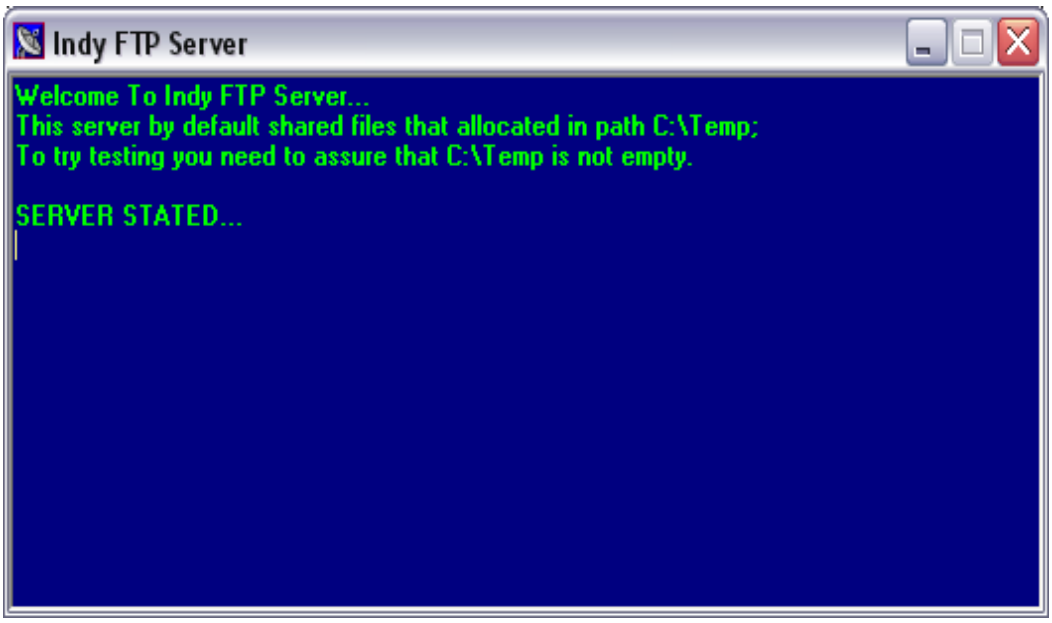

Рисунок 19.8 – Вікно тестування програми серверу

Сервер стартує відразу після запуску програми. Тепер потрібно запустити програму-клієнт, шляхом натискання у вікні (рис. 19.5) пункту меню *Client FTP (Indy)*. При натисканні *Client FTP (Indy)* з'явиться вікно (рис. 19.9).

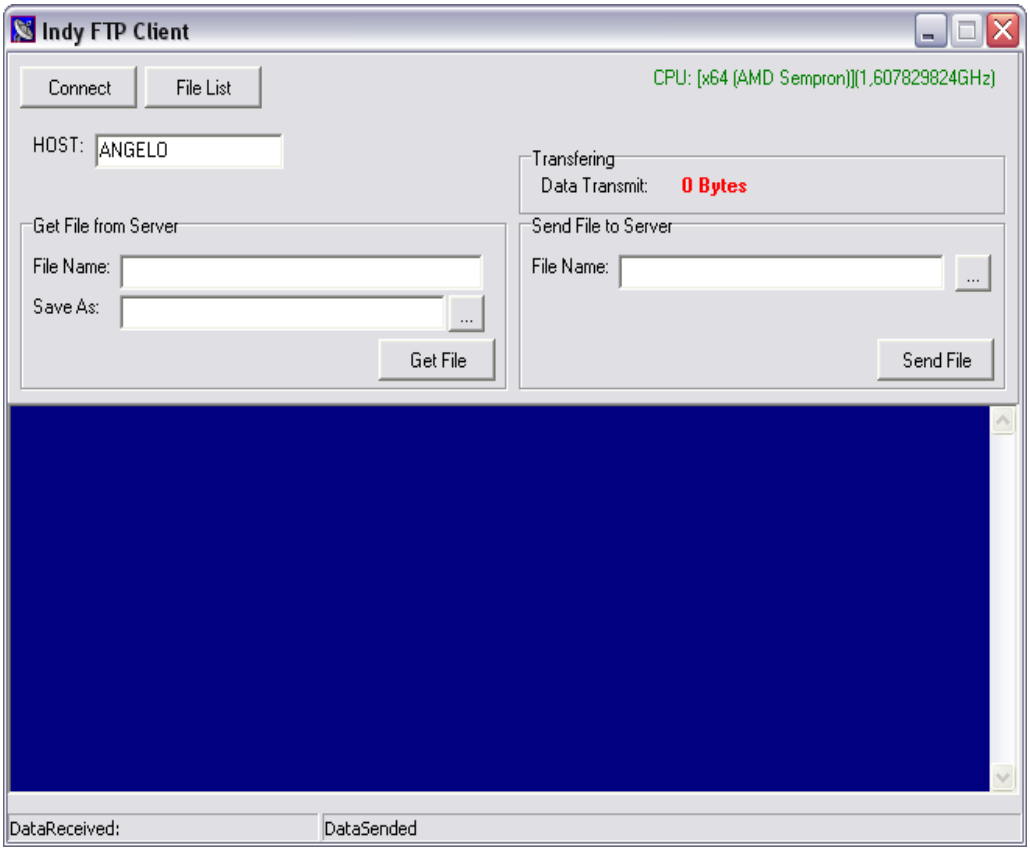

Рисунок 19.9 – Вікно тестування клієнта (*Indy*)

Порядок проведення тестування є таким:

- для з'єднання з сервером потрібно натиснути «*Connect*»;
- для отримання списку файлів, розташованих на сервері, потрібно натиснути кнопку *«File List»*;
- в рядок *«File Name»* потрібно ввести ім'я файла для завантаження;
- в рядок *«Save as»* (ліва частина вікна) потрібно ввести шлях збереження завантаженого файлу;
- потрібно натиснути *«Get File»* для початку завантаження;
- для завантаження файлів на Сервер необхідно обрати файл за допомогою рядка *«File Name»* (права частина вікна) та натиснути кнопку *«Send File»*.

Якщо потрібно протестувати мережу за допомогою *WinSocket FTP* (рис 19.10) – необхідно викликати вікно, на якому розташовані елементи керування для здійснення тестування як Серверу так і Клієнта (за допомогою *«Client - Server(WinSocket)»*).

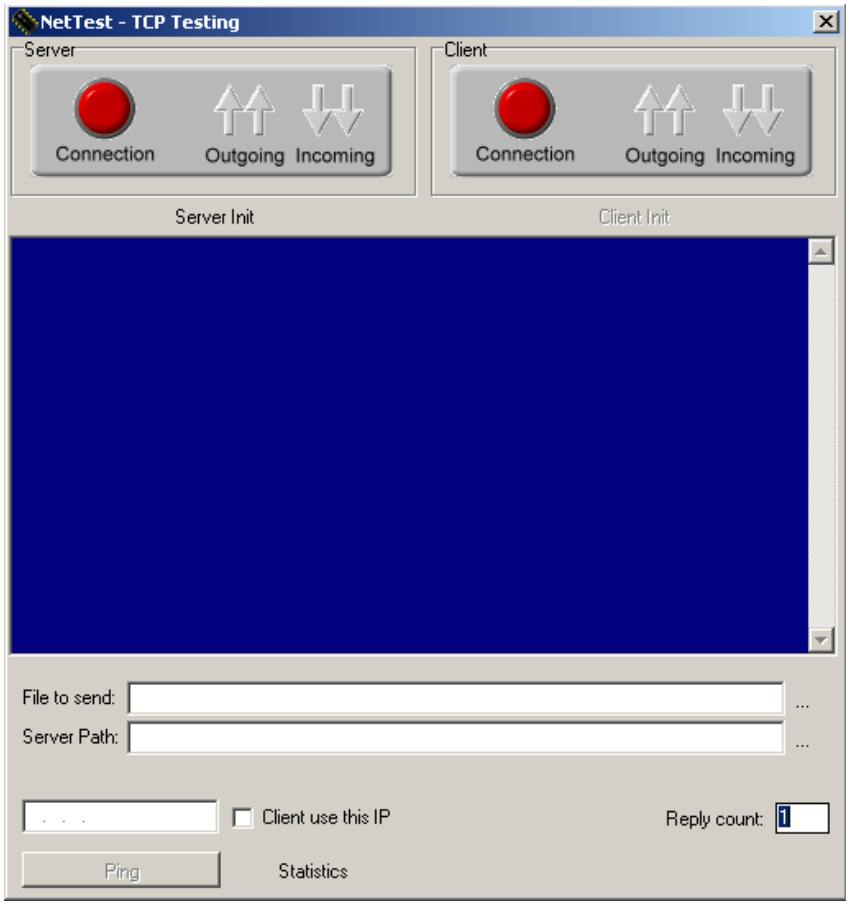

Рисунок 19.10 – Вікно тестування клієнта (WinSocket FTP)

Спочатку треба ініціалізувати роботу Серверу, потім Клієнта (потрібно натиснути відповідні кнопки), після чого з'явиться повідомлення типу:

TCPServer : wsaStartup OK
TCPServer : Server socket creation successfuly.
TCPServer : Binding socket... - Server [ANGELO.bbk] *(*172.18.16.149:1111*) ready*

TCP_Client : connection established
TCP_Server : client accepted

Потім потрібно вказати файл, який потрібно передавати (шлях до нього вказується за допомогою кнопки «…» справа від рядка введення), та шлях на Сервері, куди він завантажиться після копіювання. Для здійснення передавання потрібно натиснути кнопку *«Ping»*.

Контрольні запитання

1. Що використовують для забезпечення мережних комунікацій ?
2. Що таке сокет?
3. Де існують сокети?
4. Чи мають можливість вилучені процеси посилатися на сокет без імені?
5. Який системний виклик використовують для зв’язування сокета з адресою і номером порту?

20 ПРОЄКТУВАННЯ ЗАСОБІВ ДІАГНОСТИКИ З БІБЛІОТЕКАМИ *WINSOCK, INDY, ICS*

20.1 Проектування тестувальних засобів мереж

TCP (*Transmission Control Protocol*) – це протокол, який дозволяє прикладним програмам, запущеним на різних комп'ютерах мережі, обмінюватися потоками даних. TCP ділить потоки даних на ланцюжки, які називаються *TCP*-сегментами, і передає їх за допомогою *IP*. В більшості випадків кожен *TCP*-сегмент пересилається в одній *IP*-дейтаграмі. Проте за необхідності *TCP* розщеплюватиме сегменти на декілька *IP*-дейтаграм, що вміщаються у фізичні кадри даних, які використовують для передачі інформації між комп'ютерами в мережі. Оскільки *IP* не гарантує, що дейтаграми будуть отримані в тій же самій послідовності, в якій вони були послані, *TCP* здійснює повторне "збирання" *TCP*-сегментів на іншому кінці маршруту, щоб утворити безперервний потік даних. *FTP і telnet* - це два приклади популярних прикладних програм *TCP/IP*, які спираються на використання *TCP*.

Розглянемо можливості бібліотек.

Опис функцій WinSock2.

Функції бібліотеки *WinSock* призначені для керування мережею на низькому рівні. Це досягається завдяки роботі безпосередньо із сокетами.

Функції бібліотеки:

1. Ініціалізація *WinSock.*

WSAStartup(wVersionRequired: word; var WSData: TWSAData): Integer;

2. Створення сокету.

function socket(const af, struct, protocol: Integer): TSocket;

3. Прив'язка адреси до сокета.

function bind(const s:TSocket; const addr: PSockAddr; const namelen:Integer): Integer;

4. Прослуховування порту.

function listen(s: TSocket; backlog: Integer): Integer;

5. Підтвердження з'єднання (сторона серверу).

function accept(const s: TSocket; var addr: TSockAddr; var addrlen: Integer): TSocket;

6. Встановлення зв'язку (сторона клієнта)

function connect(const s: TSocket; const name: PSockAddr; namelen: Integer): Integer;

7. Запис до сокета.

function send(s: TSocket; var Buf; len, flags: Integer): Integer;

8. Читання з сокета.

function recv(s: TSocket; var Buf; len, flags: Integer): Integer;

Опис бібліотеки ICS (Internet Component Suite)

ICS – пакет компонентів роботи з Інтернетом.

Компоненти пакета *ICS*:

TWSocket – основний компонент, на базі якого будується вся робота з *winsock.dll*. Компонент добре відлагоджений, достатня кількість *event'*ов для керування поведінкою додатку. Компонент може використовуватися в різних потоках програми. Реалізована підтримка *TCP* і *UDP*. Даний компонент може використовуватися для побудови як клієнтських, так і серверних додатків. На сервері доступна технічна інформація про *TCP* і *UDP*.

TSmtpCli – компонент для підтримки протоколу *SMTP*. Використовується для відсилання на поштовий сервер листів і вкладених файлів.

TPop3Cli – компонент реалізує підтримку протоколу POP3 для доставки повідомлень з поштового серверу.

TMimeDecode – компонент призначений для декодування вкладень в листи.

TFtpCli – компонент реалізує підтримку *FTP* клієнта. Дозволяє працювати як у режимі *Download*, так і в *Upload*. Має підтримку для роботи з каталогами на видаленому сервері.

TFtpSrv – компонент дозволяє побудувати повністю працездатний FTP-сервер. Версія - Бета.

TNntpCli – компонент, в якому реалізована підтримка протоколу *NNTP*.

THttpCli – компонент дозволяє побудувати власний *Web*-браузер. Реалізована можливість роботи через *Proxi*-сервер.

TTnClx – компонент, який реалізує підтримку протоколу *TELNET*.

TEmulVT – емуляція *ANSI* термінала (аналог компоненту *TMemo,* але з підтримкою кодів, що керують).

TFingerCli – в компоненті повністю реалізований *Finger*-клієнт. Дозволяє отримувати інформацію про користувача, якого підключено до *UNIX*-серверу або будь-якого іншого, на якому запущений *Finger*-сервіс.

TPing – реалізація *ICM* ping'а.

Для розробки програм тестової діаностики мережі можуть використовуватися два компонента бібліотеки *ICS*.

1. *WSocket* – на стороні клієнта.
2. W*SocketServer* – на стороні серверу.

Опис бібліотеки Indy (Internet Direct)

Бібліотека *Indy* – це набір компонентів для роботи із більшістю інтернет-протоколів (*HTTP, FTP, SMTP, UDP, ICMP, Telnet*). Компоненти цієї бібліотеки можуть використовуватися для створення крос-платформених додатків.

Технологія, що вживається в *Indy*, використовує операції читання і запису з блокуванням. Будь-яка операція *Connect*, яка використовується в

Indy, чекає завершення з'єднання. При роботі з клієнтськими компонентами *Indy*, як правило, потрібне виконання таких операцій:

- запитати з'єднання з сервером;
- здійснити запити до серверу на читання і запис (залежно від типу серверу крок виконується одного разу або повторюється багато разів);
- закінчити з'єднання з сервером і роз'єднатися.

Компоненти *Indy* розроблялися так, щоб забезпечити надвисокий рівень абстракції. Заплутаність і подробиці стека *TCP/IP* приховані від програміста, з тим щоб він міг зосередитися на поточних завданнях.

На стороні клієнта може використовуватися компонент *IdTCPClient*, а на стороні серверу – *IdTCPServer*. Ці компоненти використовуються для підтримки одного з основних мережевих протоколів – *TCP*, а також є базовими класами для компонентів *TIdSMTP* і *TIdFTP*. Клас *TIdTCPServer* має властивість *ThreadMgr*, яка за замовчанням дорівнює *nil*. Якщо *ThreadMgr* дорівнює *nil*, а *TIdTCPServer* активізовано, клас *TIdThreadMgrDeafault* буде створено неявно. Інакше використовується встановлений менеджер процесів.

20.2 Узагальнений алгоритм роботи модулів

Робота модуля-клієнта починається з встановлення параметрів (адреси, номера порту, протоколу) серверу, з яким клієнт повинен з'єднатися. Потім здійснюється з'єднання з сервером. Після цього починається відлік часу для вимірювання часу обміну даними. Виконуються посилання і приймання даних, після чого відлік часу зупиняється. Інформація про результати і час обміну даними виводиться на форму і зв'язок з сервером припиняється.

Робота модуля-серверу починається з встановлення його параметрів (номера порту, протоколу), знаючи які клієнт зможе встановити з'єднання із сервером. Потім сервер переводиться у стан очікування підключення клієнта. Коли з`єднання з клієнтом відбувається, сервер отримує посилку та відповідає на неї. По закінченні обміну даними зв'язок з клієнтом припиняється.

20.2.1 Алгоритми роботи модуля Client

Алгоритм роботи модуля Client з WinSock2:

1. Ініціалізація бібліотеки *WinSock2:*

```
if WSAStartup($0002,D)<>0 then begin
  ShowMessage('error..WSA Client!');
  exit;
end;
```

2. Створення сокета:

```
S:=Socket(AF_INET, SOCK_STREAM, 0);
```

```
if S=INVALID_SOCKET then
  begin ShowMessage('Error...Soket!'); exit; end;
```

3. Ініціалізація адресной структури:

```
CAdr.sin_family:=AF_INET;
CAdr.sin_addr.S_addr:=inet_addr(Pchar(edit3.text));
if (CAdr.sin_addr.S_addr=INADDR_NONE) then
  begin ShowMessage('Неверно введён IP-адрес!!'); exit; end;
CAdr.sin_port := htons(5000);
```

4. З'єднання із сервером:

```
i:=connect(S,@CAdr,sizeof(CAdr));
if (i<>0) then begin ShowMessage('connection error!'); exit; end;
```

5. Початок вимірювання часу:

```
asm  dw   310Fh
    mov  TimerLo,eax
    mov  TimerHi,edx
end;
```

6. Посилання даних:

```
send(s,buf_send,4,0);
```

7. Прийом даних:

```
recv(S,buf,4,0);
```

8. Зупинка вимірювання часу:

```
asm  dw   310Fh
    sub  eax,TimerLo
    sub  edx,TimerHi
    mov  TimerLo,eax
    mov  TimerHi,edx
end;
```

9. Підрахунок пройденого часу у мікросекундах:

```
t:=(TimerHi*$100000000+TimerLo)/CPU_Speed;
```

10. Перевірка отриманих даних та виведення результатів у форму:

```
memo1.lines.add('Sent message: '+inttohex(buf_send[0],2)
+inttohex(buf_send[1],2)+inttohex(buf_send[2],2)+inttohex(buf_send[3],
2));
memo1.lines.add('Received message: '+inttohex(buf[0],2)
+inttohex(buf[1],2)+inttohex(buf[2],2)+inttohex(buf[3],2));
memo1.lines.add('Time: '+FloatToStr(t)+' microseconds');
for i:=0 to 3 do if buf[i]<>buf_send[i]
  then begin
   memo1.lines.add('ERROR! Receiving message is wrong!');
   break;
  end;
memo1.lines.add('==============================');
```

11. Закриття сокета:

```
closesocket(s);
```

12. Вивантаження бібліотеки:

WSACleanup;

Схема алгоритму роботи клієнта, що використовує бібліотеку *WinSock2*, наведена на рис. 20.1.

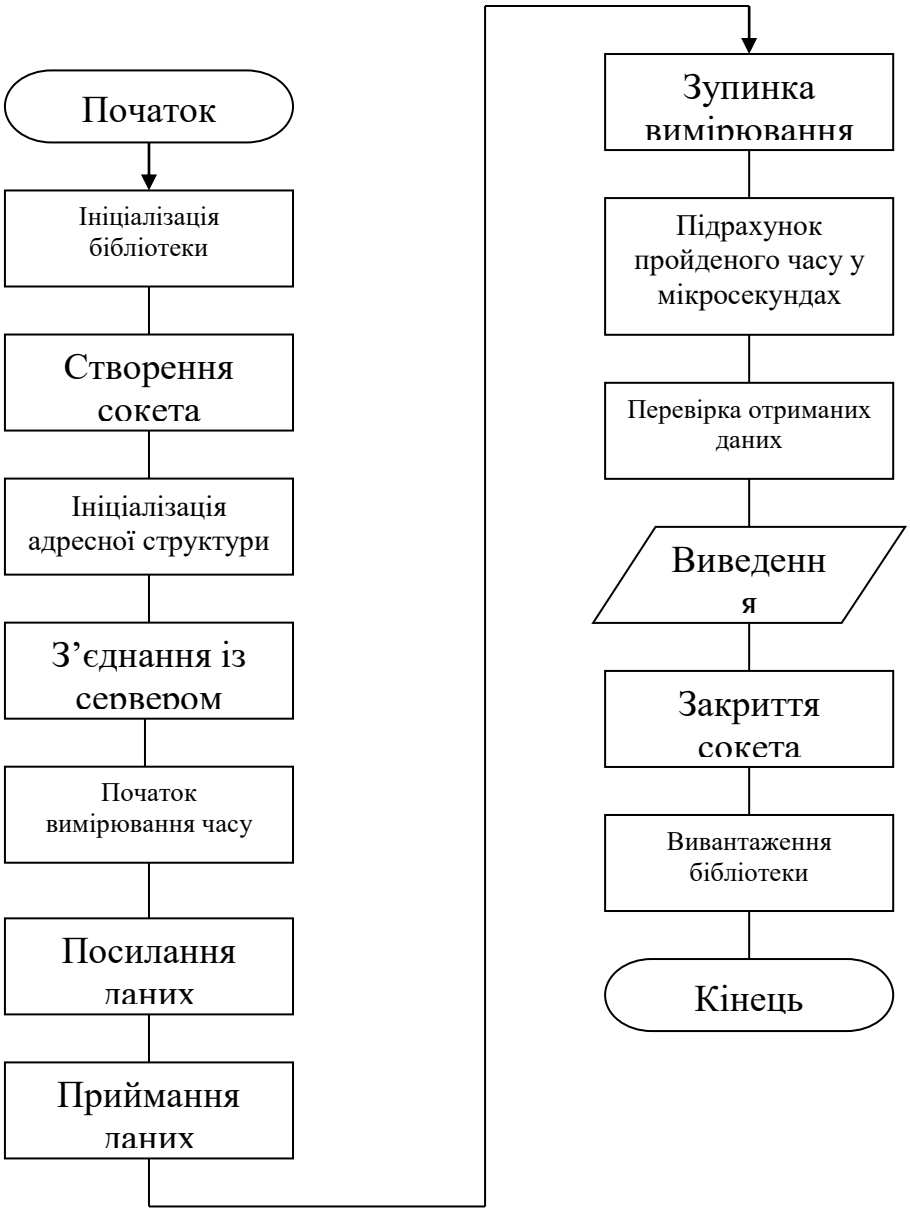

Рисунок 20.1 – Схема алгоритму роботи клієнта, що використовує бібліотеку *WinSock2*

Алгоритм роботи модуля з бібліотекою Indy

1. Ініціалізація параметрів серверу (адреси и номера порту):

```
with IdTCPClient1 do
begin
  Host:=edit3.Text;
  Port:=5001;
```

2. З'єднання із сервером:

```
Connect;
```

3. Початок вимірювання часу:

```
asm  dw   310Fh
    mov  TimerLo,eax
    mov  TimerHi,edx
end;
```

4. Посилання даних:

```
WriteLn('Test');
```

5. Прийом даних:

```
s:=readln;
```

6. Зупинка вимірювання часу:

```
asm  dw   310Fh
    sub  eax,TimerLo
    sub  edx,TimerHi
    mov  TimerLo,eax
    mov  TimerHi,edx
end;
```

7. Підрахунок пройденого часу у мікросекундах:

```
t:=(TimerHi*$100000000+TimerLo)/CPU_Speed;
```

8. Перевірка отриманих даних та виведення результатів у форму:

```
memo1.Lines.Add('Send to server string: "Test"');
if s='Ok' then
   memo1.Lines.Add('Recive from server string: "'+s
+'"'+#13+#10+'Test was completed without errors!')   else
memo1.Lines.Add('Recive from server string: '+s+#13+#10+'Test was not
completed!');
memo1.lines.add('Time: '+FloatToStr(t)+' microseconds');

memo1.lines.add('==================================');
```

9. Роз'єднання зв'язку:

```
Disconnect;
```

Схема алгоритму роботи клієнта, що використовує бібліотеку *Indy*, наведена на рис. 20.2.

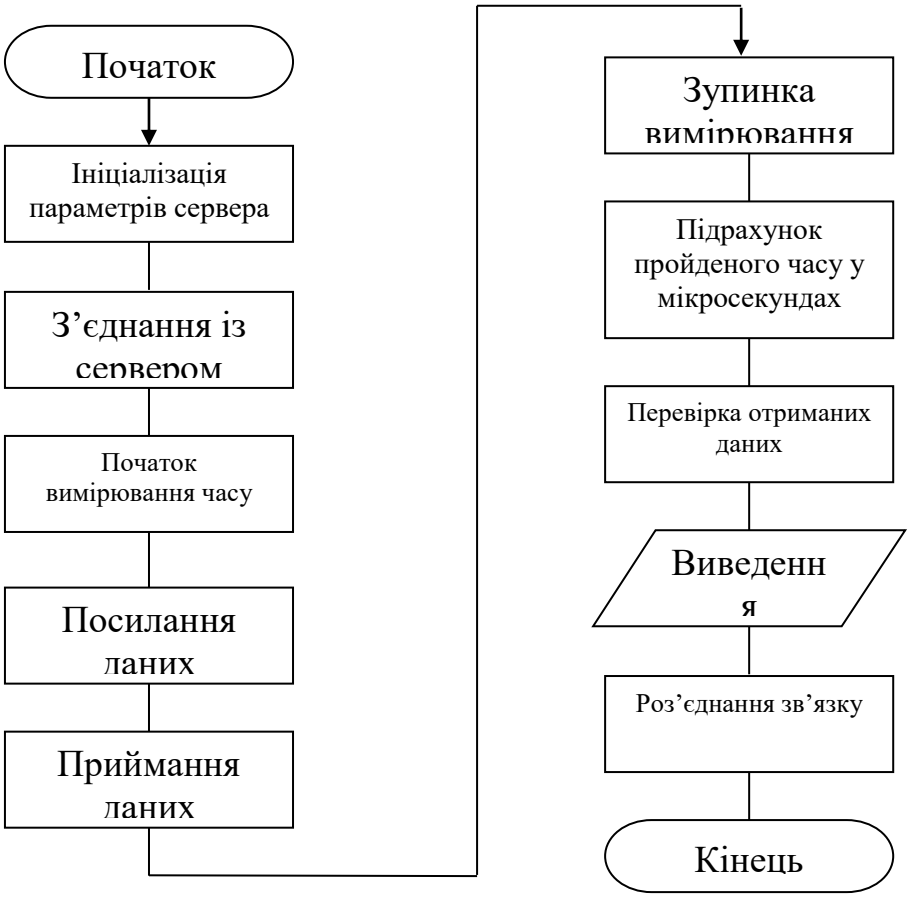

Рисунок 20.2 – Схема алгоритму роботи клієнта, що використовує бібліотеку *Indy*

Алгоритм роботи модуля з бібліотекою ICS

1. Ініціалізація параметрів серверу (адреси і номера порту):

```
WSocket1.Addr:=edit3.Text;
WSocket1.Port:='5002';
WSocket1.Proto:='tcp';
```

2. З'єднання із сервером:

```
WSocket1.Connect;
```

3. Початок вимірювання часу:

```
asm  dw   310Fh
     mov  TimerLo,eax
     mov  TimerHi,edx
end;
```

4. Посилання даних:

```
WSocket1.SendStr('Test');
```

5. Прийом даних:

```
s:=WSocket1.ReceiveStr;
```

6. Зупинка вимірювання часу:

```
asm  dw   310Fh
    sub  eax,TimerLo
    sub  edx,TimerHi
    mov  TimerLo,eax
    mov  TimerHi,edx
end;
```

7. Підрахунок пройденого часу у мікросекундах:

```
t:=(TimerHi*$100000000+TimerLo)/CPU_Speed;
```

8. Перевірка отриманих даних та виведення результатів у форму:

```
Memo1.Lines.Add('Send to server: "Test"');
Memo1.Lines.Add('Receive from server: "'+s+'"');
if s='Test is Ok!' then
  memo1.Lines.Add('Test was completed without errors!')
    else   memo1.Lines.Add('Test was not completed!');
memo1.lines.add('Time:  '+FloatToStr(t)+' microseconds');

memo1.lines.add('==================================');
```

9. Роз'єднання зв'язку:

```
WSocket1.Close;
```

Схема алгоритму роботи клієнта, що використовує бібліотеку *ICS*, наведена на рис. 20.3.

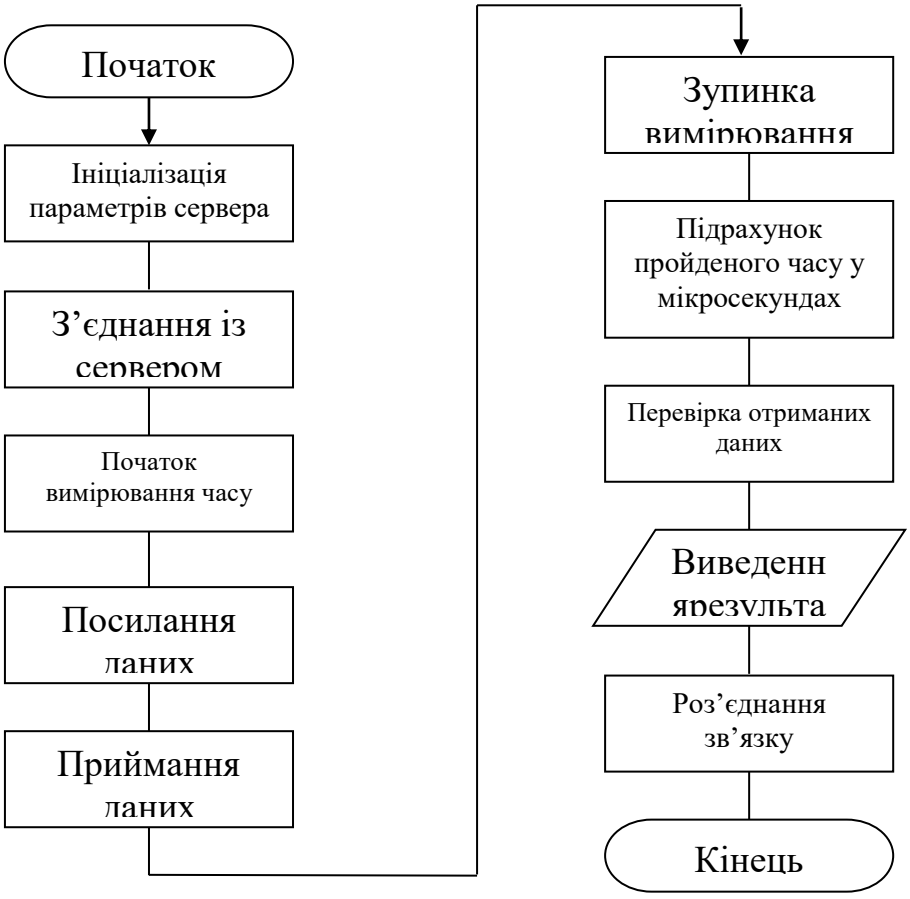

Рисунок 20.3 – Схема алгоритму роботи клієнта, що використовує бібліотеку *ICS*

20.2.2 Алгоритми роботи модуля Server

Алгоритм роботи модуля з бібліотекою *WinSock2*

1. Ініціалізація бібліотеки WinSock2:

```
if WSAStartup($0002,D)<>0 then
begin ShowMessage('Error..WSADate');
    memo1.lines.add('WSA Startup failed.');
    exit;end;
```

2. Створення сокета:

```
S:=Socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if S=INVALID_SOCKET then begin
 ShowMessage('Error...Soket!'); exit; end;
```

3. Ініціалізація адресной структури:

```
SAdr.sin_family := PF_INET;
SAdr.sin_addr.S_addr := INADDR_ANY;
SAdr.sin_port := htons(5000);
```

4. Прив'язка адреси до сокета:

```
if bind(S,@SAdr,Sizeof(SAdr))=SOCKET_ERROR then
  begin ShowMessage('Error...bind!');exit; end;
```

5. Прослуховування порту:

```
if listen(S,1)=SOCKET_ERROR then begin
 ShowMessage('Error...listen!'); exit; end;
```

6. Підтвердження з'єднання:

```
L:=Sizeof(ClAdr);
S1:= accept(S, ClAdr, L);
```

7. Приймання даних:

```
recv(S1,buf,4,0);
```

8. Посилання даних:

```
send(S1,buf,4,0);
```

9. Виведення результатів у форму:

```
memo1.lines.add('Received and sent message: '+inttohex(buf[0],2)
+inttohex(buf[1],2)+inttohex(buf[2],2)+inttohex(buf[3],2));
memo1.lines.add('============================');
```

10. Закриття сокета:

```
closesocket(S1);
closesocket(S);
```

11. Вивантаження бібліотеки:

```
WSACleanup;
```

Схема алгоритму роботи серверу, що використовує бібліотеку *WinSock2*, наведена на рис. 20.4.

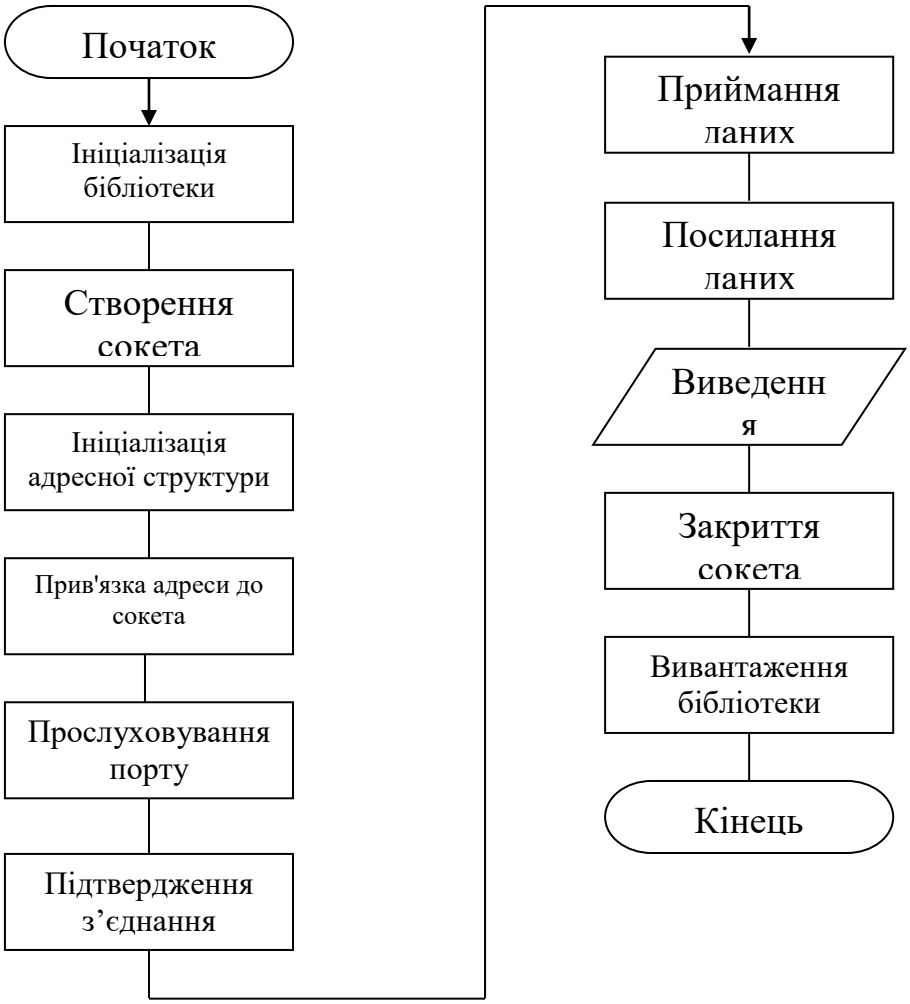

Рисунок 20.4 – Схема алгоритму роботи сервера, що використовує бібліотеку WinSock2

Алгоритм роботи модуля з бібліотекою *Indy*

1. Ініціалізація параметрів серверу (номер порту):
   IdTCPServer1.DefaultPort:=5001;
2. Активація (запуск) серверу в режимі очікування підключення:
   IdTCPServer1.Active:=true;
3. Приймання даних у разі підключення:
   with AThread.Connection do
   try
   s := ReadLn;
4. Посилання даних:
   if s= 'Test' then writeln('Ok') else writeln('No');

5. Роз’єднання зв’язку:

finally
Disconnect;
end;

Схема алгоритму роботи серверу, що використовує бібліотеку *Indy*, наведена на рис. 20.5

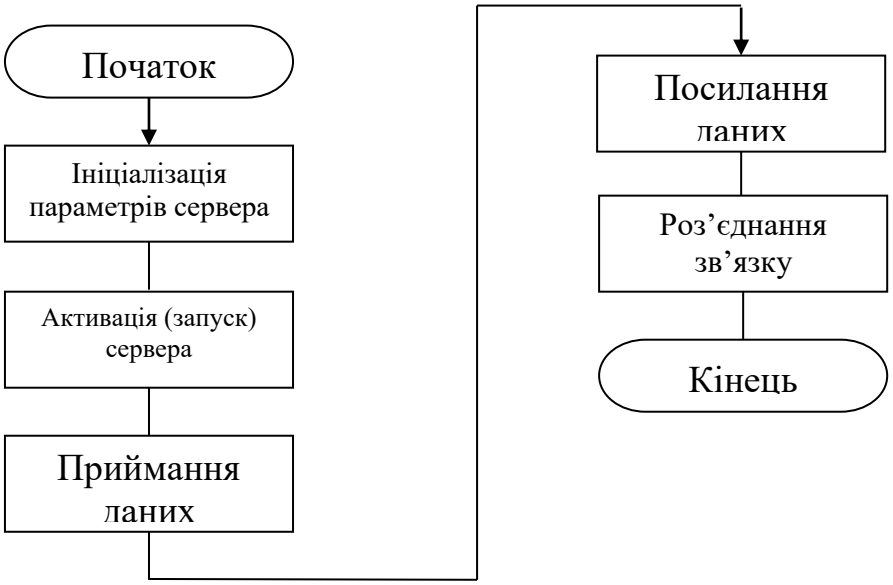

Рисунок 20.5 – Схема алгоритму роботи клієнта, що використовує бібліотеку *Indy*

Алгоритм роботи модуля з бібліотекою *ICS*

1. Ініціалізація параметрів серверу:

WSocketServer1.Port:= ‘5002’;
WSocketServer1.Proto:= ‘tcp’;
WSocketServer1.Addr := ‘0.0.0.0’;

2. Прослуховування порту:

WSocketServer1.Listen;

3. Підтвердження з’єднання з клієнтом:

NewHSocket := WSocketServer1.Accept;
CliSocket := TWSocket.Create(self);
CliSocket.Dup(NewHSocket);

4. Встановлення процедури, яка викликається при надходженні на порт даних:

CliSocket.OnDataAvailable := ClientReceive;

5. Приймання даних:

s := (Sender as TWSocket).ReceiveStr;

6. Посилання даних:

if s= ‘Test’ then (Sender as TWSocket).SendStr(‘Test is Ok!’)
else (Sender as TWSocket).SendStr(‘Test is failed!’);

7. Роз’єднання зв’язку:

(Sender as TWSocket).Close;

Схема алгоритму роботи серверу, що використовує бібліотеку *ICS*, наведена на рис. 20.6

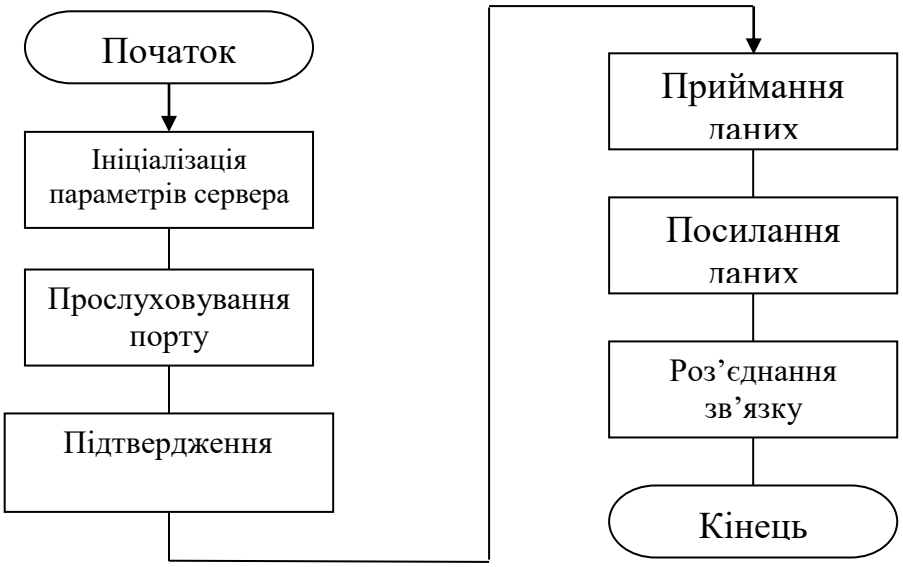

Рисунок 20.6 – Схема алгоритму роботи клієнта, що використовує бібліотеку *ICS*

20.3 Результати діагностики мережі

Програма *Server* запускається першою на тому ПК, зв'язок з яким ми перевіряємо з даного ПК, на якому буде запущена програма *Client*.

Маємо три віконця відповідно до бібліотек *WinSock, ICS, Indy*.

Для *WinSock* маємо таке заповнення форми:

- версія *WinSock*;
- стан.

Програма очікує запиту клієнта.

Усі форми серверу необхідно по черзі ініціювати.

Ініціювання серверу *WinSock* наведено на рис. 20.7:

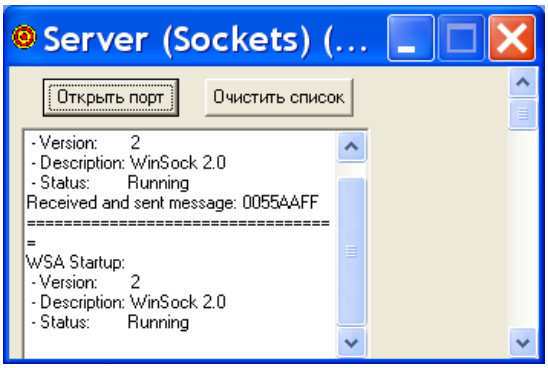

Рисунок 20.7 – Ініціювання серверу *WinSock*

Ініціювання серверу *Indy* наведено на рис. 20.8.

Рисунок 20.8 – Ініціювання серверу *Indy*

Ініціювання серверу *ICS* наведено на рис. 20.9.

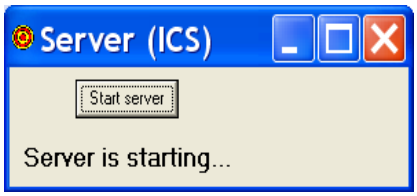

Рисунок 20.9 – Ініціювання серверу *ICS*

Після запуску та відкриття програми сервер запускаємо програму клієнт.

В ході запуску програми *Client* маємо три віконця з *WinSock2*, *Indy*, *ICS*.

Ініціювання клієнта *WinSock* наведено на рис. 20.10.

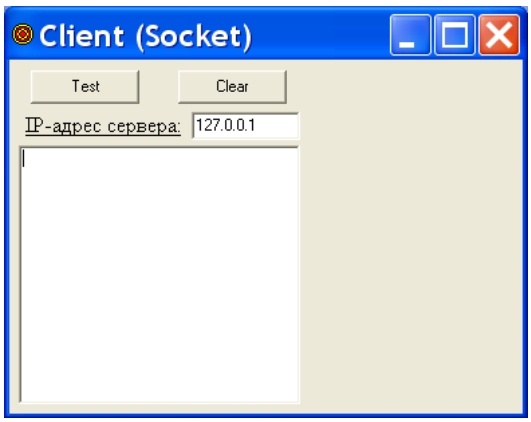

Рисунок 20.10 – Ініціювання клієнта *WinSock*

Ініціювання клієнта *Indy* наведено на рис. 20.11.

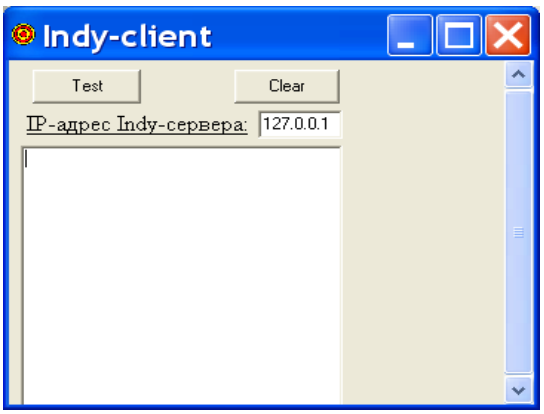

Рисунок 20.11 – Ініціювання клієнта *Indy*

Ініціювання клієнта *ICS* наведено на рис. 20.12.

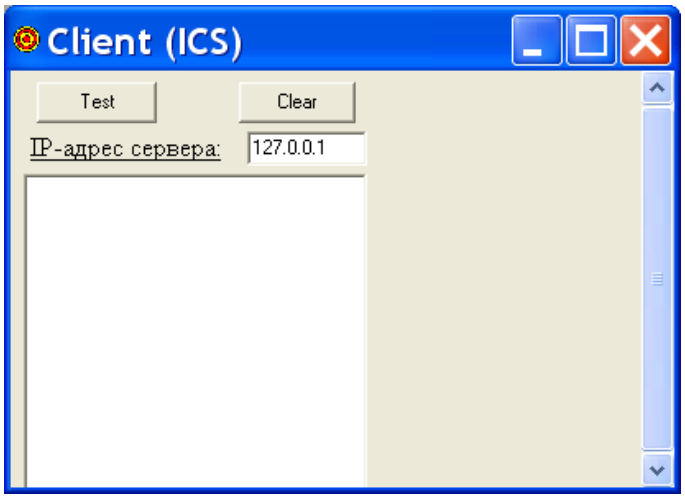

Рисунок 20.12 – Ініціювання клієнта *ICS*

Після запуску клієнтських вікон відповідно отримаємо повідомлення: для W*inSock* (рис. 20.13), для *Indy* (рис. 20.14), для *ICS* (рис. 20.15):

Рисунок 20.13 – Повідомлення для *WinSock*

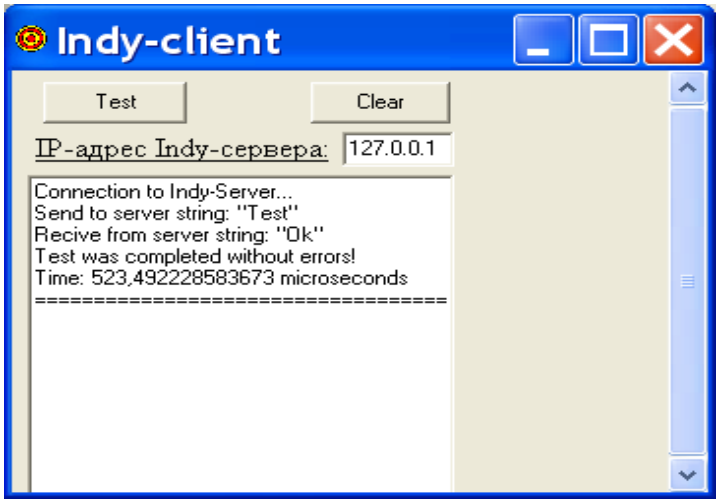

Рисунок 20.14 – Повідомлення для *Indy*

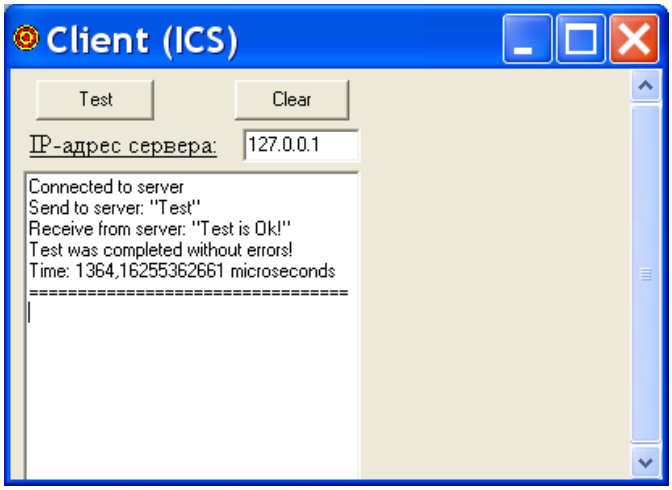

Рисунок 20.15 – Повідомлення для *ICS*

Кожна форма програми – клієнт виводить прийняте повідомлення “*Test is Ok*!” , а також час проходження запиту – відповіді у мкс. Отримавши подібні повідомлення можна зробити висновок про працездатність мережі.

Контрольні запитання

1. Що таке TCP (Transmission Control Protocol) ?
2. Для чого призначені функції бібліотеки *WinSock* ?
3. Назвіть приклади популярних прикладних програм *TCP/IP*, які спираються на використання *TCP*.
4. Яке призначення TWSocket – основного компонента пакета *ICS* ?
5. Яке призначення *TSmtpCli*t – компонент пакета *ICS* ?
6. Яке призначення *TPop3Cli* – компонент пакета *ICS* ?
7. Яке призначення *TMimeDecode* – компонент пакета *ICS* ?
8. Яке призначення *TFtpCli* – компонент пакета *ICS* ?
9. Яке призначення *TFtpSrv* – компонент пакета *ICS* ?
10. Яке призначення *TNntpCli* – компонент пакета *ICS* ?
11. Яке призначення *THttpCli* – компонент пакета *ICS* ?
12. Яке призначення *TTnClx* – компонент пакета *ICS* ?
13. Яке призначення *TEmulVT* – компонент пакета *ICS* ?
14. Яке призначення *TFingerCli* – компонент пакета *ICS* ?
15. Яке призначення *TPing* – компонент пакета *ICS* ?

ЗМІСТ

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Опис операційної системи Mikrotik RouterOS // http://wiki.mikrotik.com, 02.07.24.
2. Коробейнікова Т.І. Комп'ютерні мережі. Навчальний посібник / Т.І. Коробейнікова, С.М. Захарченко – Львів: Львівська політехніка, 2022. – 228 с.
3. Микитишин А.Г. Комп'ютерні мережі. Книга.1. Навчальний посібник для технічних спеціальностей ВНЗ (Рекомендовано МОН) / А.Г.Микитишин, М.М. Митник, П.Д. Стухляк – Львів: Магнолія 2006, 2023. – 256 с.
4. Dordal Peter Lars Peter. An Introduction to Computer Networks – Peter L. Dordal, 2022. – 951 p.
5. Olifer Natalia, Olifer Victor. Computer Networks: Principles, Technologies and Protocols for Network
6. Design-International Edition – Wiley India Private Limited; 1st. edition, 2006. – 1000 p.
7. Peterson Larry, Davie Bruce. Read more about Computer Networks: A Systems Approach – Peterson and Davie, 2019. – 485 p.
8. Ross Keith W., Kurose James F. Computer Networking: A Top-Down Approach. 8th Edition. – Pearson, 2022. – 820 p.
9. Seifert Rich, Edwards James. The All-New Switch Book: The Complete Guide to LAN Switching Technology, 2nd Edition : Wiley, 2008. – 816 p.
10. Tanenbaum Andrew S. Computer networks / Andrew S. Tanenbaum, David J. Wetheral. – Prentice Hall, 2012 – 962 p.

Навчальне видання

МЕЗЕНЦЕВ Микола Вікторович
НОСКОВ Валентин Іванович

КОМП’ЮТЕРНІ МЕРЕЖІ

для студентів спеціальності 123 – «Комп’ютерна інженерія»

В авторскій редакції

Відповідальний за випуск О.Ю. Заковоротний
Роботу до видання рекомендував М.Й. Заполовський

План 2024, поз. 101

Видавничий центр НТУ «ХПІ»
61002, Харків, вул. Кирпичова, 2.
Свідоцтво суб’єкта видавничої справи ДК № 5478 від 21.08.2017 р.

Електронна версія