Моделиране и симулации на динамичните характеристики на системни мрежи чрез симулатор OMNeT++

1. Актуалност на проблема

1.1. OMNeT++ – общи сведения и основни указания за използване

OMNeT++ е модулна симулационна среда, включваща специфични библиотеки (simulation framework and library). По същество тя представлява набор от софтуерни инструменти и библиотеки, които подпомагат разработката на симулационни модели. Най-често с OMNeT++ се разработват модели на компютърни мрежи и протоколи, но продуктът би могъл да се използва и за изготвяне на модели с по-широко предназначение, като жични и безжични комуникационни мрежи, вградени мрежи върху чип (NoCs) и други. Допълнителна функционалност, специфична за дадена сфера, като симулиране на интернет протоколи, поддръжка за сензорни мрежи, оптични мрежи и други, може да бъде имплементирана като отделен самостоятелен проект и да бъде вградена, т.е. преизползвана в различни проекти.

OMNeT++ включва базирана на Eclipse графична среда за разработка (Graphical IDE) и някои допълнителни инструменти за улеснение работата на разработчиците. Съществуват и разширения за симулиране в реално време, емулиране на мрежи, възможност за използване на допълнителни програмни езици като Java и C#, възможност за интегриране на бази данни и още много допълнителни функции.

Цялостният процес на създаване на симулация в OMNeT++ е представен на фиг. 1, като е необходимо да се направят следните уточнения: важно е да се отбележи, че по същество процесът на симулиране и събиране на данни е цикличен, тъй като обикновено се налага модифициране на модела и повторното му компилиране и изпълнение, при получаването на нови резултати.

Имплементирането на основни модули, структурното дефиниране на модела, групирането на елементарни модули в сложни, настройването на модела, построяването на модела и извличането на резултати се създават от дизайнера на симулацията.

1. Създава се симулационен модел, изграден от отделни компоненти(модули), които комуникират помежду си посредством обмен на съобщения. Модулите могат да бъдат вложени, т.е. множество „елементарни модули“ могат да бъдат групирани в един по-голям модул, наречен „сложен модул“.

2. Дефинира се структурата на модела, като се използва езика NED – описателен език, използван в OMHeT++ приложения. Възможно е дефинирането на модела да се осъществи посредством графичния редактор на OMHeT++ за езика NED-GNED, или да се отвори като текстов файл, в който да се отразят необходимите промени.

3. Активните компоненти на модела (елементарните модули) трябва да бъдат имплементирани (на C++), като се използва библиотека за симулации, предоставена от ядрото на платформата(simulation kernel and class library).

4. Създава се подходящ конфигурационен файл с наименование „omnetpp.ini“, който съдържа освен конфигурацията също и някои примери на модела. С подходяща конфигурация би могло параметрите на модела да се изменят за всяко следващо изпълнение на модела (всеки следващ „run‘‘).

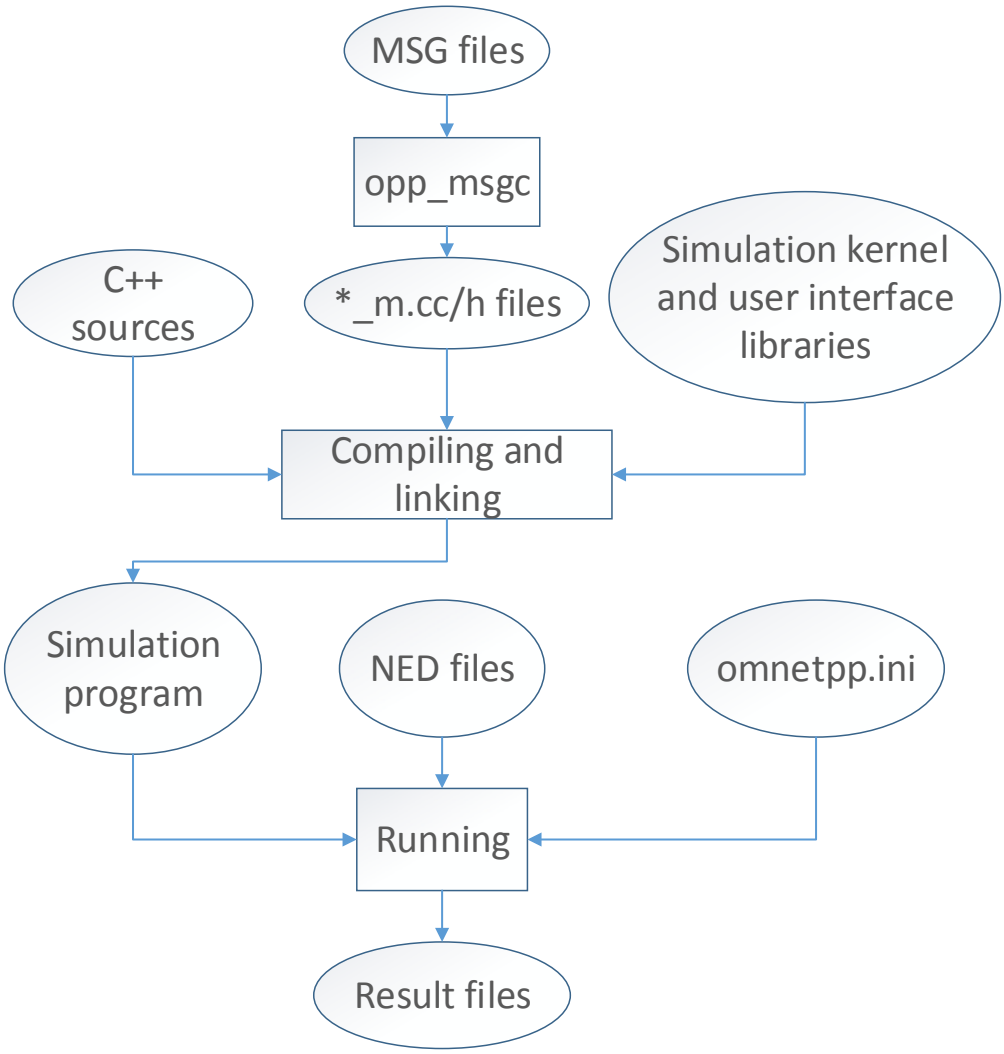
5. Построява се модела (build) и се стартира. При изпълнение на тази стъпка обектните файлове, създадени след компилиране на потребителския код (от стъпка 3) се свързват с необходимите библиотеки, предоставено от ядрото и се стартира изпълнение в един от потребителските интерфейси, предоставени от OMHeT++. Към момента съществува възможност за изпълнение от командния ред (command line, batch) или чрез графичния интерфейс (interactive graphical interface).

6. Резултатите от симулацията се записват в изходни векторни и/или скаларни файлове. За визуализиране на резултатите могат да се използват инструментите Plove или Scalars. За по-подробно анализиране на резултатите биха могли да се използват допълнителни програмни продукти като: R, Octave, Matlab, Gnumeric и други.

Фиг. 1. Фази на симулационния процес в OMNeT++

1.2. Създаване на модел в OMNeT++

Процесът на създаване, настройка и изпълнение на модел, както и събирането на данни, е представен на фиг. 2.



Фиг. 2. Създаване на симулационен модел в OMNeT++

Един завършен модел в OMNeT++ се състои от следните компоненти:

- *Описание на топологията посредством NED езика (.ned файлове)* – описват структурата на модулите, с техните параметри и т.н. Както бе отбелязано по-рано, NED файловете могат да бъдат създадени чрез текстови редактор или чрез предоставения от OMNeT++ графичен и текстови редактор GNED;
- *Дефиниране на съобщения (.msg файлове)* – могат да се дефинират различни типове съобщения и да се добавят различни полета с данни като съдържание към съобщенията. OMNeT++ транслира дефинициите на съобщения в C++ класове.
- *Сорскод на елементарните модули* – C++ файлове с разширение *.h* и *.cc*;

Симулационната среда предоставя следните компоненти:

- *Симулационно ядро* – в него се съдържа кода, който управлява симулацията, както и библиотеката със симулационни класове (simulation class library). Написан е на C++ и се компилира като споделена (shared) или статична (static) библиотека.
- *Потребителски интерфейс* – потребителските интерфейси в OMNeT++ се използват за да се онагледи симулацията по време на изпълнението й, за откриване на грешки и др. Интерфейсите са написани на C++ и се компилират като библиотеки.

Симулационните модели се построяват от изброените по-горе компоненти. Първо *.msg файловете* се свеждат до C++ код посредством *opp_msgc програмата*. След това всички C++ сорсове се компилират и свързват (compile and link) със симулационното ядро и библиотеката на потребителския интерфейс, за да се получи изпълним файл на симулацията или споделена библиотека.

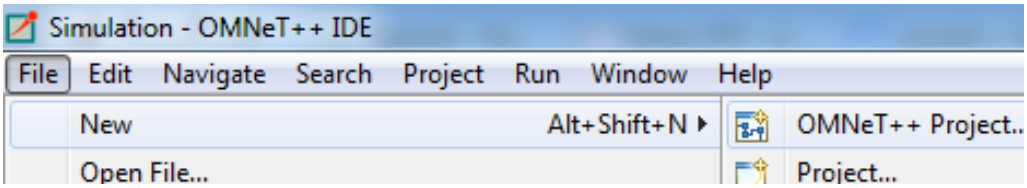
Когато симулацията се стартира, *.ned файловете,* съдържащи топологията на модела, се зареждат динамично в текстовата им форма. Следва прочитане на конфигурационния файл (обикновено този файл се казва omnetpp.ini), съдържащ указание за начина, по който ще се изпълни симулацията, стойности за параметрите на модела и др.

Резултатът от симулацията се записва в изходящи файлове, които са най-често изходни векторни и скаларни файлове, съдържащи последователността на събитията и др., както и евентуално изходящи файлове, съдържащи информация, поискана от потребителя.

Средата за разработка на OMNeT++ (OMNeT++ IDE) съдържа множество инструменти за анализ на изходните файлове от симулацията, като визуализирането им е възможно и посредством друг софтуер, пригоден за визуализация на текстови данни, като графични, таблични и т.н.

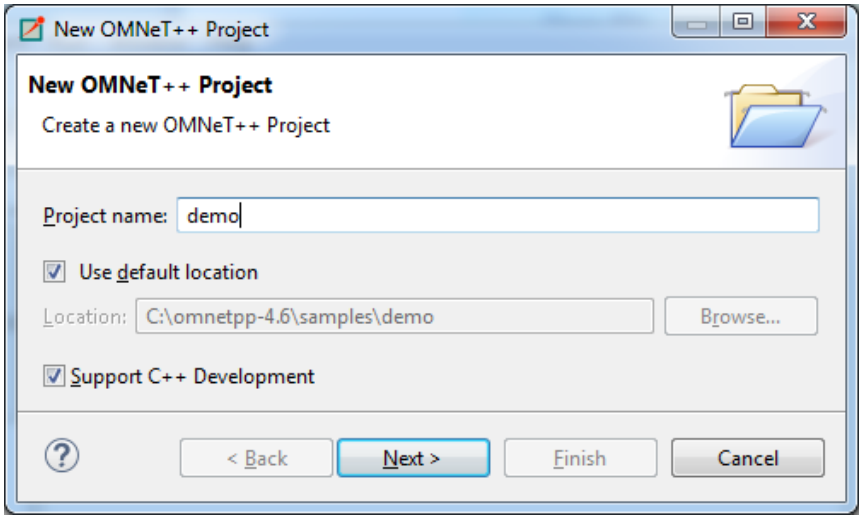
С цел начално запознаване със средата OMNeT++ се препоръчва създаването на примерен проект по стъпките описани в следващите екрани.

Създаване на нов OMNeT++ проект.



Фиг. 3. Създаване на нов OMNeT++ проект

Новият проект се задава с име „demo“ и се избира да е от тип „Empty“.

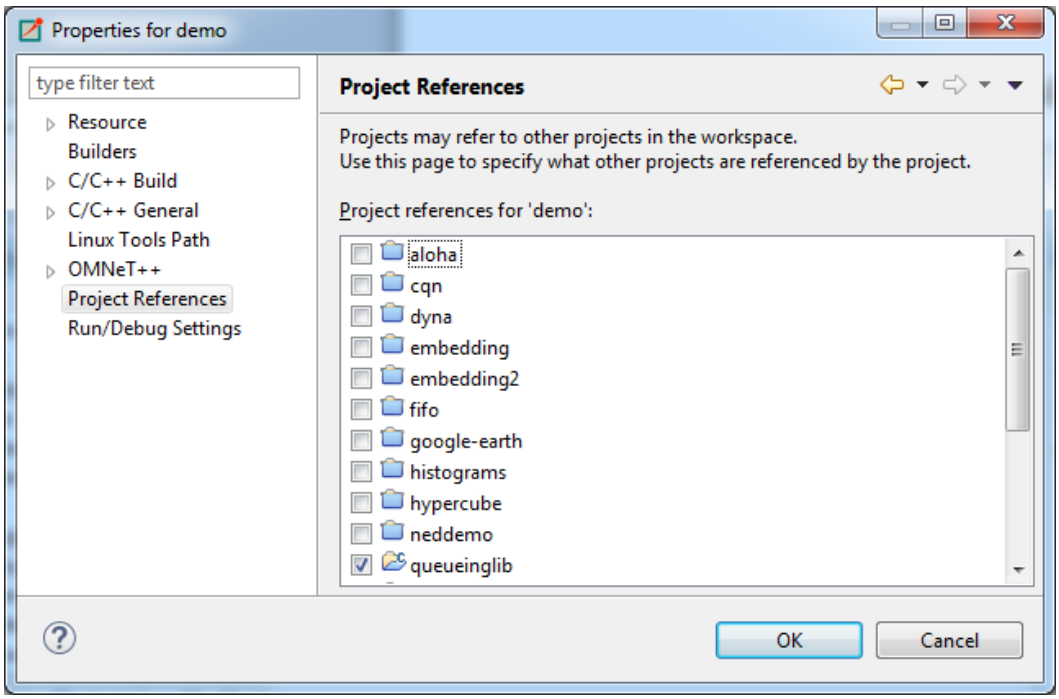


Фиг. 4. Избор на име за новия проект

Преизползване на съществуващи модули.

Проектът „demo“ ще симулира мрежа от опашки, за чието изграждане се използват готови модули от проекта „queueinglib“, който се намира в папката „samples“. Ако не наличен, се добавя от File -> Import -> Existing Projects into Workspace.

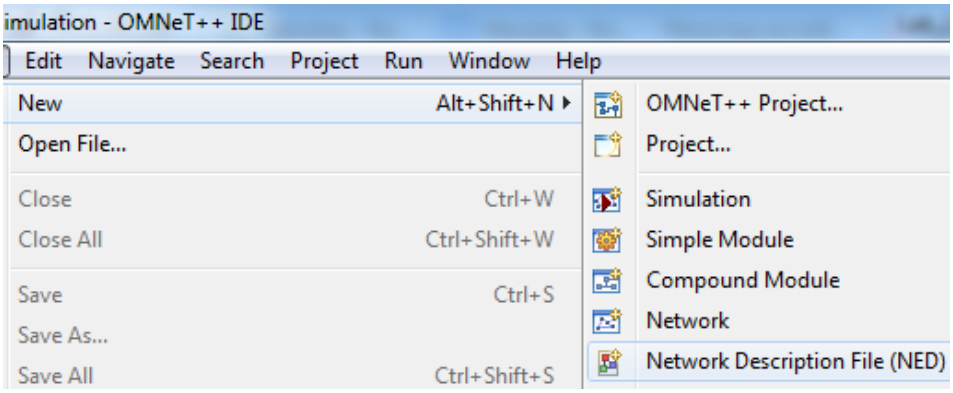
След като проектът „queueinglib“ е наличен в работното пространство е необходимо да бъде добавен към проекта „demo“. Тези разширени настройки на проекта се правят от менюто „Properties“, след което се избира „Project Preferences“.



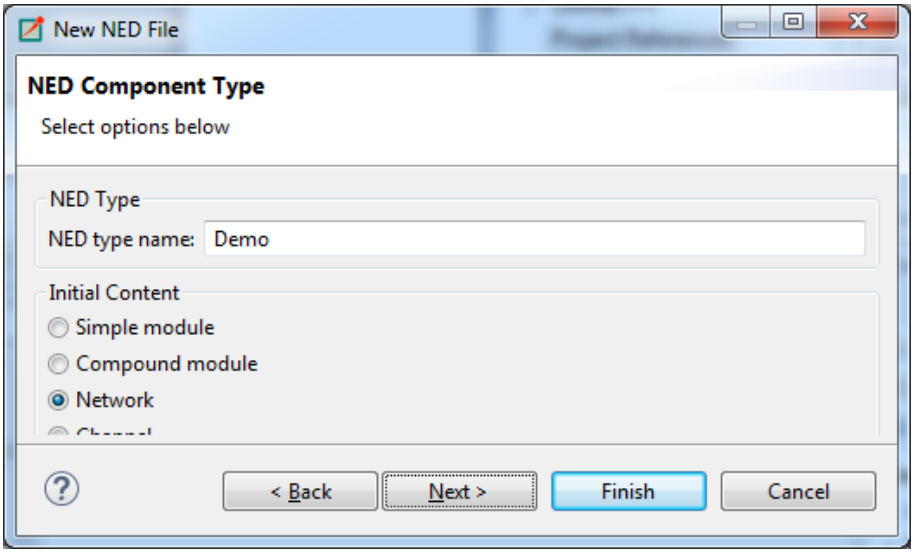
Фиг. 5. Добавяне на външен проект

Създаване на нов „Network Description file (NED)“ файл.

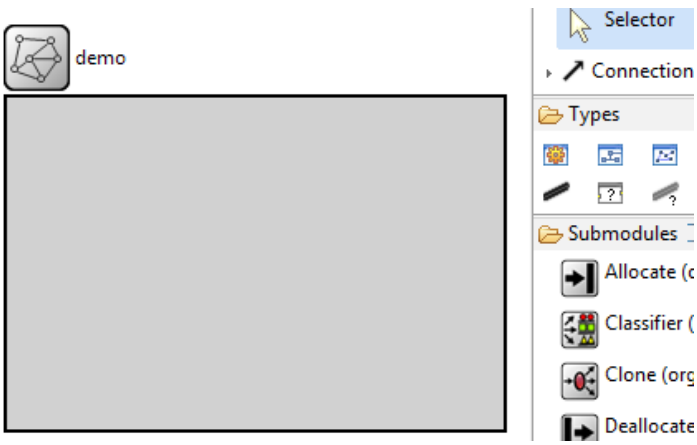
Този тип файлове описват модела, който ще бъде симулиран. Именувайте NED файла „demo.ned“. Изберете „NET file with one item“ от тип „Network“.



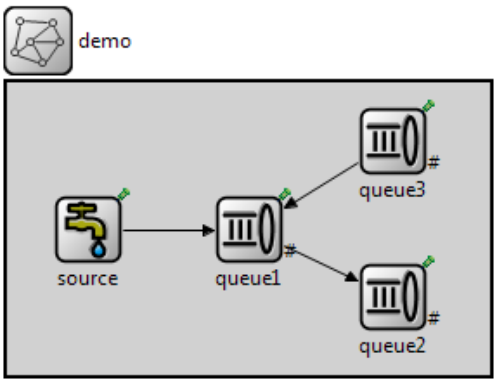
Фиг. 6. Създаване на нов NED файл



Фиг. 7. Създаване на нова мрежа



Фиг. 8. Създаване на нова мрежа чрез използване на модулите (submodules)



Фиг. 9. Създаване на мрежата в графичен режим

Мрежата може да бъде редактирана и в текстови режим (Source).

Добавете последната връзка, която прави мрежата циклична, като в текстови режим използвате „Content Assist“ от десен бутон или Ctrl + Space.

```
            @display("p=223,124");
        }
        source: Source {
            @display("p=47,85");
        }
    connections:
        source.out --> queue1.in++;
        queue1.out --> queue2.in++;
        queue3.out --> queue1.in++;
        queue2.out --> queue3.in++;

}
```

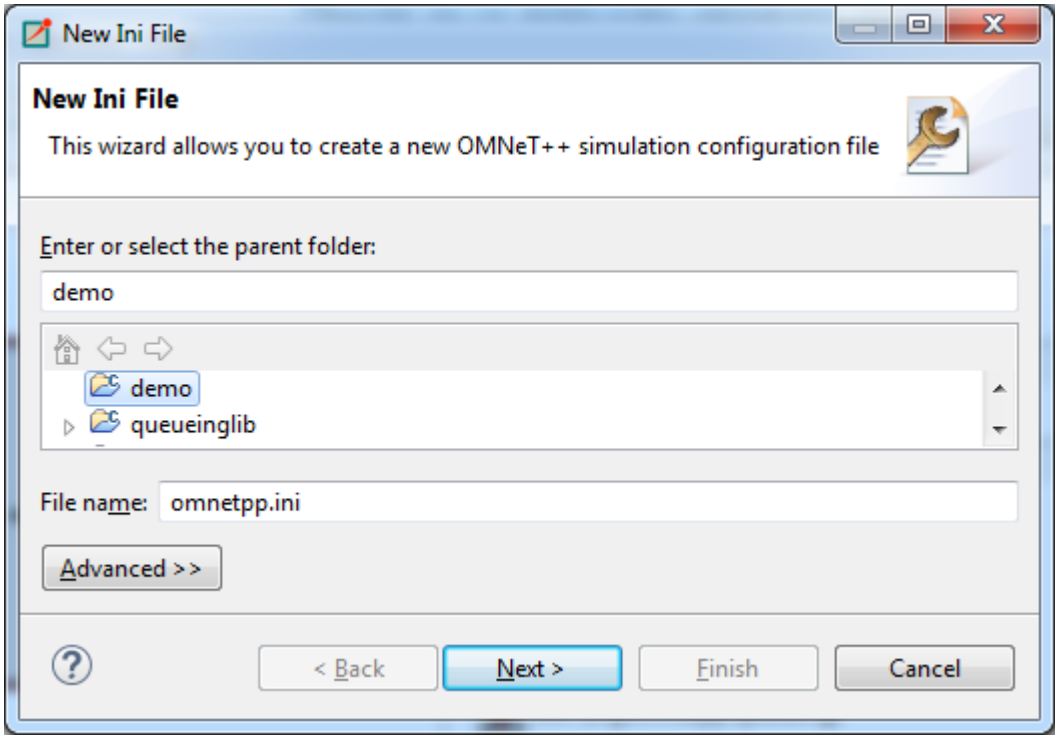
Design Source

Фиг. 10. Редактиране на мрежата в текстови режим

Преди да се използва, мрежата трябва да бъде конфигурирана.

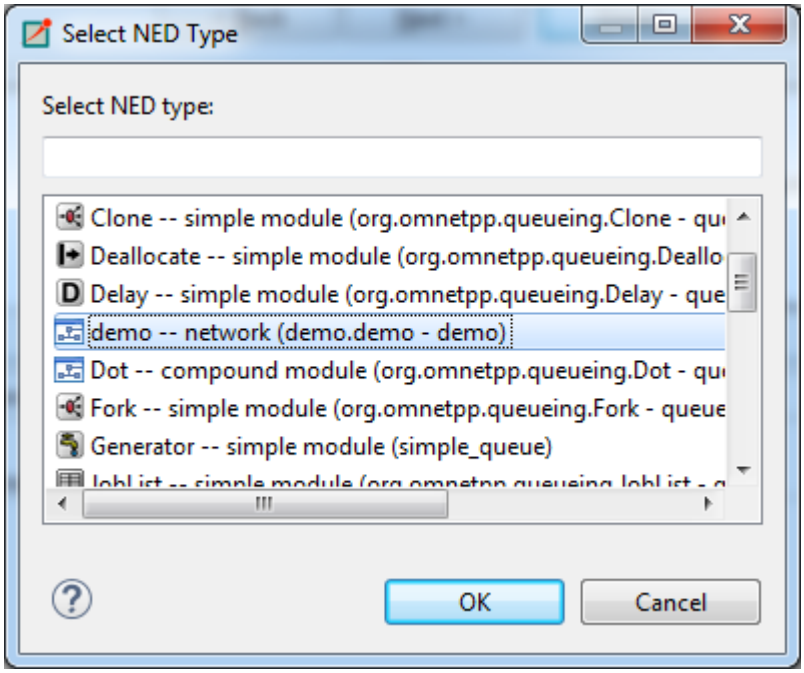
Както беше пояснено на фиг. 1, конфигурационните параметри се описват във файла „omnetpp.ini“.

Изберете New -> Initialization File (ini) и след това името на проекта.



Фиг. 11. Създаване на конфигурационен файл

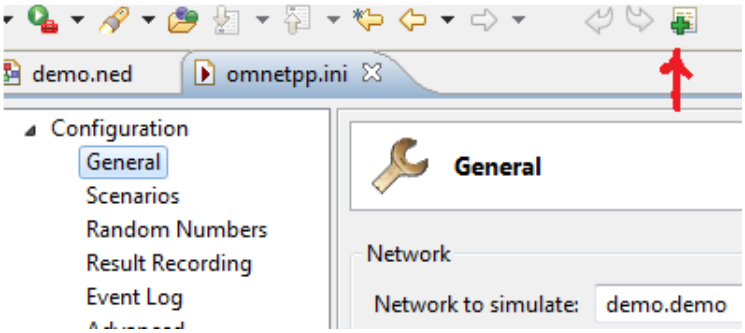
Изберете типа на компонента, за който се отнася конфигурационния файл. В случая това е мрежата „demo“.



Фиг. 12. Избор на модул

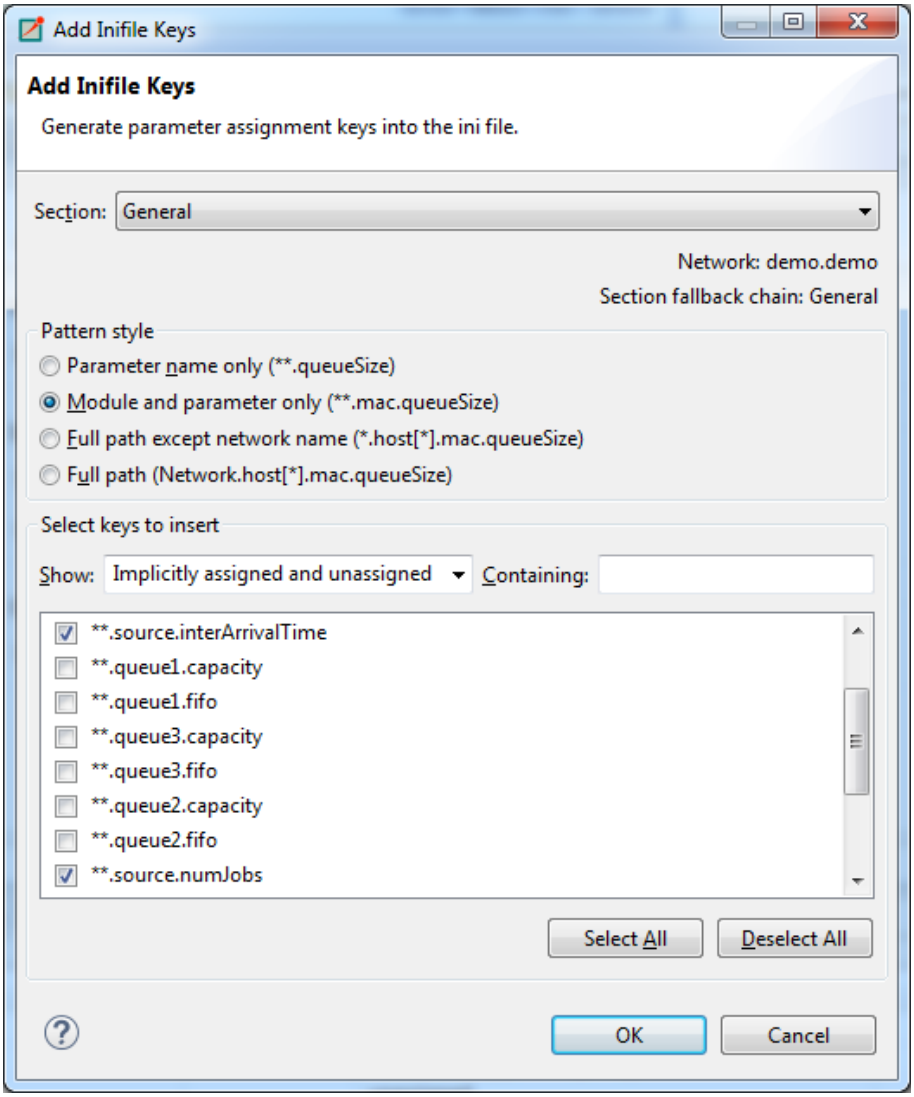
Необходимо е да се дадат стойности на параметрите.

Диалоговият прозорец „Add Inifile Keys“ показва всички параметри (Keys), за които не е зададена стойност.

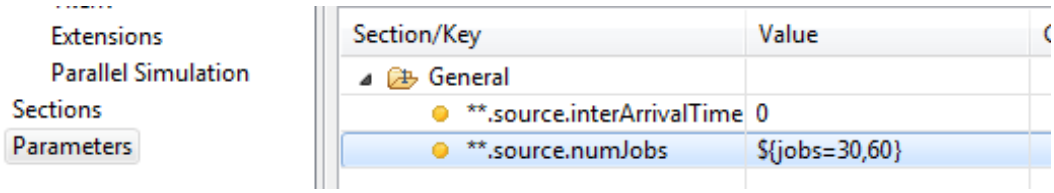


Фиг. 13. Добавяне на липсващи стойности на параметрите

Изберете „Deselect All“ и маркирайте само „source.interArrivalTime“ и „source.numJobs“.



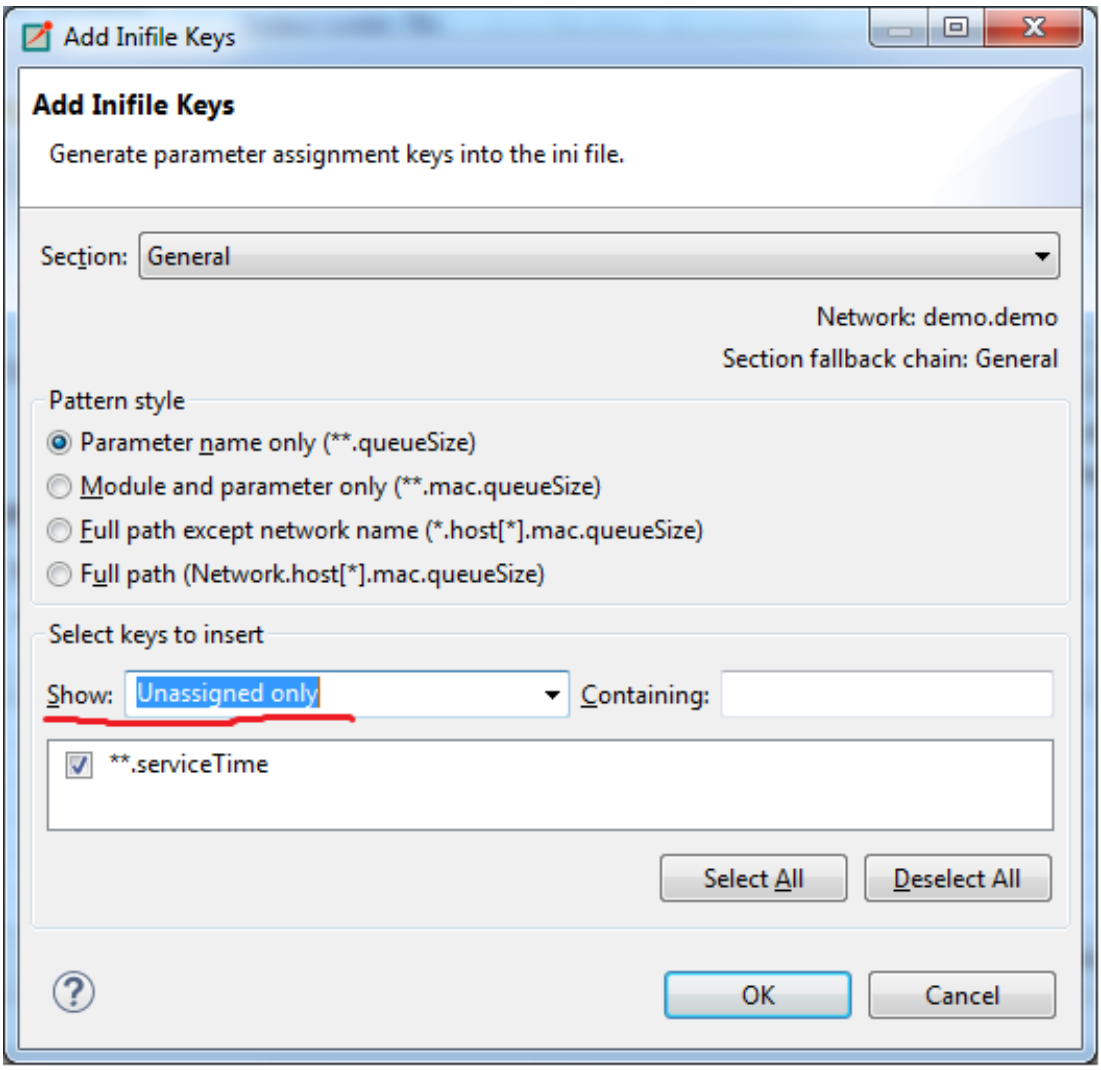
Фиг. 14. Избор на конфигурационни параметри



Фиг. 15. Въвеждане на стойности

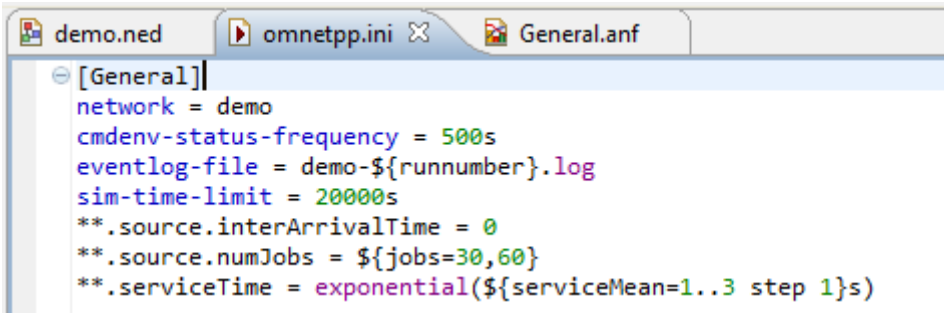
Моделът трябва да се стартира при 30 и 60 задачи. Синтаксисът ${...}позволява това да се опише директно в конфигурационния файл, като „${jobs=30,60}“.

Добавете времето за изпълнение за всички заявки, като изберете „Parameters name only“ и показване на параметрите, които не получават стойност по подразбиране.



Фиг. 16. Въвеждане на времето за изпълнение на всички заявки

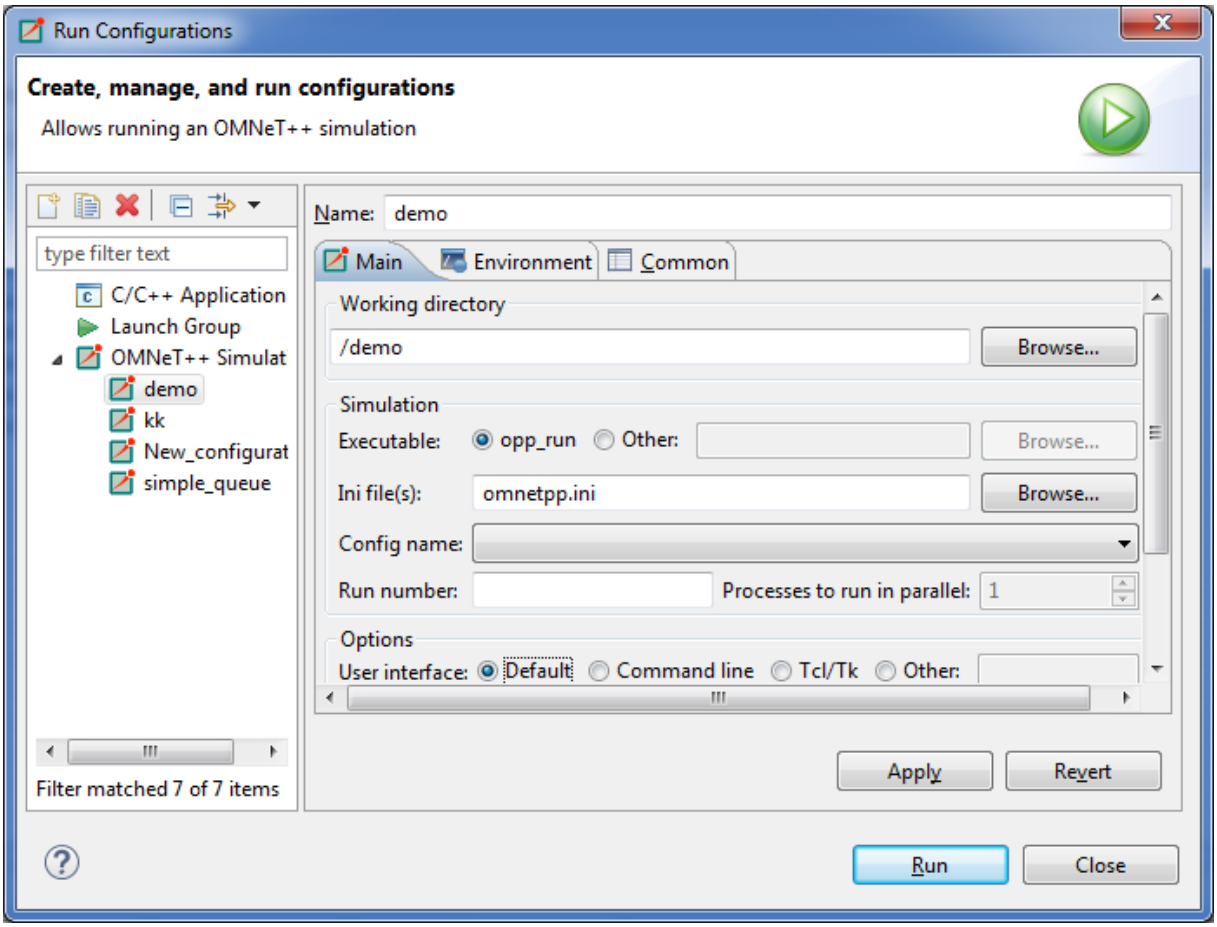
След това в „Event log“ въвеждаме „demo-${runnumber}.log“. В полето „Simulation time limit“ въвеждаме 20000s, а в Cmdenv в полето „Status frequency“ въвеждаме 500s.



Фиг. 17. Редакция на параметрите на omnetpp.ini файл

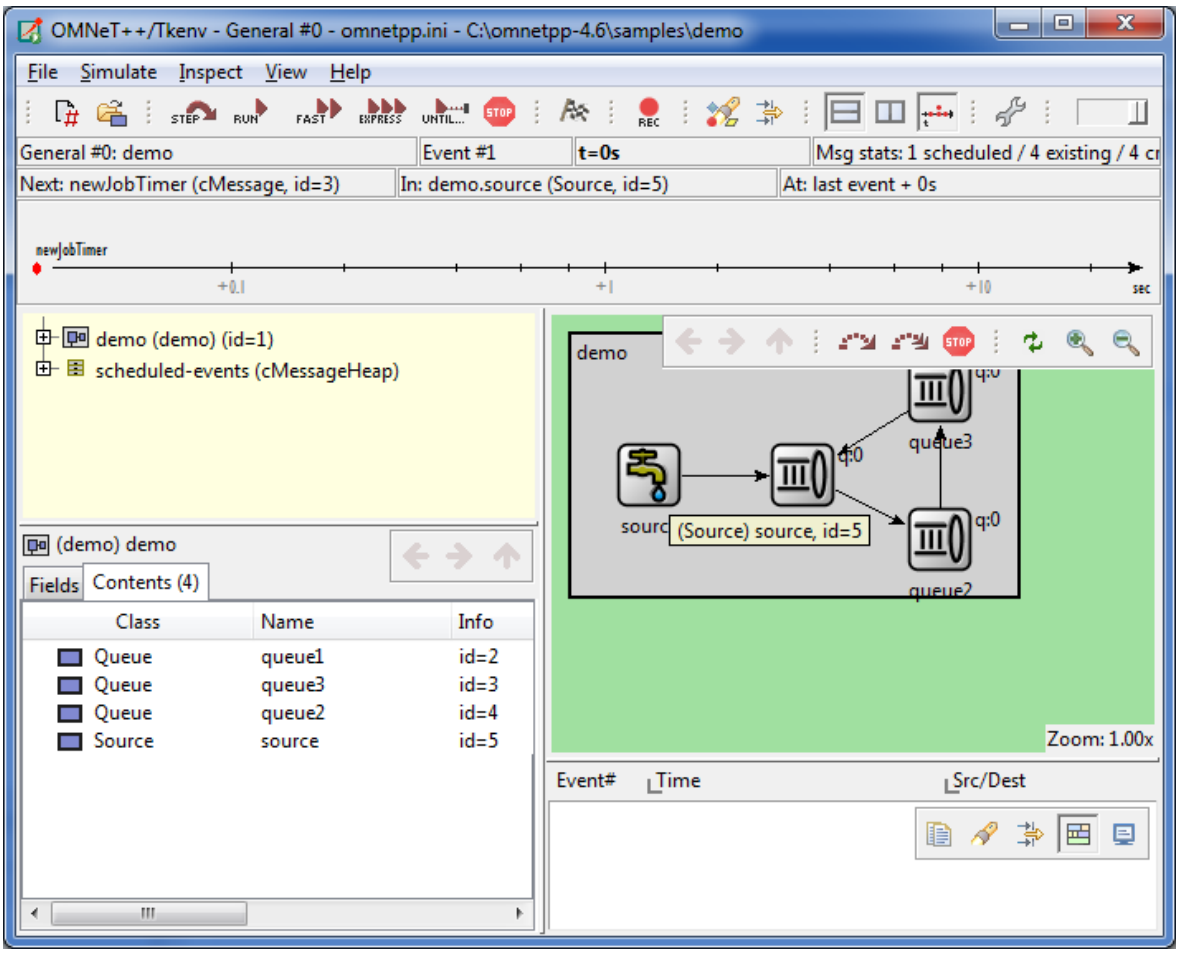
За да започнем симулация избираме:

Run As -> Run Configuration …



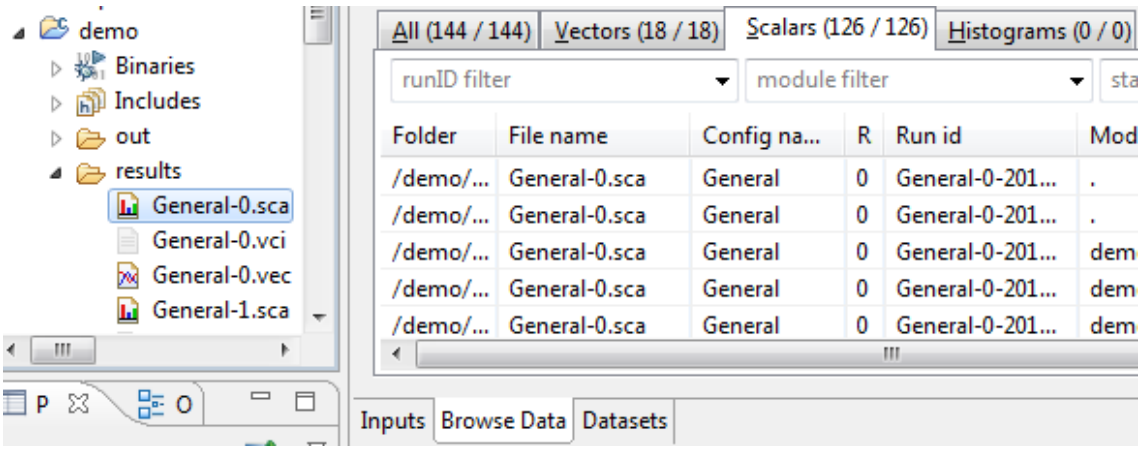
Фиг. 18. Конфигуриране и стартиране на симулация

Избираме Run, което при избран „User interface: Default“ се стартира прозорец, от който може да се управлява симулацията.



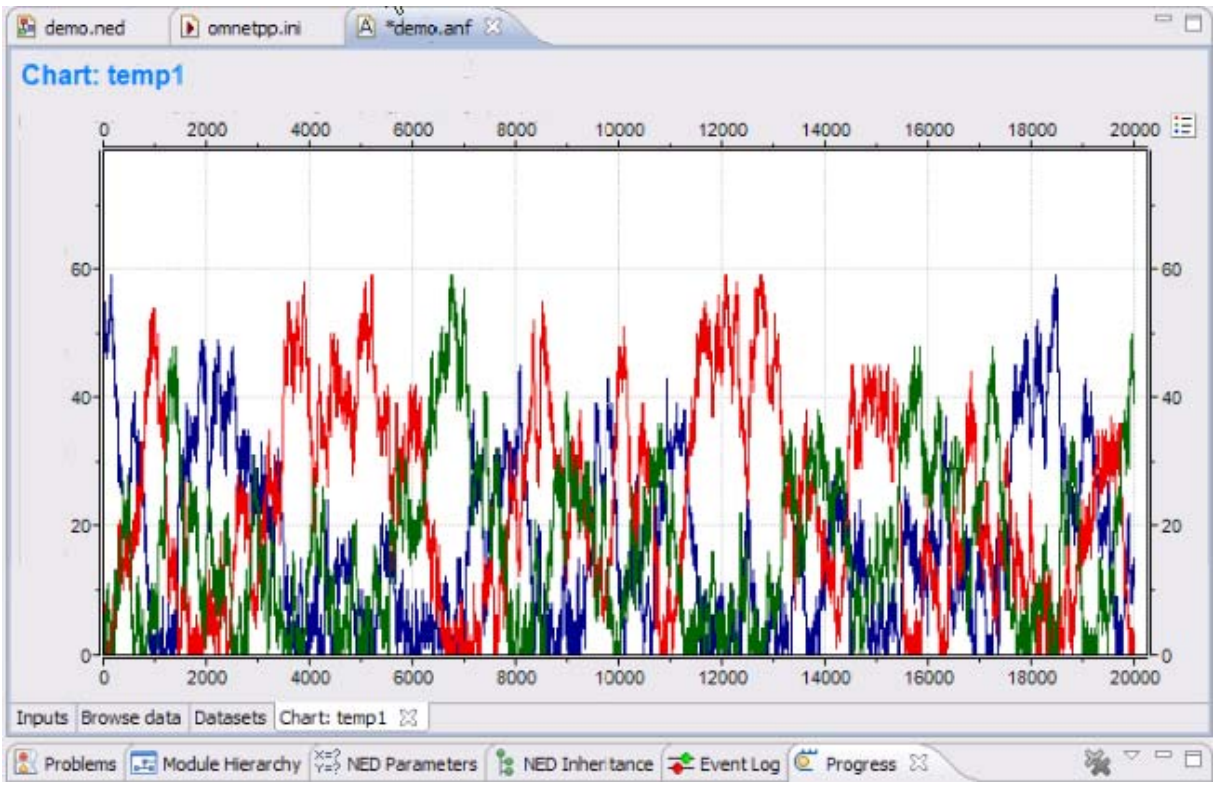
Фиг. 19. Стартиране и управление на симулацията

Изходните данни се записват в папката result на проекта, като чрез двойно кликване върху всеки от тях се създава Analysis File (anf).



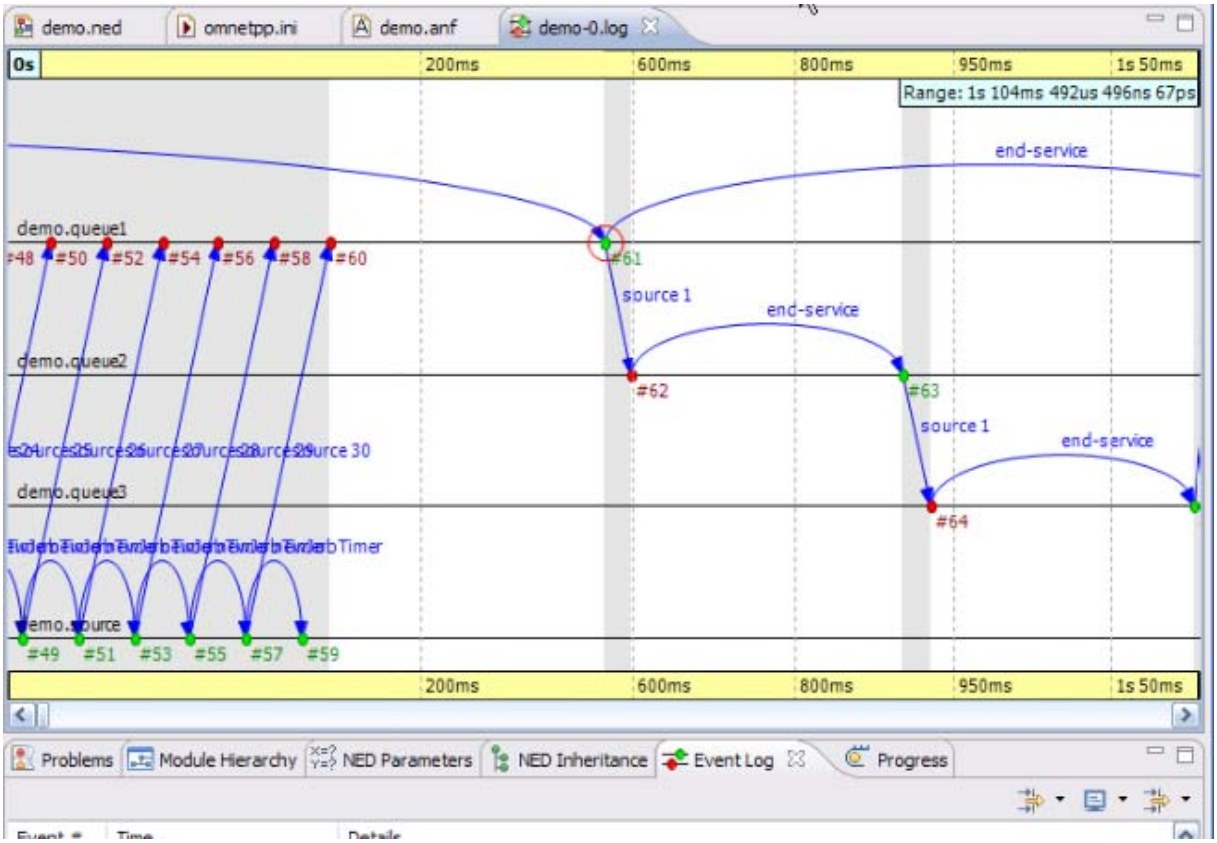
Фиг. 20. Анализ на резултатите

При стартиране на симулацията, тя получава уникален идентификационен номер (Run ID). В раздел „Browse Data“ са показани данните, които могат да се визуализират, като първо се селектират с десен бутон „Plote“.



Фиг. 21. Графика на изходящите резултати

Разгледайте log файловете в проекта.

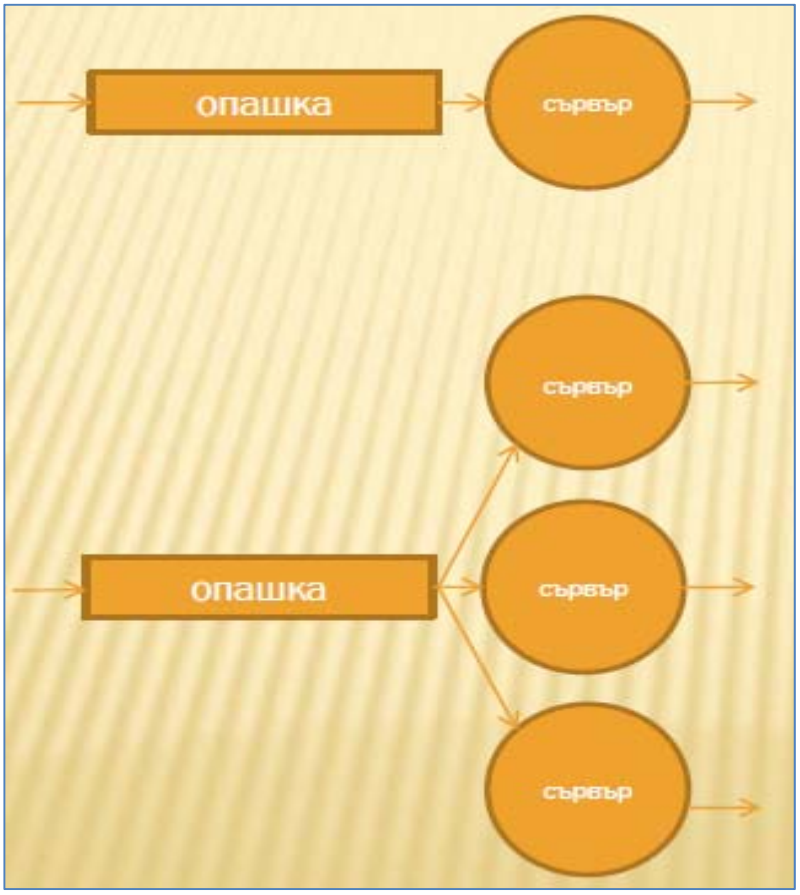


Фиг. 22. Log файл на симулацията

2. Цел и задачи на експериментите

Задачата е да се моделира и симулира системата „генератор на задачи - опашка - сървър“ в среда на OMNeT++ при различни разпределения на трафика. Да се снемат данните от проведените симулации за средното закъснение и броят чакащи заявки и да се построят графики, на базата на които да се направи анализ на симулационните данни.

Кратко описание на изходната постановка:



Фиг. 23. Система „генератор на задачи - опашка - сървър“

Теорията на масовото обслужване (теорията на опашките) – основни параметри на изходната постановка: $E = \lambda T$, където:

E – брой заявки, чакащи на опашката;

λ – интензивност на входящите заявки (бр./единица време);

T – средно време на престой на заявките в системата.

Разпределения на трафика:

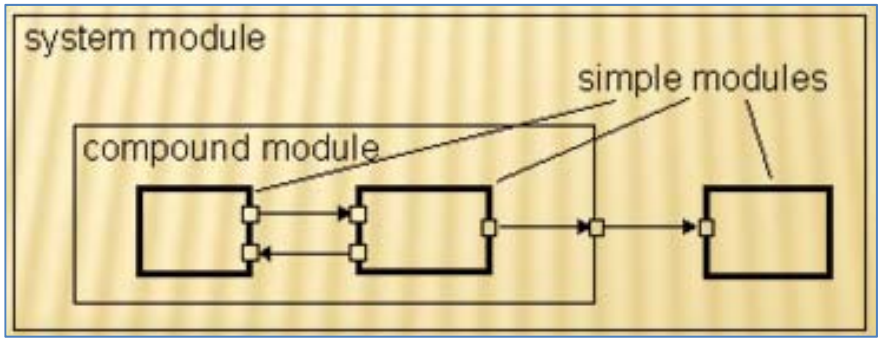
- M (Markovian) – процес с експоненциално разпределение;
- D (Degenerative) – процес с детерминистично разпределение (или на фиксирани интервали);
- E (Erlang) – процес с Ерлангово разпределение;
- G (General) – процес с общо разпределение.

За *D (дисциплина на опашките)* са валидни следните означения:

- FIFO (First-In, First-Out) – първи влязъл - първи излязъл;
- LIFO (Last-In, First-Out) – последен влязъл - първи излязъл;
- SIRO (Service In Random Order) – обслужване без да има значение реда на постъпване в опашката;
- PNPN (Priority service) – обслужване с приоритети.

Кратко описание на симулацията на дискретни събития в OMNeT++:

- Симулацията е метод, който цели имитирането на реални обекти и системи, което позволява тяхното безопасно, евтино и гъвкаво изследване. OMNeT++ представлява симулатор на дискретни събития, предимно (но не единствено) насочен към симулация на мрежи.
- Основни елементи в OMNeT++ са: елементарни модули (simple module), сложни модули (compound module) и мрежа (network).
- Елементарните модули се свързват помежду си с връзки, описани на езика NED, давайки съставни модули, които от своя страна могат да се свързват с други елементарни и съставни модули, формирайки мрежа.



Фиг. 24. Модули в OMNeT++

Добавяне и стартиране на проекта в средата OMNeT++:

- File ->Import -> Existing Projects into Workspace
- Select archive file -> simple_queue.zip
- Finish.

Разгледайте сорс файловете и конфигурационните файлове на проекта.

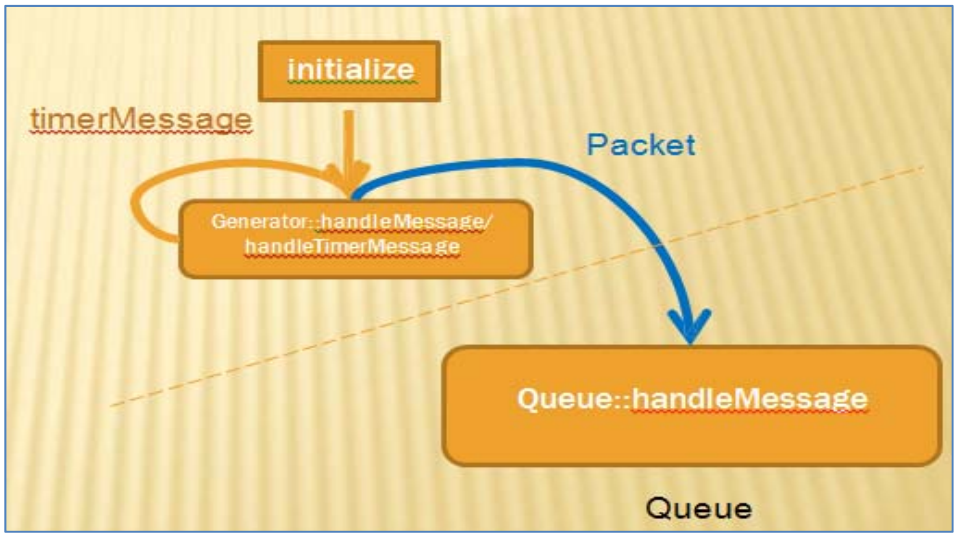
Компилирайте проекта, след което стартирайте модела от менюто:

Run -> Run As -> OMNeT++ Simulation

3. Методика за провеждане на експериментите

Системата, която е обект на изследване в това упражнение, се състои от три модула: генератор на пакети (входящ процес), опашка и сървър.

Генератор на пакети (входящ процес)

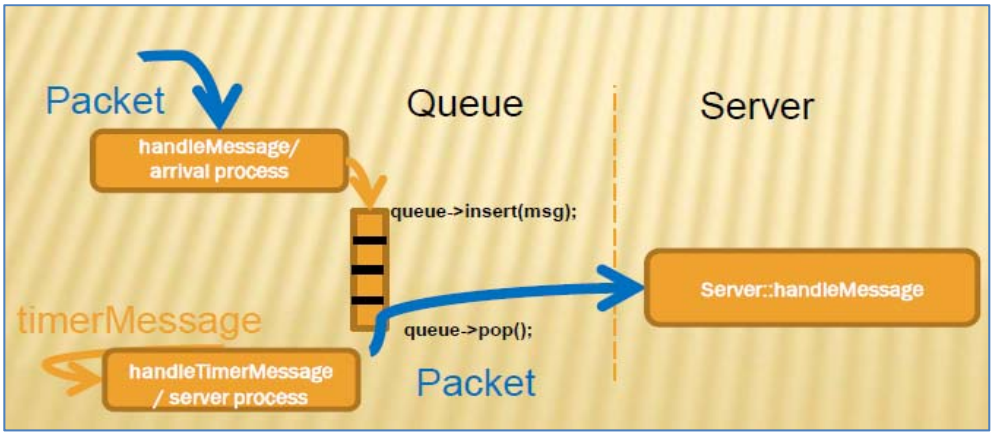


Фиг. 25. Модул „Генератор на пакети“

```
void Generator::initialize()
{
timerMessage = new cMessage("timer");
scheduleAt(simTime(), timerMessage);
}
.
.
void Generator::handleTimerMessage(cMessage *msg)
{
Packet * pPacket = new Packet ("Packet");
scheduleAt(simTime() + par("inter_arrival_time").doubleValue(),
timerMessage);
sendMessage(pPacket, 0, "out");
}
.
.
void Generator::handleMessage(cMessage *msg)
{
if (msg->isSelfMessage()) {
handleTimerMessage(msg);
} else {
ASSERT(0);
}
}
```

Фиг. 26. Код за модул „Генератор на пакети“

Опашка



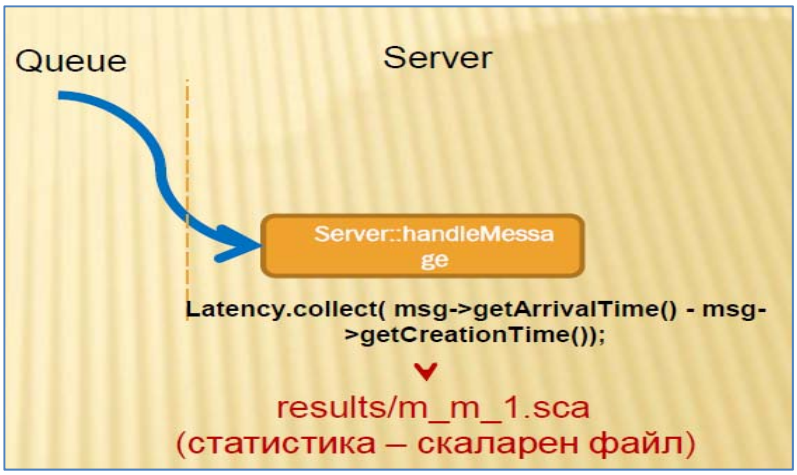
Фиг. 27. Модул „Опашка“

```
void Queue::initialize()
{
queue_length = par("queue_length").longValue();
queue = new cQueue;
Lenght.setName("queque lenght");
timerMessage = new cMessage("timer");
scheduleAt(simTime(), timerMessage);
}
void Queue::handleTimerMessage(cMessage *msg)
{
Packet *pPacket;
if(!queue->empty()) {
pPacket = (Packet *)queue->pop();
UpdateDisplay(queue->getLength());
sendDelayed(pPacket, 0, "out");
}
Lenght.collect(queue->getLength());
scheduleAt(simTime() + par("inter_arrival_time").doubleValue(), timerMessage);
}
void Queue::handleMessage(cMessage *msg)
{
if (msg->isSelfMessage()) { // server process
handleTimerMessage(msg);
} else { // arrival process
msg->setTimestamp();
queue->insert(msg);
UpdateDisplay(queue->getLength());
Lenght.collect(queue->getLength());
}
}
```

Фиг. 28. Код за модул „Опашка“

Функцията „handleMessage“ получава съобщения от два източника – от генератора (входящия процес) и от таймера, който генерира първото съобщение от конструктора на класа опашка „Queue::initialize“ и впоследствие от самия хендлър на таймера „handleTimerMessage“ чрез „scheduleAt(simTime() + par("inter_arrival_time").doubleValue(), timerMessage)“. Както се вижда, следващото таймер съобщение се планира да бъде изпратено след „inter_arrival_time“, което се взема всеки път от инициализационния файл omnetpp.ini: „**.queue.inter_arrival_time = exponential(0.1)“, като exponential(0.1) представлява функция, генерираща експоненциално разпределени случайни стойности, отговарящи на Марковски процес (M). Съобщенията, които не са „isSelfMessage“, идват от модула „Generator“ през канала „generator.out --> queue.in“, описан в „QueueNet.ned“.

Сървър



Фиг. 29. Модул „Сървър“

```
void Server::handleMessage(cMessage *msg)
{
if (msg->isSelfMessage()) {
ASSERT(0);
} else {
Latency.collect( msg->getArrivalTime() - msg-
>getCreationTime());
delete msg;
}
}
.
.
.
void Server::finish()
{
Latency.record();
}
```

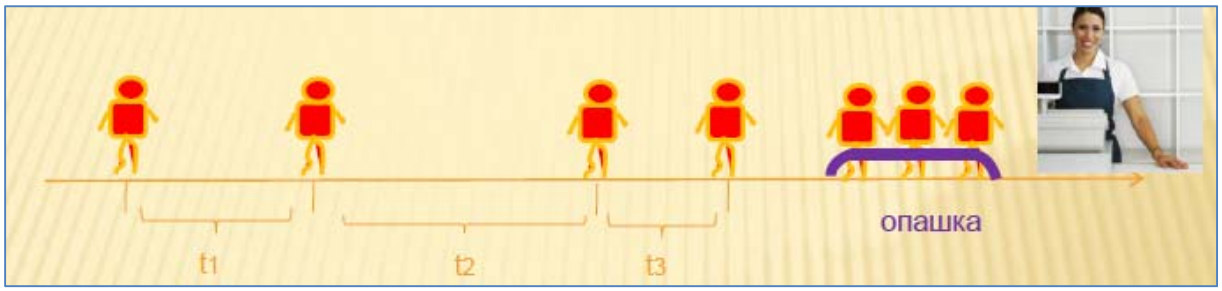
Фиг. 30. Код за модул „Сървър“

Разпределения на трафика

Едни от най-широко използваните разпределения на трафика са M (Markovian) – процес с експоненциално разпределение, и D (Degenerative) – процес с детерминистично разпределение (или на фиксирани интервали).

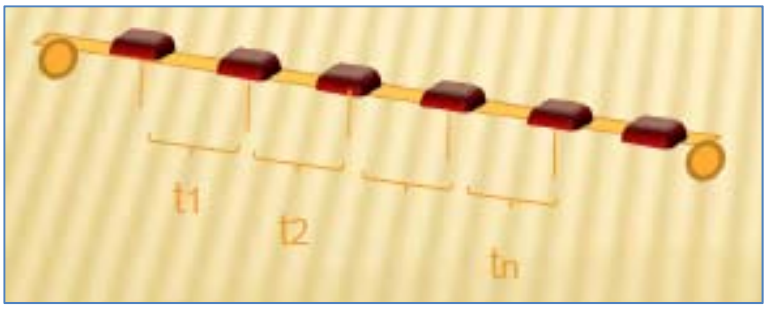
При M разпределението средният интервал между пристигащите заявки е tavg=1/λ, с функция на разпределение $f(t)=\lambda e^{-\lambda t}$.

Повечето случайни процеси от заобикалящия ни свят са от този тип, като заявките, които пристигат към даден сървър, са еквивалентни например на студенти, редящи се на опашката в стола, и много други.



Фиг. 31. Пример за M разпределение – студенти, редящи се на опашка в стола

При D разпределението имаме еднакви интервали t1 = t2 = tn = 1/ λ. Типичен пример е индустриален конвейер.



Фиг. 32. Пример за D разпределение – индустриален конвейер

4. Оформяне на протокол

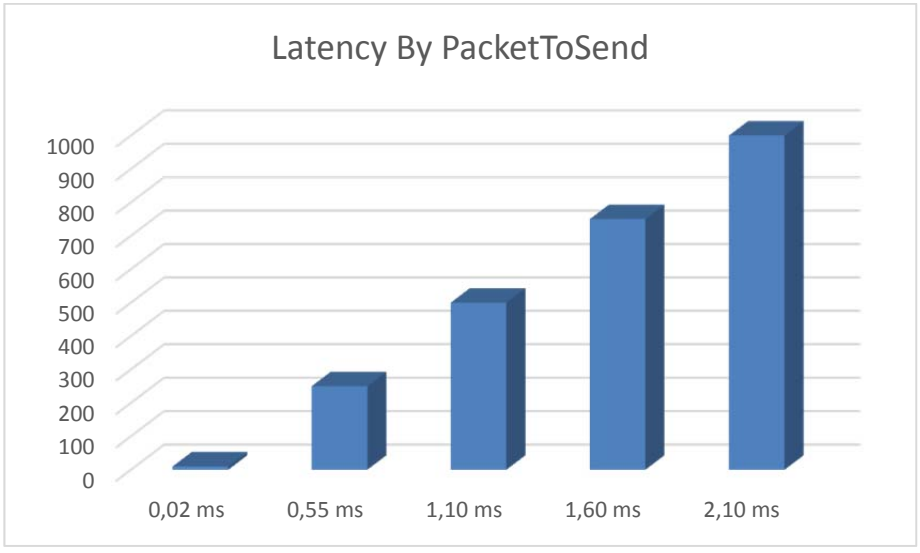
Да се снемат данни от симулациите за средното закъснение и броя чакащи заявки при вариращ интензитет на входните заявки и да се построят графики:

- При M (експоненциално) разпределение;
- При D (детерминистично) разпределение.

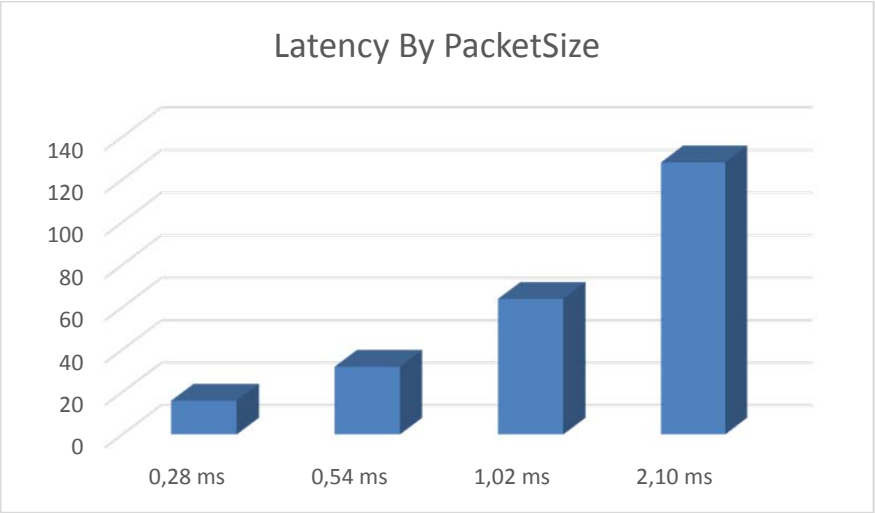
Получените от изследванията резултати да се оформят в протокол.

Таблица 1. Примерни изходни резултати от симулациите

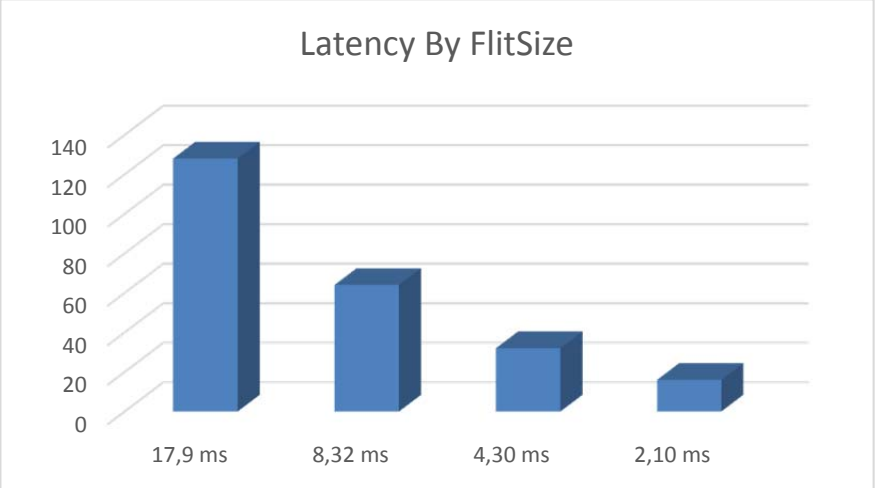
PacketSize	PacketToSend	FlitSize	latency
128	1000	16	***2,10 ms***
64	1000	16	***1,02 ms***
32	1000	16	***0,54 ms***
16	1000	16	***0,28 ms***
128	750	16	***1,60 ms***
128	500	16	***1,10 ms***
128	250	16	***0,55 ms***
128	10	16	***0,02 ms***
128	1000	32	***4,30 ms***
128	1000	64	***8,32 ms***
128	1000	128	***17,9 ms***



Фиг. 33. Зависимост на закъснението от броя на изпратените пакети



Фиг. 34. Зависимост на закъснението от големината на пакетите



Фиг. 35. Зависимост на закъснението от размера на флитовете