РЕАЛИЗАЦИЯ ИМИТАЦИОННОЙ МОДЕЛИ ЛОКАЛЬНОЙ СЕТИ В СРЕДЕ МОДЕЛИРОВАНИЯ *AnyLogic 5.5*

Александр Дудко

Институт транспорта и связи
Факультет компьютерных наук и электроники
ул. Ломоносова, 1, Рига, LV-1019, Латвия
E-mail: prg7af@yandex.ru

1. Постановка задачи

Моделирование – один из современных подходов в исследовании систем, которое активно используется в разных областях человеческой деятельности. Оно позволяет апробировать свои догадки и идеи сначала на виртуальной системе (модели) и лишь потом переходить на реальную реализацию. В данной статье в качестве предмета исследования выбрана локальная сеть со специфическим алгоритмом маршрутизации. По ходу изложения материала будет описан процесс создания модели, а затем готовая модель будет проанализирована, чтобы определить, насколько эффективно данная сеть будет справляться с доставкой сообщений.

Начнем с более формального описания задачи, которую предстоит выполнить. Предположим: существует сеть вычислительных машин с 5 серверами. Все серверы соединены друг с другом и образуют кольцо (рис. 1). К каждому серверу присоединено по 5 абонентов (клиентских машин), которые обмениваются сообщениями между собой. Каждый абонент имеет свой уникальный номер в сети. Длина передающихся сообщений распределена по нормальному закону распределения со средним значением 50 Кбайт и стандартным отклонением 0.2 Кбайт. Все сообщения перед передачей по сети разбиваются на пакеты длиной 2 Кбайта. Каждый пакет обеспечивается адресом абонента-получателя и передвигается по сети независимо от других. Сначала каждый пакет передается на сервер, непосредственно обслуживающий группу абонентов, в числе которых находится отправитель. Затем по каналам между серверами он идет до сервера, обслуживающего получателя, при этом на каждом сервере для дальнейшего продвижения будет выбираться тот канал, который наименее загружен. Когда пакет дойдет до нужного сервера, он будет отправлен к получателю. Когда все пакеты одного сообщения дойдут до адресата, это сообщение принимает свой изначальный вид (происходит сборка пакетов) и только после этого оно считается доставленным. Скорость передачи от абонента к серверу и от сервера к абоненту – 2400 байт в секунду. Скорость обмена данными между серверами – 48 Кбайт в секунду. Поток сообщений, поступающий от абонентов, – пуассоновский со средним значением 20 сообщений в час.

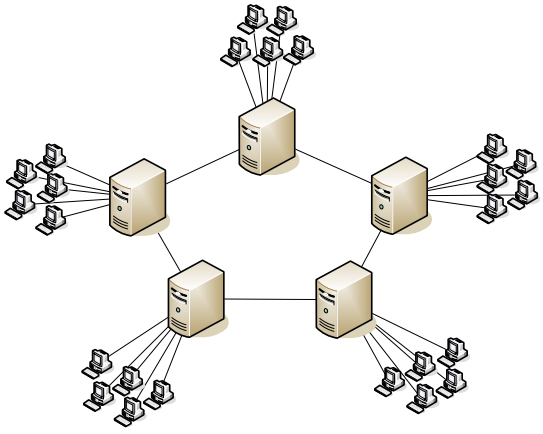

Рис. 1. Топология исследуемой сети

2. Описание допущений, сделанных на стадии построения концептуальной модели системы

Реальный мир слишком сложен для полной его имитации, поэтому при построении модели нам надо отбрасывать несущественные детали и идти на некоторые допущения. В ходе анализа реальной системы необходимо определить, какие допущения будут сделаны. При этом важно сохранить ключевые свойства системы, чтобы не произошло искажения и полученная модель отображала свойства исходной системы. На этой стадии строится концептуальная модель.

Наиболее полезная характеристика сети – это время доставки сообщения, и это означает, что нас в первую очередь интересуют задержки на разных участках. Допустим, что сервера совсем не задерживают пакеты при обработке, так как на самом деле время на перенаправление было бы ничтожно мало. Таким образом, время на доставку каждого пакета будет состоять из времени пересылок между узлами сети и времени ожидания в очередях. Задержка при пересылке будет моделироваться при помощи устройства с одним каналом обработки, что будет соответствовать захвату передающей среды и задержке на передачу. Такие устройства будем называть *transfer*-устройствами, и они будут размещаться между каждой соединенной парой узлов в сети. Концептуально это можно представить так, как показано на рис. 2, где заштрихованными прямоугольниками показаны *transfer*-устройства.

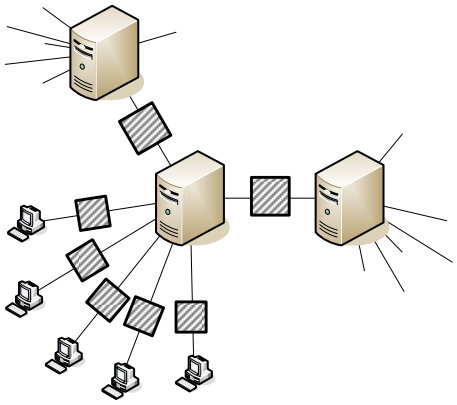

Рис. 2. Фрагмент концептуальной схемы модели

Сообщения генерируются в каждом узле-абоненте, после чего случайным образом выбирается адрес получателя и присваивается в атрибут транзакта-сообщения (сущность, которая будет представлять сообщение). Транзакт-сообщение разбивается на транзакты-пакеты, подобно тому, как это происходило бы на самом деле. Транзакты-пакеты пересылаются на ближайший сервер, а затем они «крутятся» в модели до тех пор, пока не достигнут желаемого сервера, а затем и пункта назначения, где им придется дождаться всех членов группы, чтобы восстановить транзакт-сообщение. При выходе сообщения из модели в таблице фиксируется время пребывания его в модели для сбора статистики.

3. Описание процесса построения модели в среде *AnyLogic 5.5*

Сегодня в мире существует немало систем для моделирования – от простых до больших и серьезных. Для моделирования нашей системы был выбран пакет *AnyLogic 5.5* – довольно новая и мощная система с визуальной средой для построения модели [1]. Также можно отметить, что *AnyLogic* предоставляет удобные механизмы для расширения.

При анализе концептуальной модели легко заметить множество повторяющихся частей. К сожалению, для многих из них невозможно подобрать готовые блоки пакета *AnyLogic*. Будет намного проще, если будут разработаны нужные нам блоки, инкапсулирующие всю логику работы в себе, к тому же среда это позволяет.

Для начала выделим в отдельный блок то самое *transfer*-устройство, о котором говорилось при построении концептуальной модели. Поскольку для нас не принципиально, будет ли передача происходить в полудуплексном (передача в разных направлениях происходит последовательно) или полнодуплексном режиме (одновременная передача в обоих направлениях),

сделаем наш блок *Transfer* полнодуплексным. Для организации задержки в каждом направлении будет использован стандартный блок *Server*, а для организации очереди – блок *Queue*:

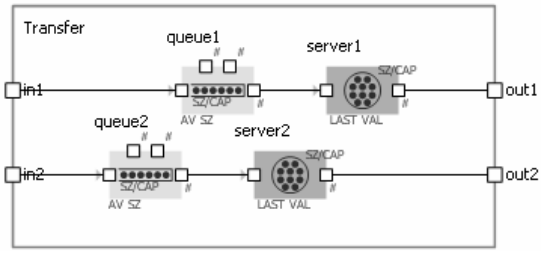

Рис. 3. Реализованный блок *Transfer*

Поскольку скорость передачи может быть разная, выделим свойство для этого блока. Называться оно будет *Speed*, обозначая скорость передачи в байтах в секунду. Тогда время на задержку в устройствах *server1* и *server2* будут вычисляться как размер обрабатываемого пакета (берется из свойства обрабатываемого транзакта-пакета) и делится на значение скорости.

Далее, хотелось бы также выделить в отдельный блок узел-абонент, который будет как генерировать, так и получать пакеты. На его долю достанется ответственность за разбиение сообщения на пакеты и наоборот – их сборка в сообщение. К сожалению, программа *AnyLogic* позволяет нам с помощью стандартного блока *Split* разбить транзакт на желаемое количество сообщений, однако нет возможности потом собрать их обратно вместе. Эту функциональность необходимо реализовать, и, чтобы не загромождать блок узла-абонента неочевидными блоками, сделаем еще один блок, который назовем *Assemble* (логика блока может быть заимствована из языка *GPSS*). Было предложено решение, изображенное на рис. 4.

В транзакт-сообщение нужно добавить свойство с уникальным идентификатором сообщения. При разбиении его на пакеты каждый пакет сохраняет в себе данный идентификатор. Это позволит находить все пакеты одного сообщения. Также при разбиении на пакеты в каждый из них следует поместить информацию о количестве пакетов, на которое было разделено сообщение. Тогда перед входом в блок *SelectOutput* в нашем блоке *Assemble* из обрабатываемого транзакта-пакета нужно взять уникальный идентификатор сообщения и занести в созданный нами массив информацию о числе оставшихся пакетов данного сообщения. Если такая информация в массиве уже была, то необходимо обновить ее – это поможет определить, какой пакет будет последним. Когда наконец придет последний пакет, его надо будет пометить специальным образом. Именно по такой пометке блок *SelectOutput* должен пропускать или отбрасывать в блок *Sink* приходящие транзакты-пакеты. Таким образом, на блок *Split* будут поступать лишь последние пришедшие пакеты от всех сообщений. Блок *Split* в этом месте выполняет не прямую свою роль по разделению, а стоит лишь для подмены транзакта-пакета на транзакт-сообщение. Это имитирует процесс сборки пакетов сообщения в само сообщение. Все ненужные транзакты отправляются на блок *Sink* для уничтожения.

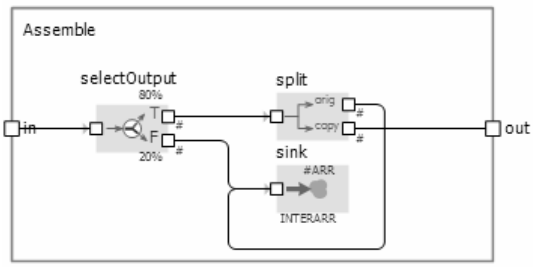

Рис. 4. Блок *Assemble*

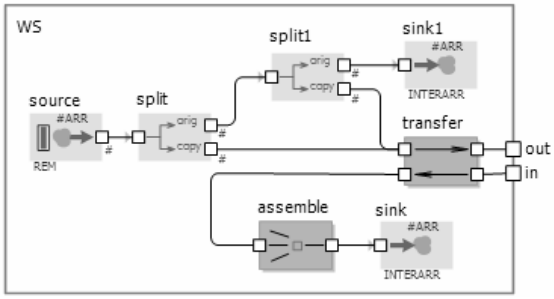

Рис. 5. Блок узла-абонента

Вернемся к блоку узел-абонент. Назовем его *Workstation* (*WS*). Используя только что созданный нами блок *Assemble*, задача становится очень простой. Созданный ранее блок *Transfer* также присутствует в этом блоке, хотя и не является частью данной подсистемы. Однако в

нашем случае каждый абонент сопряжен с линией связи, имитируемой *Transfer*'ом. Именно поэтому *Transfer* можно поместить внутрь *WS* для упрощения прочих подсистем модели. Если же в дальнейшем потребуется задавать разную скорость соединения для разных абонентов сети, то блок *Transfer* придется вынести наружу, для явного задания скорости. Блок *Split1*, как видно на рис. 5, опять же используется для подмены, а не для разбиения. На этот раз подменяется транзакт-сообщение на транзакт-пакет. В отличие от остальных пакетов разделенного сообщения, имеющих одинаковые размеры по 2 килобайта, пакет, вышедший из этого блока, может иметь меньший размер. Его размер рассчитывается как остаток от деления полного размера сообщения на 2048 байт (2 Кбайта). Таким образом, сумма размеров всех пакетов даст в точности размер сообщения, частями которого они являются.

В моделируемой сети, как известно, помимо абонентов присутствуют также серверы. Для них тоже необходим собственный блок с названием *Server*. Для целей маршрутизации каждый узел сети должен иметь адрес. Этой маршрутизацией занимаются именно серверы, поэтому у каждого сервера будет свой номер, который можно будет задать в виде свойства у создаваемого блока. Создадим свойство *ServerNumber*. Тогда для простоты обработки адрес назначения в пакетах будет состоять из двух частей: номер сервера и номер абонента, присоединенного к данному серверу. Структура блока *Server* представлена на рис. 6.

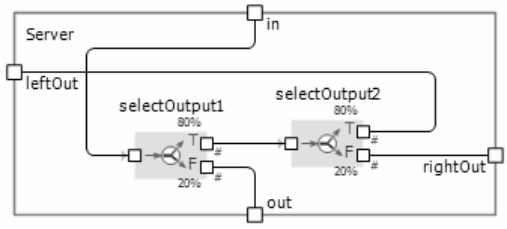

Рис 6. Структура блока *Server*

Все пакеты поступают на сервер через порт *in*, затем он должен перенаправить их по одному из трех выходов. Выход *out* направит пакет на один из непосредственно обслуживаемых абонентов, а выходы *leftOut* и *rightOut* – на левый и правый сервера соответственно. Блок *SelectOutput1* определяет, является ли текущий сервер сервером назначения для обрабатываемого пакета. Если это так, то пакет направится на выход *out* для дальнейшей маршрутизации к нужному абоненту. Иначе в блоке *SelectOutput2* выбирается канал дальнейшего продвижения: левый или правый в зависимости от их загруженности. Показателем загруженности является длина очереди на занятие линии связи. Это значит, что нашему блоку *Server* надо знать, с какими каналами передачи данных он соединен. Для этого введем еще два свойства для блока *Server*: *LeftTransfer* и *RightTransfer*, которые надо будет правильно указать при использовании нашего блока *Server*. Через эти свойства сервер сможет определять загруженность обоих каналов и выбирать наименее загруженный для правильного перенаправления пакетов.

Незатронутым остался вопрос выбора нужного абонента после достижения пакетом нужного сервера. Как уже упоминалось ранее, адрес назначения хранится в пакете в двух частях, одна из которых представляет номер абонента внутри сегмента, обслуживаемого сервером. Тогда выбор направления легко осуществить путем каскадного соединения стандартных блоков *SelectOutput*. Поскольку данная конструкция должна участвовать в модели не один раз, ее также можно заключить в пользовательский блок и даже включить туда все 5 абонентов – блоки *WS*, созданные ранее. Пусть данный блок носит название *WsSet* (рис. 7).

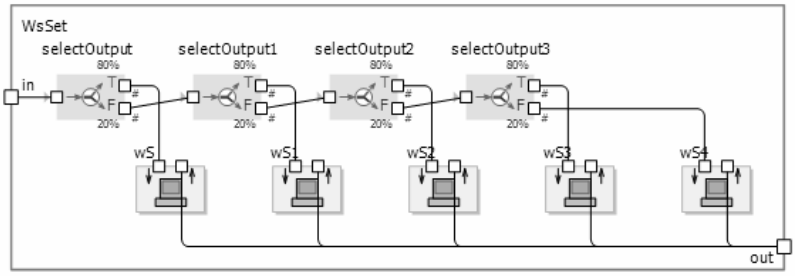

Рис 7. Пользовательский блок *WsSet*

Теперь у нас есть все необходимое для построения модели сети. Данная модель будет содержать 5 пользовательских блоков *Server*, соединенных между собой через блоки *Transfer* и образующих кольцо. Кроме того, к каждому из 5-ти блоков *Server* будет присоединен блок *WsSet* как обслуживаемый сегмент сети. Для каждого сервера указываем его левый и правый блоки *Transfer*, чтобы маршрутизация выполнялась правильно. В итоге получаем модель, изображенную на рис. 8.

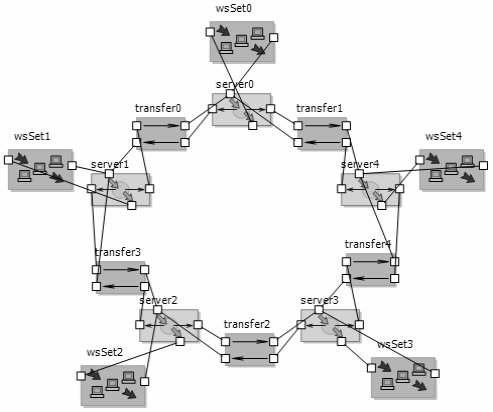

Рис. 8. Готовая модель

4. Демонстрационный пример

После получения готовой модели была проведена серия ее прогонов. Для сбора репрезентативного объема статистических данных о времени доставки сообщений модель прогонялась до тех пор, пока не было доставлено 100 000 сообщений, на что ушло более 200 часов модельного времени. После анализа полученных данных можно сказать, что 70% всех сообщений было доставлено менее чем за 24 секунды. Среднее время доставки составляет 26.7 сек. Гистограмма распределения времени доставки сообщений изображена на рис. 9.

Рис. 9. Гистограмма распределения времени доставки сообщения

Если принять во внимание, что сообщения в среднем имеют размер порядка 50 Кбайт, то затраты на пересылку данных между абонентом и сервером при скорости 2400 байт/с составляют 21.3 сек., учитывая оба направления. Значит, на маршрутизацию между серверами в среднем уходит чуть более 5 секунд, а в 70% случаев – менее 3 секунд. Безусловно, самое узкое место в данной сети – это скорость обмена данными между сервером и абонентом. При таких условиях предложенная топология и способ маршрутизации являются эффективными и могут успешно использоваться вместо других более сложных подходов при создании локальных сетей.

Используемые источники

1. Карпов Ю. Г. *Имитационное моделирование систем. Введение в моделирование с AnyLogic 5.* Издательство: *BHV*, 2006, 400 с.