УДК 004.7

О.П. Ігнатенко, Д.В. Цицкун

СЕРЕДОВИЩЕ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ КОНФЛІКТНИХ ПРОЦЕСІВ У КОМП'ЮТЕРНИХ МЕРЕЖАХ

При проектуванні або аналізі функціонування складних комп'ютерних мереж необхідним етапом являється їх імітаційне моделювання. За допомогою адекватної моделі роботи мережі розробники можуть суттєво зменшити витрати часу і грошей на оптимізацію функціонування мережі, розробку та апробацію алгоритмів керування при виникненні конфліктних ситуацій. Одною з нагальних проблем сучасного моделювання мереж є побудова систем їх захисту від атак на відмову. Тому сучасне середовище має включати комплексну систему моделювання атак та системи захисту. В роботі побудовано імітаційну модель системи обробки різнотипних пакетів за умов нормальної роботи та за наявності атаки на відмову і проведений аналіз характеристик її роботи.

Вступ

У роботі [1] авторами проведено порівняльний аналіз сучасних середовищ моделювання комп'ютерних мереж. Найкращі характеристики для моделювання конфліктних процесів у комп'ютерних мережах серед яких має середовище OMNeT++ [2]. Воно представляє собою інтегроване середовище, призначене для створення, налаштування і запуску імітаційних дискретних моделей. Це середовище, розроблене на основі мови C++, є масштабованим, модульним рішенням. OMNeT++ надає інфраструктуру для конструювання модулів, визначення структури повідомлень та визначення правил їх обробки. Його назва утворена від – Objective Modular Network Testbed in C++ (об'єктний модульний полігон для моделювання мереж на C++).

Середовище розроблено за принципами вільного розповсюдження і може вільно використовуватися для наукових цілей. Воно може використовуватися на платформах Windows і Unix (Linux також).

OMNeT++ може використовуватися для проведення дослідження в наступних напрямках:

- моделювання провідних і безпровідних комунікаційних мереж;
- моделювання протоколів роботи мереж;
- моделювання задач теорії черг;
- моделювання багатопроцесорних та інших розподілених комп'ютерних систем;
- тестування архітектури;
- проведення імітаційного моделювання роботи складних програмних систем;

Взагалі кажучи середовище може використовуватися для моделювання будь-яких систем, які допускають дискретне представлення і можуть бути описані у вигляді сутностей, що обмінюються повідомленнями.

1. Загальний опис

Інтегроване середовище розробки IDE OMNeT++ 4.0 побудоване на базі платформи Eclipse. Воно дозволяє створювати та налаштовувати моделі (NED та ini файли), виконувати запуск моделі на виконання, проводити аналіз результатів. Eclipse надає можливості редагування C++ файлів, компіляції, UML моделювання та інших додаткових функцій. Основні компоненти IDE OMNeT++ 4.0 показані на рис. 1. Редактор NED дозволяє редагувати NED файли як у графічному, так і в текстовому режимі. Користувач має можливість переключатися між цими режимами в будь-який час, використовуючи закладки у вікні редагування.

У графічному режимі користувач має можливість створювати складні модулі, канали зв'язку та інші компоненти моделі. Підмодулі, що були описані раніше, можуть бути використані за допомогою палітри доступних компонентів.

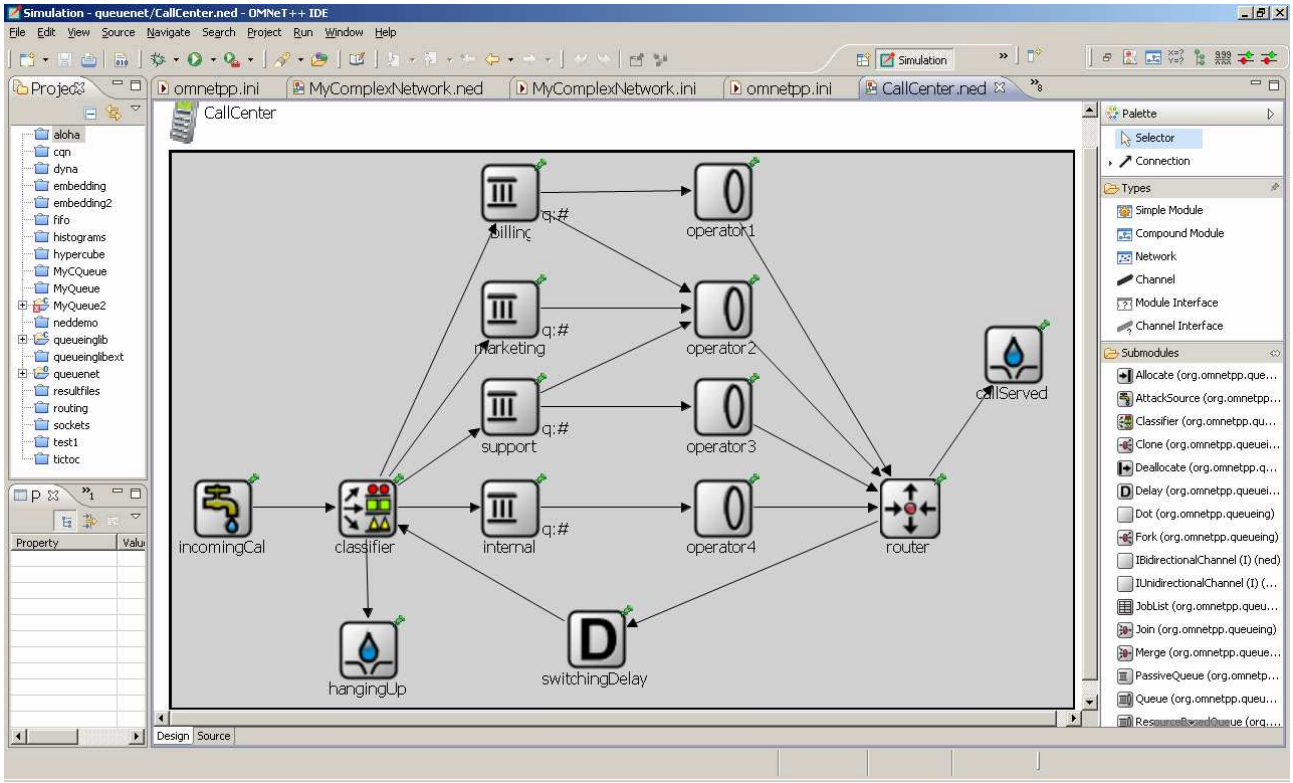

Рис. 1. Інтерфейс розробки середовища OMNet++

1.1. Концепція моделювання

Під моделлю розуміють множину модулів, які можуть обмінюватися повідомленнями. При цьому базові модулі, які мають функціональність описану мовою C++, називаються *простими* модулями. Прості модулі можуть бути об'єднані у *складені*, ті у свою чергу можуть входити в інші модулі; при цьому кількість ієрархічних вкладень не обмежена. Вся модель у цілому називається *мережею* і є по суті складеним модулем. Повідомлення можуть передаватися через канали зв'язку проведені між модулями або напряму. На рис. 2 показана взаємодія простих і складених модулів.

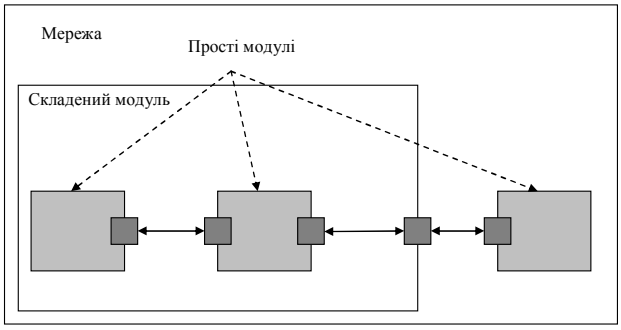

Рис. 2. Взаємодія простих і складених модулів

Сірі прямокутники – прості модулі, темно-сірі квадратики – портали (gates) через які відбуваються зв'язки (зображені стрілками).

Модулі взаємодіють шляхом надсилання повідомлень, які можуть містити будь-які додаткові інформаційні поля (не тільки стандартний час відправки). Прості модулі як правило надсилають повідомлення через портали, однак існує можливість надсилати повідомлення напряму до заданого модуля.

Портали є вхідними і вихідними інтерфейсами модулів. Існують вхідні й вихідні портали, які можна зв'язати каналом передачі повідомлень (конекторами).

Конектори створюються на одному рівні ієрархії. Наприклад, всередині складеного модуля конектори можуть бути створені між порталами двох підмодулів, або між порталом підмодуля і порталом складеного модуля.

Портали є вхідними і вихідними інтерфейсами модулів. Існують вхідні і вихідні портали, які можна зв'язати каналом передачі повідомлень (конекторами).

Конектори створюються на одному рівні ієрархії. Наприклад, всередині складеного модуля конектори можуть бути створені між порталами двох підмодулів, або між порталом підмодуля і порталом складеного модуля. Зв'язки, що виходять на інші рівні ієрархії не дозволені, оскільки перешкоджають повторному використанню моделей.

Внаслідок ієрархічної побудови моделей повідомлення, як правило, передаються через ланцюжок конекторів, який починається і закінчується у простому модулі. Складені модулі при цьому виконують роль контейнерів, які передають повідомлення зсередини, зовні та навпаки.

Параметри зв'язку, такі як затримка розповсюдження, частота передачі даних та частота виникнення помилок можуть бути призначені кожному конектору. Існує також можливість визначення своїх власних типів конекторів із особливими параметрами. Модулі також можуть мати параметри, які в основному використовуються для передачі конфігураційних даних у прості модулі та для більшої гнучкості при визначенні топології моделі. Параметри можуть бути числами, рядками або булевими змінними.

Оскільки параметри доступні у програмі (імплементації простого модуля) у вигляді об'єктів, то їх можна використовувати не тільки як константи але як джерело випадкових чисел з розподілом, заданим у активній конфігурації моделі. Параметри також можуть бути запитані у користувача під час виконання. Складені модулі можуть передавати значення параметрів своїм компонентам.

Таким чином, середовище надає ефективний інструментарій для визначення структури системи. Зазначимо основні особливості, що дозволяють гнучко перебудовувати конфігурації системи:

- ієрархічна структура модулів;
- модулі являються екземплярами типів;
- модулі взаємодіють шляхом обміну повідомленнями;
- можливість визначення власних параметрів модулів;
- можливість визначати топологію мережі за допомогою спеціальних конструкцій.

Ієрархічні модулі і модульні типи. Модель складається з ієрархічно вкладених модулів, які взаємодіють шляхом обміну повідомленнями. Структура моделі описується за допомогою спеціально створеної мови NED. Функціональність простих модулів реалізується мовою C++. Прості і складені модулі є екземплярами певних типів. При розробці моделі користувач спочатку визначає базові модульні типи. На їх основі створюються більш складні моделі. Наприкінці користувач розробляє загальний системний модуль, який містить інші у вигляді ієрархії підмодулів.

При використанні модулів як конструкторські елементи немає різниці прості вони чи складені. Такий підхід дозволяє розділити простий модуль на складові, вбудовувати їх в інші модулі або навпаки перенести функціональність складеного модуля у простий без впливу на інші модулі, що використовують даний тип. Модульні типи можуть зберігатися окремими файлами. Це означає, що користувачі можуть створювати бібліотеки активних типів.

Повідомлення, портали та зв'язки. Як уже зазначалось, модулі взаємодіють шляхом обміну повідомленнями. В реальній системі повідомлення можуть представляти фрейми або пакети в комп'ютерній мережі, завдання або клієнтів у комунікаційній системі, або інші види мобільних сутностей. Повідомлення можуть містити як завгодно складну структуру даних. Прості модулі можуть надсилати повідомлення напряму у місце призначення або визначеним маршрутом через портали і зв'язки. З кожним модулем пов'язаний так званий «локальний час моделювання», який змінюється при отриманні повідомлення, яке може прийти від іншого або того самого модуля (авто надсилання повідомлень використовується для встановлення таймерів).

Портали – це вхідні та вихідні інтерфейси модулів; розрізняють вхідні і вихідні портали.

З коннекторами можуть бути пов'язані три параметри, які були створені для моделювання мереж, але можуть використовуватися і в інших моделях:

- затримка розповсюдження;
- частота передачі даних;
- частота виникнення помилок.

Параметри задаються для кожного зв'язку окремо, але можна створити свій тип з'єднання для використання у моделі.

Затримка розповсюдження – це час на який всі повідомлення затримуються при переміщенні даним конектором.

Частота виникнення помилок – є ймовірність того, що біт даних буде спотворено при передачі. Цей параметр дозволяє використовувати просту модель шуму в каналі.

Частота передачі даних задається у вигляді біти/секунди і використовується для обчислення часу передачі пакетів по каналу. При використанні цього параметра надсилання повідомлення пов'язано з передачею даних – починається з надсиланням першого біта і закінчується прийомом останнього. Ця модель не завжди може використовуватися, наприклад, протоколи Token Ring або FDDI не чекають отримання останнього біта пові-домлення а починають передавати перші біти через деякий час після їх отримання. Іншими словами, фрейми «протікають» крізь вузли затримуючись лише на декілька бітів. Для моделювання таких мереж потрібно змінити існуючі програми обробки повідомлень.

Параметри. Значення параметрів може бути задано при описані мережі в NED файлі або в конфігураційному файлі, який виконується перед запуском моделі.

Параметри корисні для налагодження поведінки простих модулів або для параметризації топології моделі. Вони можуть бути числом, рядком або булевською константою.

Числові значення це, як правило, вирази, що використовують виклики функцій С, випадкових величин з різними розподілами або значення, які вводяться користувачем. У складеному модулі параметри можуть задавати кількість підмодулів, порталів та спосіб утворення внутрішніх з'єднань.

Програмування алгоритмів. Прості модулі містять алгоритми поведінки у вигляді функцій C++. Використовуючи бібліотеку класів можна швидко і гнучко описувати будь-яку поведінку елементів мережі. Середовище моделювання побудоване на об'єктно-орієнтованому підході, тому розробник може легко розширювати існуючу функціональність і додавати спеціальні функції.

Всі об'єкти – елементи моделі (повідомлення, модулі, черги та ін.) є C++ класами. Вони були розроблені як частина середовища для забезпечення ефективної роботи. Виділяють наступні класи:

- модулі, портали, зв'язки;
- параметри;
- повідомлення;
- контейнерні класи (черги, масиви та інше);
- набори даних;
- статистика та розподіли (гістограми, алгоритми оцінки й інше).

Ці класи мають спеціальні функції для стеження за об'єктами під час виконання та виведення необхідної інформації, такої як назва, клас, змінні та їх значення.

1.2. Основні особливості середовища моделювання

Побудовані моделі. Некомерційні продукти імітаційного моделювання, звичайно, не можуть конкурувати з комерційними програмними комплексами, які мають великий вибір підготовлених моделей. OMNeT++ не є винятком – наразі існує відносно невелика кількість побудованих моделей. З іншого боку, у цьому середовищі пропонується широкий вибір можливостей (що дозволяє користувачам моделювати не тільки комунікаційні мережі).

На сьогоднішній день в середовищі реалізовані модулі для моделювання черг, різноманітних протоколів функціонування комп'ютерних провідних та безпровідних мереж. Водночас існує нагальна необхідність створення пакета для моделювання процесів керування мережами, особливо у випадках конфлікту і невизначеності.

В роботах [3, 4] докладно розглядалися основні принципи побудови системи моделювання атак на відмову у комп'ю-

терних мережах. При побудові такої системи перед розробником постає необхідність розробити комплексну систему, яка б включала модулі:

- імітаційного моделювання поведінки нападника;
- систему виявлення;
- систему керування ресурсами мережі;
- систему протидії.

Розробка моделей. Ядром моделювання середовища є бібліотека класів, тобто моделі відділені від основного середовища моделювання. Користувач використовує при описі компонент своєї моделі будь-які класи та генерує файл запуску.

Надійність. Досить поширеною є ситуація, коли стандартні моделі, реалізовані у середовищі розробки, не перевірені на реальних даних, тобто не верифіковані. Одним з прикладів є реалізація TCP протоколу в розширенні INET framework. Це загальна проблема некомерційних засобів: кожен може написати модель але ніхто не дає гарантії її роботи. Більше того, деякі моделі все ще доопрацьовуються і містять лише частину заявленої функціональності.

Визначення топології мережі. Засоби для моделювання мереж, як правило, використовують інтуїтивне представлення про модель мережі як об'єкт, що складається з вузлів (блоків, сутностей, модулів та інших) які з'єднані між собою зв'язками (каналами, конвекторами та іншими).

Велика кількість комерційних продуктів використовує спеціальні графічні редактори для визначення топології мережі. Однак, такий підхід можливий тільки у тому випадку, якщо існує альтернативний спосіб визначення топології (наприклад, текстовим файлом), який дозволяє генерувати мережу безпосередньо у програмі.

З іншого боку більшість некомерційних інструментів не дають можливості зовнішнього редагування (і графічного представлення) топології: користувач має запускати для файлу з описом певну «програму генерації» яка завантажить у модель необхідні описи вузлів та зв'язків між ними.

У середовищі моделювання використовується, мабуть, найбільш гнучкий спосіб: адаптований для сприйняття людиною текстовий формат топології (мова NED), який легко створюється будь-яким текстовим редактором і одночасно цей формат використовується у графічному редакторі. Також є можливість створити «програму генерації» для побудови мережі під час роботи програми. Підтримується навіть створення підмодулів без обмежень на рівень вкладеності.

Конфігурації запуску. Параметри для запуску зберігаються у файлі *omnetpp.ini*, що відповідає загальній концепції відокремлення моделі від процесу її виконання. Як правило, модель та її виконання (параметри запуску, результати) взаємопов'язані. Наприклад, в системі ns-2 параметри вмонтовані в Tcl скріпти і важкодоступні для редагування. Відлагодження моделей у C++ орієнтованих середовищах рідко коли надає більш широкі ніж C++ стандартні можливості по стеженню за процесом виконання. Пропонується принципово іншу архітектуру, яка надає доступ до бібліотеки функцій стеження за даними під час виконання. Використання цієї бібліотеки дозволяє не тільки побачити будь-який елемент моделі, створений користувачем через спеціальне вікно, але й впливати на значення змінних під час виконання.

1.3. Мова опису модулів NED

Користувач описує структуру імітаційної моделі за допомогою мови NED. NED – це скорочення від Network Description (опис мережі). NED дозволяє користувачу описувати прості модулі, з'єднувати їх зв'язками і об'єднувати у складені модулі. Користувач може позначати складені модулі як *мережі* – автономні імітаційні моделі. Зв'язки є іншим типом компонентів, які також можуть використовуватися у складених модулях.

Мова NED має декілька особливостей, які особливо важливі у масштабах великих проектів.

Ієрархічність. Традиційний спосіб боротьби зі складності полягає у введення ієрархії. У середовищі будь-який модуль, який би він не був складним, можна розді-

лити на окремі модулі, та водночас використовувати як складений модуль.

Компонентність. Всі модулі в OMNeT++ можуть використовуватися в будь-якій моделі. Це не тільки знижує затрати на кодування але дозволяє створювати бібліотеки компонент (наприклад, INET Framework, Mixim, Castalia та інші).

Інтерфейси. Інтерфейси модулів і каналів зв'язку можуть використовуватися як «вільні змінні». При цьому конкретний тип модуля може бути параметром конфігурації при здійсненні імітаційного моделювання і змінюватися при необхідності.

Спадкування. Для модулів і зв'язків можна визначити спадкові модулі. Останні можуть мати додаткові портали, параметри, підмодулі та з'єднання. Певні вільні параметри для них можуть бути зафіксовані, наприклад, розмірність вхідного порталу. Це дозволяє, наприклад, визначити модуль GenericTCPClientApp та визначити на його основі модуль FTPClientApp шляхом фіксування певних параметрів, або отримати модуль Web-ClientHost успадкувавши його від модуля BaseHost, та додавши модуль Web-ClientApp і його підключення до TCP підмодуля.

Внутрішні типи. Для зменшення кількості імен зв'язки і модулі, що використовуються у складеному модулі можуть бути визначені локально.

Мова NED має еквівалентне представлення у формі дерева, яке може бути переведене в XML формат. Таким чином, NED файли можуть транслюватися у XML і назад без втрати даних.

Це розширює можливості роботи з NED файлами, наприклад, вилучення інформації, рефакторинга, трансформації та автоматичної генерації NED файлів на основі інформації з інших систем.

2. Моделювання паралельних розподілених процесів

Середовище OMNeT++ надає можливість паралельного виконання великих імітаційних моделей. Опишемо основні особливості визначення і виконання моделей паралельного моделювання дискретних подій.

Для паралельного виконання, модель має бути розділена на кілька ЛП (логічних процесів), які будуть виконуватися незалежно, на різних вузлах або процесорах. Кожен ЛП буде мати свою власну *множину подій*, які будуть виконуватися щодо локального часу виконання. Основна проблема з паралельним виконанням моделей полягає у синхронізації різних ЛП у часі. Без синхронізації, повідомлення, надіслане одним ЛП може бути отримане іншим ЛП, запізно. Це може порушити логіку виконання всієї моделі.

Існують дві основні категорії паралельних алгоритмів моделювання, які відрізняються тим, яким чином вони організовують синхронізацію подій у різних ЛП:

1. Консервативний алгоритм запобігає виникненню непослідовності подій за допомогою алгоритму Null-повідомлень (Null Message Algorithm). Цей алгоритм використовує інформацію про час, коли ЛП надсилає повідомлення іншим ЛП та за допомогою допоміжних Null-повідомлень розповсюджує цю інформацію серед інших ЛП. Якщо ЛП знає, що отримання повідомлень протягом часу $[t, t+\Delta t]$ не очікується, то він може виконуватися протягом цього часу без зовнішньої синхронізації.

Консервативне моделювання, за певних умов, може наближатися до послідовного (уповільнення відбувається за рахунок обміну повідомлень між ЛП), якщо модель недостатньо розпаралелена, або не налагоджений обмін повідомленнями (недостатня для злагодженої роботи кількість).

2. Оптимістична синхронізація допускає появу невідповідностей при виконанні але виявляє і виправляє їх. Виправлення включає в себе відкат у попередній стан, та розсилку повідомлень для скасування всіх обмінів даними, що відбувалися протягом періоду з виникнення невідповідності. Оптимістичну синхронізацію вкрай важко реалізувати на практиці, оскільки вона вимагає періодичного збереження стану та реалізації механізмів відновлення. У будь-якому випадку, реалізація оптимістичної синхронізації потребує (на додаток до більш складного ядра виконання) напи-

сання значно складніших модулів від користувача. Оптимістична синхронізації може бути повільною у випадках виникнення частих відкатів.

Опишемо архітектуру паралельного моделювання в OMNeT++. Середовище дозволяє виконувати паралельне імітаційне моделювання без змін коду – вимагається тільки налаштування конфігурації виконання. Реалізація такого підходу суттєво використовує поняття підстановки вільних інтерфейсів при виконанні різних ЛП, що дозволяє використовувати елементи середовища імітаційного моделювання (наприклад, прямий обмін повідомленнями) і для паралельного виконання.

Архітектура – модульна та відкрита, тому вона може слугувати середовищем для дослідження паралельних обчислень.

Як уже зазначалось, OMNeT++ розділяє місця зберігання моделей та результати їх виконання. Основна причина цього полягає у тому, що кількість експериментів для одної моделі може бути значною. Експерименти можуть вимагати частої зміни параметрів, тому природним рішенням є виконання експериментів імітаційного моделювання окремо від моделі.

Відповідно до цього принципу середовище дозволяє паралельне виконання моделей без їх модифікації. Не потрібно спеціальних змін коду або топології мережі, оскільки всі параметри описуються у конфігураційному файлі запуску.

Середовище підтримує Null Message Algorithm для статичної топології з використанням затримки на каналах зв'язку для прогнозування. Також підтримується Ideal Simulation Protocol (ISP) запропонований в 2000 році [5]. ISP – це потужний дослідницький засіб для визначення ефективності алгоритмів розпаралелювання. Цей алгоритм допомагає визначити максимальну прискорення обчислень, яке може бути досягнуте для конкретної моделі у конкретному середовищі. В OMNeT++ ISP може використовуватися для оцінки працездатності Null Message Algor thm.

Для взаємодії між ЛП OMNeT++ використовує MPI [6].

Майже кожна модель може бути виконана у паралельному режимі. Однак існують певні обмеження:

- модулі можуть взаємодіяти тільки через надсилання повідомлень (прямі звернення і доступ до параметрів всередині модуля не допускаються якщо тільки вони не розташовуються на одному вузлі);
- глобальні змінні не допускаються;
- існують обмеження на надсилання повідомлень (наприклад, не можна надсилати повідомлення підмодулю іншого модуля);
- потрібно визначити затримки ланок зв'язку для роботи Null Message Algorithm;
- на сьогодні реалізована робота тільки для статичної топології.

3. Пакет моделювання черг

Один з відомих підходів до моделювання комп'ютерних і комунікаційних мереж – теорія черг [7, 8]. Моделі справжніх комп'ютерних мереж нерідко дуже складні. Однак, як правило, вони допускають декомпозицію за допомогою базового набору простих модулів або їх комбінацій.

Середовище містить бібліотеку модулів для моделювання мереж. Ця бібліотека надає базовий інструментарій для створення та дослідження моделей.

3.1. Опис бібліотеки

У системі реалізовані наступні модулі (рис. 3).

Allocate. Даний модуль розподіляє ресурси для обслуговування кожного завдання. Якщо ресурс завантажений, завдання ставиться у чергу на обслуговування.

Classifier. Класифікатор надсилає повідомлення у різні виходи в залежності від типу або пріоритету повідомлення. Якщо для даного значення вихідний портал не визначений, то повідомлення надсилається у портал «інші».

Clone. Надсилає вхідне повідомлення в усі вихідні портали. Оригінальне повідомлення надсилається у портал out[0], в інші надсилаються його копії.

Deallocate. Звільняє певну кількість ресурсу при отриманні завдання, що надсилаються у вихідний портал без змін.

Delay. Блок, що затримує повідомлення на певний час. Час затримки може задаватися випадковою величиною.

Fork. Розділяє завдання на частини по одній на кожний вихідний портал. Частини можуть бути об'єднані у початкове завдання за допомогою модуля **Join**.

Job. Повідомлення «завдання» для моделювання черг.

JobList. Представляє собою монітор всіх поточних завдань, що існують в кожний момент часу в системі.

Join. Модуль чекає коли надійдуть всі створені модулем **Fork** підзавдання. Після приходу останнього модуль заміняє їх на оригінальне завдання.

Рис. 3. Доступні для використання модулі

Merge. Модуль об'єднує декілька вхідних зв'язків. Будь-яке завдання, що надійшло через ці канали відправляється в вихідний портал.

PassiveQueue. Повідомлення мають викликатися з черги на обробку. Вихід черги має бути з'єднаний з сервером.

Queue. Черга з сервером обробки завдань.

ResourceBasedQueue. Черга, що використовує для обробки завдань ресурси з модуля **ResourcePool.** Якщо ресурси недоступні, черга чекає певний час.

ResourcePool. Модуль, що надає доступ до ресурсів одного типу. Декілька модулів типу **ResourceBasedQueue** можуть бути пов'язані з використанням цих ресурсів. У цьому разі запити на використання ресурсів буде поставлено в чергу за пріоритетністю.

Router. Надсилає повідомлення на різні виходи в залежності від алгоритму.

Server. Сервер може обслуговувати декілька вхідних черг (**PassiveQueue**), використовуючи певний описаний алгоритм.

Sink. Знищує (або утримує) повідомлення та збирає статистику (час проходу, максимальна затримка).

Source. Модуль, що генерує завдання. Як параметри можна задати кількість завдань, час початку та закінчення, інтервали між завданнями (у вигляді числа або випадкової величини з заданим розподілом). Також задається назва, тип і пріоритет завдань.

SourceOnce. Модуль, що генерує задану кількість завдань один раз у певний момент часу. Таке джерело корисне для внесення завдань у замкнуту мережу або при дослідженні поведінки системи в умовах скачкоподібного зростання завдань.

3.2. Приклад моделювання системи обробки різнотипних пакетів

Розглянемо узагальнення моделі мережі, яка теоретично досліджувалась у [9, 10].

Модель обробки різнотипних пакетів в графічному вигляді показана на рис. 4 і складається з наступних елементів: користувачі, класифікатор пакетів, модуль обробки пакета для виконання, сховище спільних ресурсів, буфер сервера, сервер, вихідний канал.

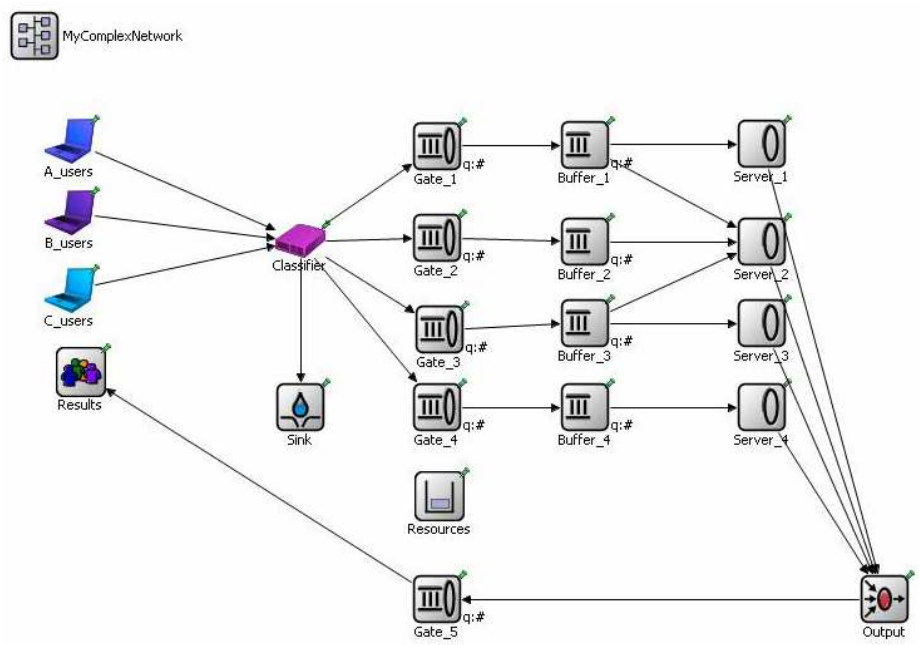

Рис. 4. Модель обробки різнотипних пакетів

Система, що розглядається призначена для обробки запитів від *користувачів* (A_users, B_usres, C_users на рис. 4). Користувачі генерують пакети з запитами, які мають різні пріоритети (від 0 – найвищій, до 3 – найнижчий). Пакети надходять на вхід *класифікатору* (Classifier), який проводить аналіз їх пріоритетності. В залежності від конкретного значення пакет буде надісланий на відповідний *модуль обробки* (Gate_1 – Gate_4) для підготовки до виконання або відкинутий як невідповідний (Sink). Ця процедура використовує певний спільний ресурс (наприклад пропускну здатність вхідного каналу або частину пам'яті). Спільний для всіх модулів обробки ресурс показаний на рис. 4 (Resources). До виконання запиту на серверах (Server_1 – Server_4) пакети очікують у відповідному буфері (Buffer_1 – Buffer_4). Після виконання пакети з результатами потрапляють у модуль обробки (Gate_5) та надсилаються користувачам.

Опишемо аналітичну модель цієї системи користуючись формалізмом керованих мереж [7 – 9].

Позначимо частоту генерації пакетів як α_i, де індекс це пріоритет пакета $i \in I$, $I = \{0,1,2,3\}$. Довжину черги в модулі обробки позначимо $q_i(t)$. Тоді процес обробки буде описуватися потоковою моделлю:

$$\dot{q}_i(t) = \alpha_i - u_i(t),\ i \in I,$$

де керування $u_i(t)$ описує схему черговості обробки пакетів. При цьому на керування накладається обмеження $\sum_{i=0}^{3} u_i(t) \le \mu$. Аналогічно позначимо затримки у буферах $p_i(t)$ та отримаємо модель:

$$\dot{p}_i(t) = u_i(t) - Bv(t),\ i \in I,$$

де матриця B описує взаємозв'язки між буферами та серверами (які можуть бути, як показано на рис. 3 зв'язувати декілька буферів і один сервер). Для моделі, що розглядається матриця B має вигляд:

$$B = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Керування $v = (v_1,...,v_4)$, $|v_i| \le \nu$ описує обробку пакетів серверами.

Функції обробки пакетів $u(t)$, $v(t)$ можуть задаватися по різному. Розглянемо один з розповсюджених підходів. Обробка пакетів на вході здійснюється по пріоритетності, тобто в першу чергу при наявності ресурсів обслуговуються більш пріоритетні пакети. В результаті пакети найвищого пріоритету обслуговуються майже неперервно, пакети найнижчого – тільки при наявності ресурсу.

Виконаємо моделювання для ситуації коли ресурсів більше ніж необхідно для обробки всіх пакетів і при цьому частота генерації пакетів всіх користувачів дорівнює α. Кожен користувач генерує пакети різного пріоритету з рівною ймовірністю. Тому можна оцінити частоту генерації пакетів певного пріоритету одним користувачем як четверту частину від загальної. Отже пакети кожного пріоритету подаються на вхід системи з частотою $\frac{3\alpha}{4}$.

На наступних рис. 5 – 7 показана залежність довжини черги $q_i(t)$ від часу при різних значеннях μ в таблиці.

Таблиця

№	Значення	Рисунок
1	$\frac{\mu}{4} > \frac{3\alpha}{4}$	Рисунок 5
2	$\frac{\mu}{4} \approx \frac{3\alpha}{4}$	Рисунок 6
3	$\frac{\mu}{4} < \frac{3\alpha}{4}$	Рисунок 7

Перший випадок відповідає ситуації, коли система обробляє пакети швидше ніж вони встигають накопичуватися у чергах. В цьому разі статистичні коливання не призводять до суттєвого зростання величини черги.

В разі, коли $\frac{\mu}{4} \approx \frac{3\alpha}{4}$, черги не будуть сильно зростати, однак статистичні коливання будуть накопичуються (рис. 6).

Нарешті рис. 7 ілюструє ситуацію, коли система не встигає обробити частину пакетів, внаслідок чого черги зростають.

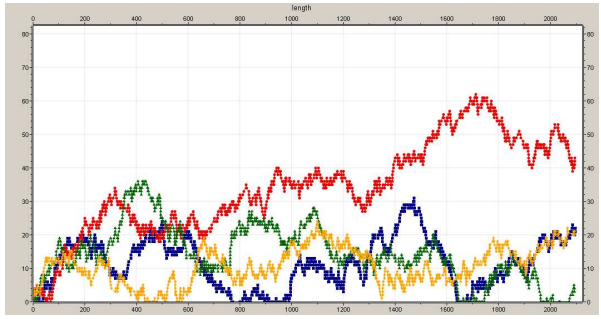

Рис. 5 Стабільна робота системи

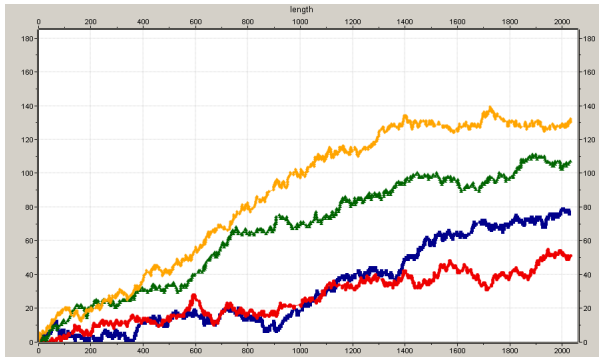

Рис. 6 Повільне накопичення помилок

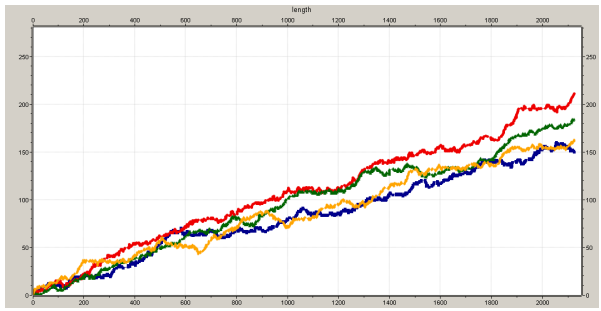

Рис. 7. Зростання черг

Стратегії $v_i(t)$ визначаються вимогою зменшення найкоротшої черги. Іншими словами:

$$\dot{p}_1 = u_1 - v_1 - v_2 \cdot shortest(p_1(t)),$$
$$\dot{p}_2 = u_2 - v_2 \cdot shortest(p_2(t)),$$
$$\dot{p}_3 = u_3 - v_3 - v_2 \cdot shortest(p_3(t)),$$
$$\dot{p}_4 = u_4 - v_4,$$

де функція $shortest(p_i(t))$ дорівнює одиниці, якщо $p_i(t)$ – найкоротша черга серед $p_1(t)$, $p_2(t)$, $p_3(t)$.

Розглянемо поведінку системи при атаці. Дії нападника будемо моделювати наступним чином: нападник генерує 50 пакетів-запитів ідентичних до запитів

користувачів і надсилає їх у систему імпульсно, тобто за короткий час. Будемо вважати, що пріоритет всіх пакетів атаки однаковий. Тоді в залежності від конкретного значення пріоритету поведінка системи буде змінюватися. Графіки черг буферів $p_i(t)$ системи показані на рис. 8 – 12. Як видно з рисунків найбільш потужний вплив на систему мають атаки з пріоритетами 0 і 1.

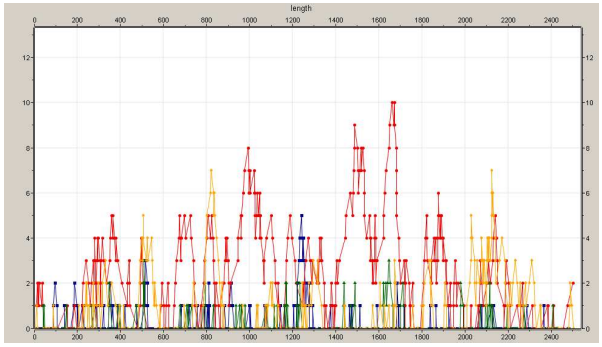

Рис. 8. Поведінка системи за відсутності атаки

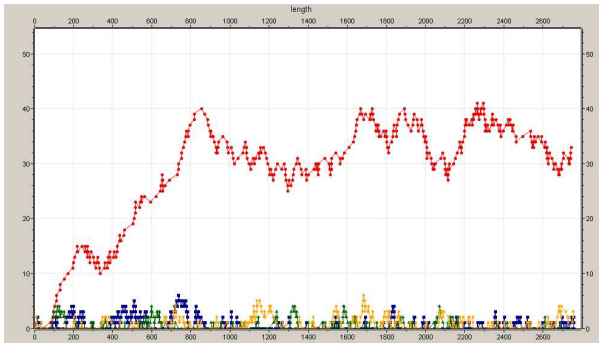

Рис. 9. Пріоритет пакетів атаки дорівнює 0

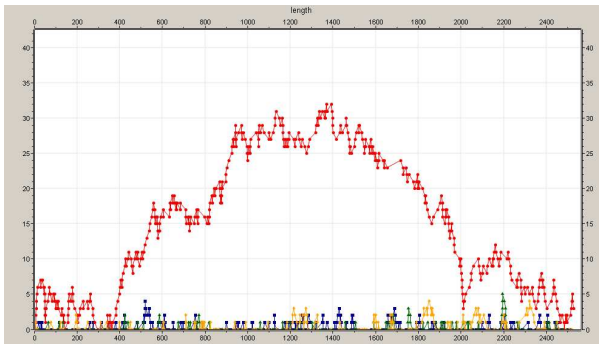

Рис. 10. Пріоритет пакетів атаки дорівнює 1

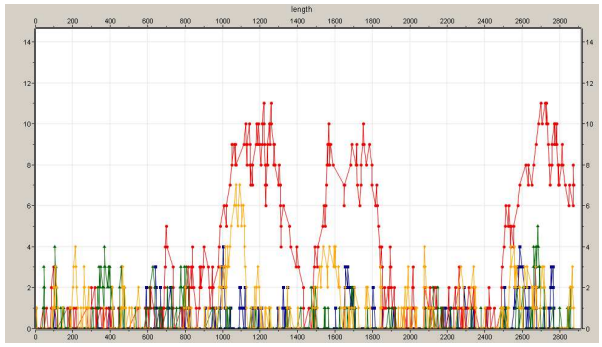

Рис. 11. Пріоритет пакетів атаки дорівнює 2

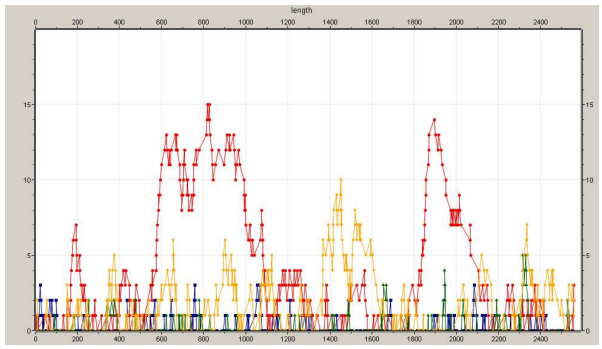

Рис. 12. Пріоритет пакетів атаки дорівнює 3

Висновки

У роботі аналізується вибране авторами для побудови імітаційних моделей конфліктно керованих систем середовище дискретного імітаційного моделювання OMNeT++. Особливу увагу приділено підходу моделюванню мереж за допомогою бібліотеки черг.

У вибраному середовищі побудовано імітаційну модель та проведено моделювання системи обробки різнотипних пакетів за умов штатної роботи та за наявності конфлікту.

1. *Ігнатенко О.П., Цицкун Д.В.* Противодействие атакам на отказ в сети Интернет: выбор среды моделирования // Проблеми програмування. – 2008. – № 2-3. – С. 579 – 586.
2. http://www.omnetpp.org/
3. *Bagrodia R.L. and Takai M.* Performance evaluation of conservative algorithms in parallel simulation languages // IEEE

Transactions on Parallel and Distributed Systems. – 2000. – N 11(4). – P. 395 – 414.
4. http://www.mpi-forum.org
5. *Андон П.І, Ігнатенко О.П.* Протидія атакам на відмову в мережі Інтернет: концепція підходу // Проблеми програмування. – 2008. – № 2-3. – С. 564 – 574.
6. *Андон П.І., Ігнатенко О.П.* Атаки на відмову в мережі Інтернет: опис проблеми та підходів щодо її вирішення. – Київ, 2008. – 50 с. – (Препр. / НАН України. Ін-т програмних систем; 2008 – 2).
7. *Meyn S.* Control Techniques for Complex Networks. – Cambridge University Press, 2007. – 582 p.
8. *Bertsekas D.* Network optimization: c ntinuous and discrete models. – Athena Scientific, Belmond, 1998. – 270 p.
9. *Ігнатенко О.П.* Моделювання обслуговування користувачів у мережі з одним сервером за умов конфлікту // Проблеми програмування. – 2009. – № 3. – С. 66 – 72.
10. *Ігнатенко О.П.* Конфліктна задача взаємодії двох гравців у відкритому інформаційному середовищі // Проблеми програмування. – 2009. – № 2. – С. 56 – 65.

Отримано 28.09.2009

Про авторів:

Ігнатенко Олексій Петрович,
кандидат фізико-математичних наук,
старший науковий співробітник,
докторант.

Цицкун Дмитро Володимирович,
аспірант Інституту програмних систем
НАН України.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, Київ 187,
Проспект Академіка Глушкова, 40,
Тел.: 526 6025.
e-mail: o.ignatenko@gmail.com